CryptoAuthLib

20190830

Chapter 1

CryptoAuthLib - Microchip CryptoAuthentication Library

Introduction

This code base implements an object-oriented C library which supports Microchip CryptoAuth devices. The family of devices supported currently are:

- ATSHA204A
- ATECC108A
- ATECC508A
- ATECC608A

Online documentation is at https://microchiptech.github.io/cryptoauthlib/

Latest software and examples can be found at:

- http://www.microchip.com/SWLibraryWeb/product.aspx?product=CryptoAuthLib
- https://github.com/MicrochipTech/cryptoauthtools

Prerequisite skills:

- strong C programming and code reading
- Atmel Studio familiarity
- Knowledge of flashing microcontrollers with new code
- Familiarity with Microchip CryptoAuth device functionality

Prerequisite hardware to run CryptoAuthLib examples:

- ATSAMR21 Xplained Pro or ATSAMD21 Xplained Pro

- CryptoAuth Xplained Pro Extension or CryptoAuthentication SOIC Socket Board to accept SOIC parts

For most development, using socketed top-boards is preferable until your configuration is well tested, then you can commit it to a CryptoAuth Xplained Pro Extension, for example. Keep in mind that once you lock a device, it will not be changeable.

There are two major compiler defines that affect the operation of the library.

- ATCA_NO_POLL can be used to revert to a non-polling mechanism for device responses. Normally responses are polled for after sending a command, giving quicker response times. However, if ATCA_NO_POLL is defined, then the library will simply delay the max execution time of a command before reading the response.
- ATCA_NO_HEAP can be used to remove the use of malloc/free from the main library. This can be helpful for smaller MCUs that don't have a heap implemented. If just using the basic API, then there shouldn't be any code changes required. The lower-level API will no longer use the new/delete functions and the init/release functions should be used directly.

Examples

- Watch CryptoAuthLib Documents for new examples coming online.
- Node Authentication Example Using Asymmetric PKI is a complete, all-in-one example demonstrating all the stages of crypto authentication starting from provisioning the Crypto Authentication device ATECC608A/ATECC508A with keys and certificates to demonstrating an authentication sequence using asymmetric techniques. http://www.microchip.com/SWLibraryWeb/product.aspx?product=CryptoAuthLib

Release notes

Next Release

- Added big-endian architecture support
- Fixes to atcah_gen_dig() and atcah_nonce()

05/17/2019

- Added support for TNG devices (cert transforms, new API)
- atcab_write_pub_key() now works when the data zone is unlocked

03/04/2019

- mbed TLS wrapper added
- Minor bug fixes

01/25/2019

- Python JWT support
- Python configuration structures added
- Restructure of secure boot app

01/04/2019

- Added GCM functions
- Split AES modes into separate files
- Bug fix in SWI START driver

10/25/2018

- Added basic certificate functions to the python wrapper.
- Added Espressif ESP32 I2C driver.
- Made generic Atmel START drivers to support most MCUs in START.
- Added AES-CTR mode functions.
- Python wrapper functions now return single values with AtcaReference.
- Added mutex support to HAL and better support for freeRTOS.

08/17/2018

- Better support for multiple kit protocol devices

07/25/2018

- Clean up python wrapper

07/18/2018

- Added ATCA_NO_HEAP define to remove use of malloc/free.
- Moved PEM functions to their own file in atcacert.
- Added wake retry to accomodate power on self test delay.
- Added ca_cert_def member to atcacert_def_s so cert chains can be traversed as a linked list.

03/29/2018

- Added support for response polling by default, which will make commands return faster (define ATCA_NO↵ _POLL to use old delay method).
- Removed atcatls related files as they were of limited value.
- Test framework generates a prompt before locking test configuration.
- Test framework puts device to sleep between tests.

- Fixed mode parameter issue in atcah_gen_key_msg().
- ATECC608A health test error code added.

01/15/2018

- Added AES-128 CBC implementation using AES command
- Added AES-128 CMAC implementation using AES command

11/22/2017

- Added support for FLEXCOM6 on SAMG55 driver

11/17/2017

- Added library support for the ATECC608A device
- Added support for Counter command
- atca_basic functions and tests now split into multiple files based on command
- Added support for multiple base64 encoding rules
- Added support for JSON Web Tokens (jwt)
- Fixed atcab_write_enc() function to encrypt the data even when the device is unlocked
- Fixed atcab_base64encode_() for the extra newline
- Updated atcab_ecdh_enc() to work more consistently

07/01/2017

- Removed assumption of SN[0:1]=0123, SN[8]=EE. SN now needs to be passed in for functions in atca_host and atca_basic functions will now read the config zone for the SN if needed.
- Renamed atcab_gendig_host() to atcab_gendig() since it's not a host function. Removed original atcab_gendig(), which had limited scope.
- Fixed atcah_hmac() for host side HMAC calculations. Added atcab_hmac().
- Removed unnecessary ATCADeviceType parameters from some atca_basic functions.
- Added atcacert_create_csr() to create a signed CSR.
- New HAL implementation for Kit protocol over HID on Linux. Please see the Incorporating CryptoAuthLib in a Linux project using USB HID devices section in this file for more information.
- Added atcacert_write_cert() for writing certificates to the device.
- Added support for dynamic length certificate serial numbers in atcacert.
- Added atcab_write() for lower level write commands.
- Fixed atcah_write_auth_mac(), which had wrong OpCode.
- Added atcab_verify() command for lower level verify commands.
- Added atcab_verify_stored() for verifying data with a stored public key.

- Removed atcab_write_bytes_slot(). Use atcab_write_bytes_zone() instead.
- Modified atcab_write_bytes_zone() and atcab_read_bytes_zone() to specify a slot
- Added atcab_verify_validate() and atcab_verify_invalidate()
- Improvements to host functions to handle more cases.
- Added atcab_updateextra(), atcab_derive_key()
- Added support for more certificate formats.
- Added general purpose hardware SHA256 functions. See atcab_hw_sha2_256().
- Removed device specific config read/write. Generic now handles both.
- Removed unnecessary response parameter from lock commands.
- Enhanced and added unit tests.
- Encrypted read and write functions now handle keys with SlotConfig.NoMac set
- atcab_cmp_config_zone() handles all devices now.
- Fixed some edge cases in atcab_read_bytes_zone().
- Updated atSHA() to work with all devices.
- Fixed atcacert_get_device_locs() when using stored sn.

01/08/2016

- New HAL implementations for
  - Single Wire interface for SAMD21 / SAMR21
  - SAMV71 I2C HAL implementation
  - XMega A3Bu HAL implementation
- Added atcab_version() method to return current version string of libary to application
- New Bus and Discovery API
  - returns a list of ATCA device configurations for each CryptoAuth device found
  - currently implemented on SAMD21/R21 I2C, SAMV71
  - additional discovery implementations to come
- TLS APIs solidified and documented
- Added missing doxygen documentation for some CryptoAuthLib methods
- Stubs for HAL SPI removed as they are unused for SHA204A and ECC508A support
- bug fixes
- updated atcab_sha() to accept a variable length message that is > 64 bytes and not a multiple of 64 bytes (the SHA block size).
- refactored Cert I/O and Cert Data tests to be smaller
- 'uncrustify' source formatting
- published on GitHub

9/19/2015

- Kit protocol over HID on Windows
- Kit protocol over CDC on Linux
- TLS integration with ATECC508A
- Certificate I/O and reconstruction
- New SHA2 implementation
- Major update to API docs, Doxygen files found in cryptoauthlib/docs
- load cryptoauthlib/docs/index.html with your browser

Host Device Support

CryptoAuthLib will run on a variety of platforms from small micro-controllers to desktop host systems. The current list of hardware abstraction layer support includes:

Rich OS Hosts:

- Linux Kit Protocol over CDC USB
- Linux Kit Protocol over HID USB
- Linux I2C protocol.
- Windows Kit Protocol over CDC USB
- Windows Kit Protocol over HID USB

Microcontrollers:

- SAMD21 (I2C, SWI, and Bit Banging)
- SAMR21 (I2C and SWI)
- SAM4S (I2C)
- SAMV71 (I2C)
- SAMB11 (I2C)
- SAMG55 (I2C)
- AVR XMEGA A3BU (I2C and SWI)
- AVR AT90USB1287 (I2C and SWI)
- PIC32MX695F512H (I2C)

If you have specific microcontrollers or Rich OS platforms you need support for, please contact us through the Microchip portal with your request.

CryptoAuthLib Architecture

See the 'docs' directory of CryptoAuthLib for supporting documentation including architecture diagrams and more detailed usage docs.

The library is structured to support portability to:

- multiple hardware/microcontroller platforms
- multiple environments including bare-metal, Windows, and Linux OS
- multiple chip communication protocols (I2C, SPI, UART, and SWI)

All platform dependencies are contained within the HAL (hardware abstraction layer).

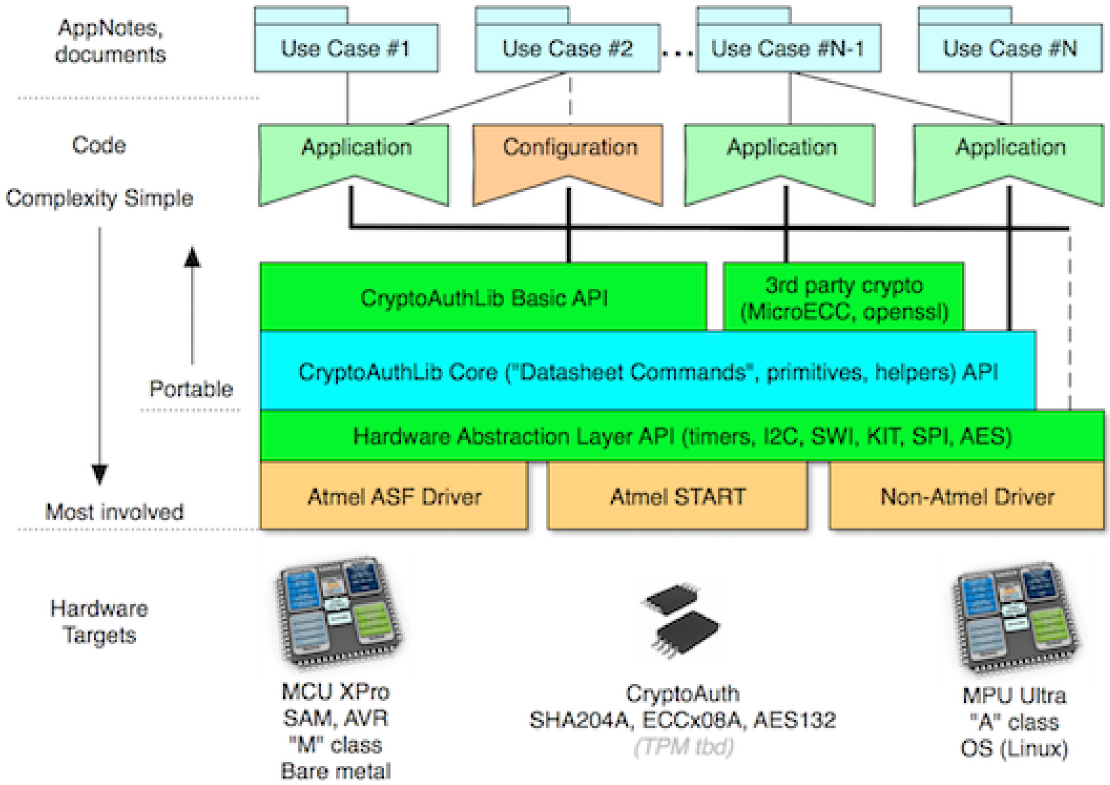

Figure 1.1 CryptoAuthLib Architecture

There are three primary object types in CryptoAuthLib:

- Device (ATCADevice)
- Command (ATCACommand)
- Interface (ATCAIface)

ATCADevice is a composite object made up of ATCACommand ATCAIface.

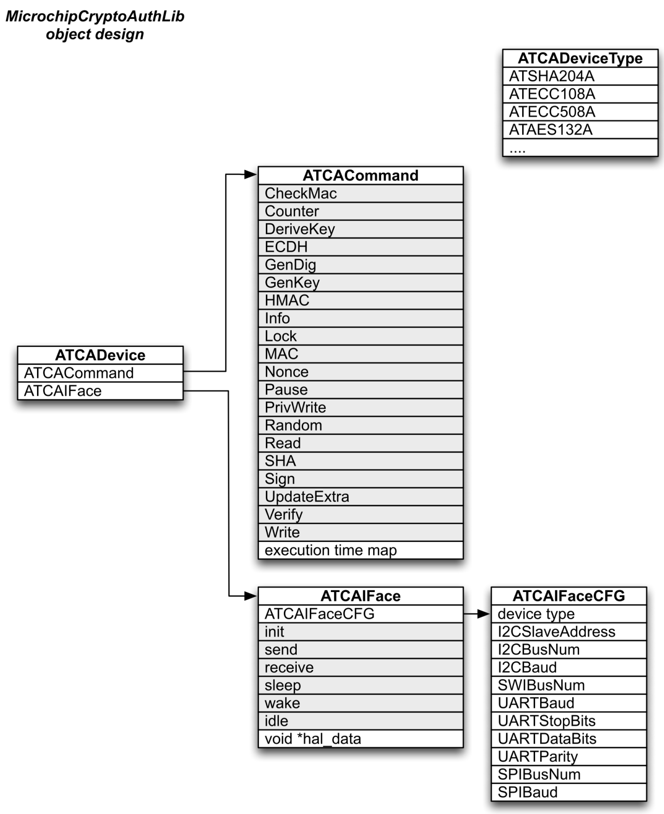

Figure 1.2 ATCADevice

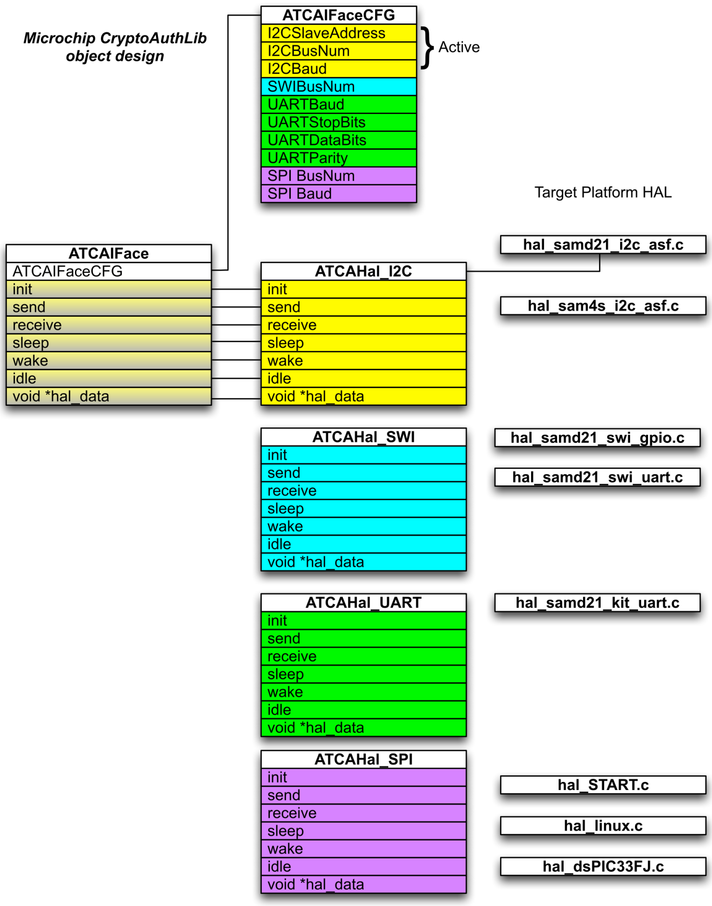

Figure 1.3 ATCAIface

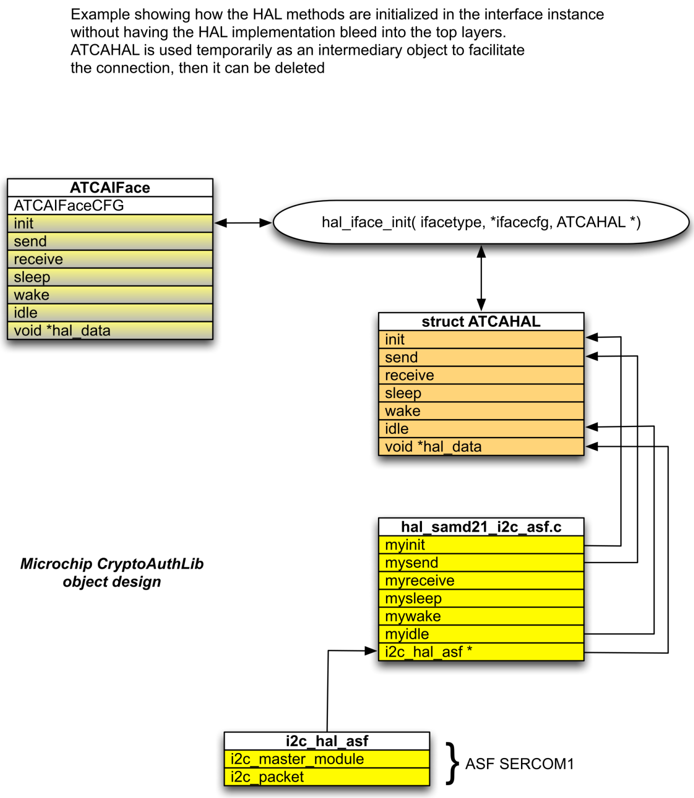

Figure 1.4 Hardware abstraction layer

Currently, the vast majority of testing has been performed on:

- ATSAMR21 Xplained Pro
- ATSAMD21 Xplained Pro
- ATSAMV71 Xplained Pro
- Windows (kit protocol HID)

These host containers implement a host test environment and test console to exercise tests. They presume that a CryptoAuth Xplained Pro or other I2C socket for an ATECC608A/ATECC508A/ATECC108A/ATSHA204A are connected to the I2C pins of the host Xplained Pro development board or in the case of windows is using a HID connection to an ATCK101 or ATCK590.

The unit tests and basic tests exercise the core datasheet commands of the device as well as the more convenient, basic API methods.

If you need an example of how to use a command, these hosts and tests are a good place to reference.

Object Architecture

Even though this is a C library, it follows object-oriented design patterns.

An object is minimally defined to be data and the actions which operate on that data.

Each CryptoAuth device is a composite object, a structure which includes the command table (list of commands) which are valid for the device, and the data used to hold the state of that device.

ATCADevice is the object which represents the Microchip CryptAuth device

ATCACommand is the object which represents the valid methods of the Device.

ATCAInterface is the physical interface object (I2C or SWI instance). Currently, each Device may have a single OATCAInterface.

ATCADevice represents an ATSHA or ATECC family device.

In order to add new protocol support for a platform, you provide a HAL (hardware abstraction layer) C file for the protocol and target. In your project's IDE or Makefile, you select which HAL support you need for the hardware configuration. Generally, there are separate files for each protocol and platform combination - (ie: samd21_i2c_↩ asf.c would target SAMD21 MCUs with I2C using the ASF low-level driver support.)

Directory Structure

```
docs - AppNotes and Doxygen HTML documentation for the library API. Load "docs/html/index.html" in your
       browser
lib - primary library source code
lib/atcacert - certificate data and i/o methods
lib/basic - the Basic API way to access the core classes
lib/crypto - Software crypto implementations (primarily SHA1 and SHA256)
lib/hal - hardware abstraction layer code for supporting specific platforms
lib/host - support functions for common host-side calculations
lib/jwt - json web token functions
test - unit tests. See test/cmd-processor.c for main() implementation.
For production code, test directories should be excluded by not compiling it
into a project, so it is up to the developer to include or not as needed.  Test
code adds significant bulk to an application - it’s not intended to be included
in production code.
```

Tests

There is a set of unit tests found in the test directory which will at least partially demonstrate the use of the objects. Some tests may depend upon a certain device being configured in a certain way and may not work for all devices or specific configurations of the device.

The test/cmd-processor.c file contains a main() function for running the tests. It implements a command-line interface. Typing help will bring up the list of commands available.

One first selects a device type, with one of the following commands:

- 204 (ATSHA204A)
- 108 (ATECC108A)
- 508 (ATECC508A)
- 608 (ATECC608A)

From there the following unit test sweets are available:

- unit (test command builder functions)
- basic (test basic API functions)
- cio (test certification i/o functions)
- cd (test certificate data functions)
- util (test utility functions)
- crypto (test software crypto functions)

Unit tests available depend on the lock level of the device. The unit tests won't lock the config or data zones automatically to allow retesting at desired lock levels. Therefore, some commands will need to be repeated after locking to exercise all available tests.

Starting from a blank device, the sequence of commands to exercise all unit tests is:

```
unit
basic
lockcfg
unit
basic
lockdata
unit
basic
cio
cd
util
crypto
```

Using CryptoAuthLib (Microchip CryptoAuth Library)

Using a new library is often easier when you can load an example and see how it works. We've provided examples in the form of "host containers" which are host projects that incorporate CryptoAuthLib and target various processors or communication APIs.

We maintain host test containers for each of the HAL layers we support. We've published the host container for SAMD21 which demonstrates a simple console interface to invoke test runners.

Look for SAMD21 Unit Tests CryptoAuthLib at http://www.microchip.com/SWLibraryWeb/product.aspx?product=CryptoAuthLib

The best way to learn how to use CryptoAuthLib is to study the host test projects that exercise the library and ATECC and ATSHA devices.

New examples will be forthcoming as the software matures. Continue checking the CryptoAuthentication web page for new updates.

Using Git to Incorporate CryptoAuthLib as a Submodule

You can include this project in your own project under git.

Using CryptoAuthLib as a git submodule, you can maintain your application separately from CryptoAuthLib.

If your project is already in git but you haven't yet intergrated CryptoAuthLib, change to the directory where you want to put CryptoAuthLib

```
git submodule add -b master <giturl to CryptoAuthLib>
```

This adds CryptoAuthLib as a subdirectory and separate git repo within your own project. Changes and commits to your project vs CryptoAuthLib will remain separated into each respective repository.

If there is a project you want to checkout that already incorporates CryptoAuthLib as a submodule if you clone the repo that incorporates CryptoAuthLib, after cloning, you'll still need to fill out the CryptoAuthLib submodule after cloning:

```
git submodule init
git submodule update --remote
cd cryptoauthlib
git checkout master
```

Now that CryptoAuthLib is a full-fledged submodule in your git project, in order to easily add it to your project within Atmel Studio, please see this tip

Incorporating CryptoAuthLib in a project

1) In your Makefile or IDE, choose the HAL support you need from the HAL directory and exclude other HAL files from your project.

2) For I2C interfaces, define the symbol ATCA_HAL_I2C in your compiler's symbol definitions. This will hook up the CryptoAuthLib interface class with your HAL implementation of I2C methods.

3) HAL implementations for CDC and HID interfaces to the ATCK101 are also included for use with Windows or Linux versions of the test host.

Incorporating CryptoAuthLib in a Linux project using USB HID devices

The Linux HID HAL files use the Linux udev development software package.

To install the udev development package under Ubuntu Linux, please type the following command at the terminal window:

```
sudo apt-get install libudev-dev
```

This adds the udev development development software package to the Ubuntu Linux installation.

The Linux HID HAL files also require a udev rule to be added to change the permissions of the USB HID Devices. Please add a new udev rule for the Microchip CryptoAuth USB devices.

```
cd /etc/udev/rules.d
sudo touch mchp-cryptoauth.rules
```

Edit the mchp-cryptoauth.rules file and add the following line to the file:

```
SUBSYSTEM=="hidraw", ATTRS{idVendor}=="03eb", ATTRS{idProduct}=="2312", MODE="0666"
```

Chapter 2

License

mbedTLS Interface Functions that enable mbedtls objects to use cryptoauthlib functions

Replace mbedTLS ECDSA Functions with hardware acceleration & hardware key security.

Replace mbedTLS ECDH Functions with hardware acceleration & hardware key security.

Copyright

Subject to your compliance with these terms, you may use Microchip software and any derivatives exclusively with Microchip products. It is your responsibility to comply with third party license terms applicable to your use of third party software (including open source software) that may accompany Microchip software.

THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.

Replace mbedTLS ECDSA Functions with hardware acceleration & hardware key security mbedTLS Interface Functions that enable mbedtls objects to use cryptoauthlib functions

Copyright

(c) 2015-2019 Microchip Technology Inc. and its subsidiaries.

Subject to your compliance with these terms, you may use Microchip software and any derivatives exclusively with Microchip products. It is your responsibility to comply with third party license terms applicable to your use of third party software (including open source software) that may accompany Microchip software.

THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE SOFTWARE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.

Chapter 3

basic directory - Purpose

The purpose of this directory is to contain the files implementing the APIs for a basic interface to the core Crypto↩AuthLib library.

High-level functions like these make it very convenient to use the library when standard configurations and defaults are in play. They are the easiest to use when developing examples or trying to understand the "flow" of an authentication operation without getting overwhelmed by the details.

This makes simple jobs easy and if you need more sophistication and power, you can employ the full power of the CryptoAuthLib object model.

See the Doxygen documentation in cryptoauthlib/docs for details on the API of the Basic commands.

Chapter 4

crypto directory - Purpose

This directory contains software implementations of cryptographic functions. The functions at the base level are wrappers that will point to the final implementations of the software crypto functions.

Chapter 5

HAL Directory - Purpose

This directory contains all the Hardware Abstraction Layer (HAL) files used to adapt the upper levels of atca-ng and abstractions to physical hardware.

HAL contains physical implementations for I2C, SWI, SPI, UART and timers for specific hardware platforms.

Include just those HAL files you require based on platform type.

CryptoAuthLib Supported HAL Layers

HAL Layers files are combined into groups. Initial group is generic files that are typically included in a project. Files are then broken out by uController Family and or Operating System Interface.

Protocol Files	Interface	Files	API	Notes
atca		atca_hal.c/h		For all projects
kit protocol		kit_protocol.c/h		For all Kit Protocol projects
		kit_phy.h		
		hal_i2c_bitbang.c/h	ASF	For all I2C Bitbang projects
		hal_swi_bitbang.c/h	ASF	For all SWI Bitbang projects

Most microcontrollers supported by `Atmel START` have generic drivers depending on the interface.

START Micros	Interface	Files	API	Notes
		hal_timer_start.c	START	Timer implementation
	I2C	hal_i2c_start.c/h	START	
	SWI	swi_uart_start.c/h	START	SWI using UART

AVR Micros	Interface	Files	API	Notes
at90usb1287	I2C	hal_at90usb1287_i2c_asf.c/h	ASF	
		hal_at90usb1287_timer_asf.c	ASF	
	SWI	swi_uart_at90usb1287_asf.c/h	ASF	
xmega_a3bu	I2C	hal_xmega_a3bu_i2c_asf.c/h	ASF	
		hal_xmega_a3bu_timer_asf.c	ASF	
	SWI	swi_uart_xmaga_a3bu_asf.c/h	ASF	

SAM Micros	Interface	Files	API	Notes
sam4s	I2C	hal_sam4s_i2c_asf.c/h	ASF	
		hal_sam4s_timer_asf.c	ASF	
samb11	I2C	hal_samb11_i2c_asf.c/h	ASF	
		hal_samb11_timer_asf.c	ASF	
samd21	I2C	hal_samd21_i2c_asf.c/h	ASF	
		hal_samd21_timer_asf.c	ASF	For all samd21 ASF projects
samd21	I2C	i2c_bitbang_samd21.c/h	ASF	For samd21 I2C bitbang projects
samd21	SWI	swi_bitbang_samd21.c/h	ASF	For samd21 SWI bitbang projects
samd21	SWI	swi_uart_samd21.c/h	ASF	For samd21 SWI uart projects
samg55	I2C	hal_samg55_i2c_asf.c/h	ASF	
		hal_samg55_timer_asf.c	ASF	
samv71	I2C	hal_samv71_i2c_asf.c/h	ASF	
		hal_samv71_timer_asf.c	ASF	

PIC Micros	Interface	Files	API	Notes
pic32mx695f512h	I2C	hal_pic32mx695f512h.c/h	plib.h	For pic32mx695f512h Standalone Mplab projects
		hal_pic32mx695f512h_timer.c	plib.h	For pic32mx695f512h Standalone Mplab projects
PIC32MZ2048	I2C	hal_pic32mz2048efm_i2c.c/h		
		hal_pic32mz2048efm_timer.c		

OS	Interface	Files	API	Notes
MS Windows	kit-cdc	hal_win_kit_cdc.c/h	windows.h	For all windows USB CDC projects
MS Windows	kit-hid	hal_win_kit_hid.c/h	windows.h	For all windows USB HID projects
			setupapi.h	
MS Windows		hal_win_timer.c	windows.h	For all windows projects
Linux	I2C	hal_linux_i2c_userspace.c/h	i2c-dev	
Linux	kit-cdc	hal_linux_kit_cdc.c/h	fopen	For USB Linux CDC projects
Linux	kit-hid	hal_linux_kit_hid.c/h	udev	For USB Linux HID Projects
Linux/Mac		hal_linux_timer.c		For all Linux/Mac projects
All	kit-hid	hal_all_platforms_kit_hidapi.c/h	hidapi	Works for Windows, Linux, and Mac
freeRTOS		hal_freertos.c		freeRTOS common routines

Chapter 6

IP Protection with Symmetric Authentication

The IP protection can be easily integrated to the existing projects.The user project should include symmetric_authentication.c & symmetric_authentication.h files which contains the api

- **symmetric_authenticate()** - For Performing the authentication between host & device.

User Considerations

- The user should take care on how the master key should be stored on the MCU side.
- The api's in the file doesn't do the provisioning of the chip and user should take care of the provisioning.

With the provisioned cryptoauthentication device and after doing the cryptoauthlib initialisation,user should only be calling the function symmetric_authenticate() with its necessary parameters for the authentication. The returned authentication status should be used in the application.

Examples

For more information about IP protection and its example project refer `Microchip github`

Chapter 7

app directory - Purpose

This directory is for application specific implementation of various use cases.

Methods in this directory provide a simple API to perform potentially complex combinations of calls to the main library or API.

Chapter 8

Secure boot using ATECC608A

The SecureBoot command is a new feature on the ATECC608A device compared to earlier CryptoAuthentication devices from Microchip. This feature helps the MCU to identify fraudulent code installed on it. When this feature is implemented, the MCU can send a firmware digest and signature to the ATECC608A. The ATECC608A validates this information (ECDSA verify) and responds to host with a yes or no answer.

The ATECC608A provides options to reduce the firmware verification time by storing the signature or digest after a good full verification (FullStore mode of the SecureBoot command).

- When the ATECC608A stores the digest (SecureBootMode is FullDig), the host only needs to send the firmware digest, which is compared to the stored copy. This skips the comparatively lengthy ECDSA verify, speeding up the secure boot process.
- When the ATECC608A stores the signature (SecureBootMode is FullSig), the host only needs to send the firmware digest, which is verified against the stored signature using ECDSA. This saves time by not needing to send the signature in the command over the bus.

The ATECC608A also provides wire protection features for the SecureBoot command, which can be used to encrypt the digest being sent from the host to the ATECC608A and add a MAC to the verify result coming back to the host so it can't be forced to a success state. This feature makes use of a shared secret between the host and ATECC608A, called the IO protection key.

The secure boot feature can be easily integrated to an existing project. The project should include the following files from the secure_boot folder:

- secure_boot.c
- secure_boot.h
- secure_boot_memory.h
- io_protection_key.h

The project should also implement the following platform-specific APIs:

- secure_boot_init_memory()
- secure_boot_read_memory()
- secure_boot_deinit_memory()

- secure_boot_mark_full_copy_completion()
- secure_boot_check_full_copy_completion()
- io_protection_get_key()
- io_protection_set_key()

The project can set the secure boot configuration with the following defines:

- SECURE_BOOT_CONFIGURATION
- SECURE_BOOT_DIGEST_ENCRYPT_ENABLED
- SECURE_BOOT_UPGRADE_SUPPORT

The secure boot process is performed by initializing CryptoAuthLib and calling the secure_boot_process() function.

Implementation Considerations

- Need to perform SHA256 calculations on the host. CryptoAuthLib provides a software implementation in lib/crypto/atca_crypto_sw_sha2.c
- When using the wire protection features:
  - The host needs to be able to generate a nonce (number used once). This is the NumIn parameter to the Nonce command that is sent before the SecureBoot command. The ATECC608A can not be used to generate NumIn, but it should come from a good random or non-repeating source in the host.
  - If the host has any protected internal memory, it should be used to store its copy of the IO protection key.
- Secure boot depends on proper protections of the boot loader code in the host. If the code can be easily changed, then the secure boot process can be easily skipped. Boot loader should ideally be stored in an immutable (unchangeable) location like a boot ROM or write-protected flash.
- Note that these APIs don't provision the ATECC608A. They assume the ATECC608A has already been configured and provisioned with the necessary keys for secure boot.

Examples

For more information about secure boot, please see the example implementation project and documentation at: https://github.com/MicrochipTech/cryptoauth_usecase_secureboot

Chapter 9

TNG Functions

This folder has a number of convenience functions for working with TNG devices (currently ATECC608A-MAHTN-T).

These devices have standard certificates that can be easily read using the functions in tng_atcacert_client.h

Chapter 10

Module Index

10.1 Modules

Here is a list of all modules:

Chapter 11

Data Structure Index

11.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 12

File Index

12.1 File List

Here is a list of all files with brief descriptions:

Chapter 13

Module Documentation

13.1 Configuration (cfg_)

Logical device configurations describe the CryptoAuth device type and logical interface.

Variables

- ATCAIfaceCfg cfg_ateccx08a_i2c_default
  default configuration for an ECCx08A device
- ATCAIfaceCfg cfg_ateccx08a_swi_default
  default configuration for an ECCx08A device on the logical SWI bus over UART
- ATCAIfaceCfg cfg_ateccx08a_kitcdc_default
  default configuration for Kit protocol over the device's async interface
- ATCAIfaceCfg cfg_ateccx08a_kithid_default
  default configuration for Kit protocol over the device's async interface
- ATCAIfaceCfg cfg_atsha204a_i2c_default
  default configuration for a SHA204A device on the first logical I2C bus
- ATCAIfaceCfg cfg_atsha204a_swi_default
  default configuration for an SHA204A device on the logical SWI bus over UART
- ATCAIfaceCfg cfg_atsha204a_kitcdc_default
  default configuration for Kit protocol over the device's async interface
- ATCAIfaceCfg cfg_atsha204a_kithid_default
  default configuration for Kit protocol over the device's async interface

13.1.1 Detailed Description

Logical device configurations describe the CryptoAuth device type and logical interface.

13.1.2 Variable Documentation

13.1.2.1 cfg_ateccx08a_i2c_default

```
ATCAIfaceCfg cfg_ateccx08a_i2c_default
```

Initial value:

```
= {
    .iface_type                = ATCA_I2C_IFACE,
    .devtype                   = ATECC608A,
    {
        .atcai2c.slave_address  = 0xC0,
        .atcai2c.bus            = 2,
        .atcai2c.baud           = 400000,

    },
    .wake_delay                = 1500,
    .rx_retries                = 20
}
```

default configuration for an ECCx08A device

default configuration for an ECCx08A device on the first logical I2C bus

13.1.2.2 cfg_ateccx08a_kitcdc_default

```
ATCAIfaceCfg cfg_ateccx08a_kitcdc_default
```

Initial value:

```
= {
    .iface_type            = ATCA_UART_IFACE,
    .devtype               = ATECC608A,
    {
        .atcauart.port      = 0,
        .atcauart.baud      = 115200,
        .atcauart.wordsize  = 8,
        .atcauart.parity    = 2,
        .atcauart.stopbits  = 1,
    },
    .rx_retries            = 1,
}
```

default configuration for Kit protocol over the device's async interface

default configuration for Kit protocol over a CDC interface

13.1.2.3 cfg_ateccx08a_kithid_default

```
ATCAIfaceCfg cfg_ateccx08a_kithid_default
```

Initial value:

```
= {
    .iface_type                  = ATCA_HID_IFACE,
    .devtype                     = ATECC608A,
    .atcahid.dev_interface       = ATCA_KIT_AUTO_IFACE,
    .atcahid.dev_identity        = 0,
    .atcahid.idx                 = 0,
    .atcahid.vid                 = 0x03EB,
    .atcahid.pid                 = 0x2312,
    .atcahid.packetsize          = 64,
}
```

default configuration for Kit protocol over the device's async interface

default configuration for Kit protocol over a HID interface

13.1.2.4 cfg_ateccx08a_swi_default

```
ATCAIfaceCfg cfg_ateccx08a_swi_default
```

Initial value:

```
= {
    .iface_type         = ATCA_SWI_IFACE,
    .devtype            = ATECC608A,
    {
        .atcaswi.bus    = 4,
    },
    .wake_delay         = 1500,
    .rx_retries         = 10
}
```

default configuration for an ECCx08A device on the logical SWI bus over UART

13.1.2.5 cfg_atsha204a_i2c_default

```
ATCAIfaceCfg cfg_atsha204a_i2c_default
```

Initial value:

```
= {
    .iface_type                 = ATCA_I2C_IFACE,
    .devtype                    = ATSHA204A,
    {
        .atcai2c.slave_address  = 0xC8,
        .atcai2c.bus            = 2,
        .atcai2c.baud           = 400000,
    },
    .wake_delay                 = 2560,
    .rx_retries                 = 20
}
```

default configuration for a SHA204A device on the first logical I2C bus

13.1.2.6 cfg_atsha204a_kitcdc_default

```
ATCAIfaceCfg cfg_atsha204a_kitcdc_default
```

Initial value:

```
= {
    .iface_type             = ATCA_UART_IFACE,
    .devtype                = ATSHA204A,
    {
        .atcauart.port      = 0,
        .atcauart.baud      = 115200,
        .atcauart.wordsize  = 8,
        .atcauart.parity    = 2,
        .atcauart.stopbits  = 1,
    },
    .rx_retries             = 1,
}
```

default configuration for Kit protocol over the device's async interface

default configuration for Kit protocol over a CDC interface

13.1.2.7 cfg_atsha204a_kithid_default

```
ATCAIfaceCfg cfg_atsha204a_kithid_default
```

Initial value:
```
= {
    .iface_type             = ATCA_HID_IFACE,
    .devtype                = ATSHA204A,
    .atcahid.dev_interface = ATCA_KIT_AUTO_IFACE,
    .atcahid.dev_identity  = 0,
    .atcahid.idx            = 0,
    .atcahid.vid            = 0x03EB,
    .atcahid.pid            = 0x2312,
    .atcahid.packetsize     = 64,
}
```

default configuration for Kit protocol over the device's async interface

default configuration for Kit protocol over a HID interface for SHA204

13.1.2.8 cfg_atsha204a_swi_default

```
ATCAIfaceCfg cfg_atsha204a_swi_default
```

Initial value:
```
= {
    .iface_type        = ATCA_SWI_IFACE,
    .devtype           = ATSHA204A,
    {
        .atcaswi.bus   = 4,
    },
    .wake_delay        = 2560,
    .rx_retries        = 10
}
```

default configuration for an SHA204A device on the logical SWI bus over UART

13.2 ATCACommand (atca_)

CryptoAuthLib command builder object, ATCACommand. Member functions for the ATCACommand object.

Data Structures

- struct atca_command
  atca_command is the C object backing ATCACommand.
- struct ATCAPacket

Macros

- #define ATCA_CMD_SIZE_MIN ((uint8_t)7)
  minimum number of bytes in command (from count byte to second CRC byte)
- #define ATCA_CMD_SIZE_MAX ((uint8_t)4 ∗ 36 + 7)
  maximum size of command packet (Verify)
- #define CMD_STATUS_SUCCESS ((uint8_t)0x00)
  status byte for success
- #define CMD_STATUS_WAKEUP ((uint8_t)0x11)
  status byte after wake-up
- #define CMD_STATUS_BYTE_PARSE ((uint8_t)0x03)
  command parse error
- #define CMD_STATUS_BYTE_ECC ((uint8_t)0x05)
  command ECC error
- #define CMD_STATUS_BYTE_EXEC ((uint8_t)0x0F)
  command execution error
- #define CMD_STATUS_BYTE_COMM ((uint8_t)0xFF)
  communication error

Typedefs

- typedef struct atca_command ∗ ATCACommand

Functions

- ATCA_STATUS initATCACommand (ATCADeviceType device_type, ATCACommand ca_cmd)
  Initializer for ATCACommand.
- ATCACommand newATCACommand (ATCADeviceType device_type)
  constructor for ATCACommand
- void deleteATCACommand (ATCACommand ∗ca_cmd)
  ATCACommand destructor.
- ATCA_STATUS atCheckMAC (ATCACommand ca_cmd, ATCAPacket ∗packet)
  ATCACommand CheckMAC method.
- ATCA_STATUS atCounter (ATCACommand ca_cmd, ATCAPacket ∗packet)
  ATCACommand Counter method.
- ATCA_STATUS atDeriveKey (ATCACommand ca_cmd, ATCAPacket ∗packet, bool has_mac)
  ATCACommand DeriveKey method.
- ATCA_STATUS atECDH (ATCACommand ca_cmd, ATCAPacket ∗packet)

ATCACommand ECDH method.
- ATCA_STATUS atGenDig (ATCACommand ca_cmd, ATCAPacket *packet, bool is_no_mac_key)

  ATCACommand Generate Digest method.
- ATCA_STATUS atGenKey (ATCACommand ca_cmd, ATCAPacket *packet)

  ATCACommand Generate Key method.
- ATCA_STATUS atHMAC (ATCACommand ca_cmd, ATCAPacket *packet)

  ATCACommand HMAC method.
- ATCA_STATUS atInfo (ATCACommand ca_cmd, ATCAPacket *packet)

  ATCACommand Info method.
- ATCA_STATUS atLock (ATCACommand ca_cmd, ATCAPacket *packet)

  ATCACommand Lock method.
- ATCA_STATUS atMAC (ATCACommand ca_cmd, ATCAPacket *packet)

  ATCACommand MAC method.
- ATCA_STATUS atNonce (ATCACommand ca_cmd, ATCAPacket *packet)

  ATCACommand Nonce method.
- ATCA_STATUS atPause (ATCACommand ca_cmd, ATCAPacket *packet)

  ATCACommand Pause method.
- ATCA_STATUS atPrivWrite (ATCACommand ca_cmd, ATCAPacket *packet)

  ATCACommand PrivWrite method.
- ATCA_STATUS atRandom (ATCACommand ca_cmd, ATCAPacket *packet)

  ATCACommand Random method.
- ATCA_STATUS atRead (ATCACommand ca_cmd, ATCAPacket *packet)

  ATCACommand Read method.
- ATCA_STATUS atSecureBoot (ATCACommand ca_cmd, ATCAPacket *packet)

  ATCACommand SecureBoot method.
- ATCA_STATUS atSHA (ATCACommand ca_cmd, ATCAPacket *packet, uint16_t write_context_size)

  ATCACommand SHA method.
- ATCA_STATUS atSign (ATCACommand ca_cmd, ATCAPacket *packet)

  ATCACommand Sign method.
- ATCA_STATUS atUpdateExtra (ATCACommand ca_cmd, ATCAPacket *packet)

  ATCACommand UpdateExtra method.
- ATCA_STATUS atVerify (ATCACommand ca_cmd, ATCAPacket *packet)

  ATCACommand ECDSA Verify method.
- ATCA_STATUS atWrite (ATCACommand ca_cmd, ATCAPacket *packet, bool has_mac)

  ATCACommand Write method.
- ATCA_STATUS atAES (ATCACommand ca_cmd, ATCAPacket *packet)

  ATCACommand AES method.
- ATCA_STATUS atSelfTest (ATCACommand ca_cmd, ATCAPacket *packet)

  ATCACommand AES method.
- ATCA_STATUS atKDF (ATCACommand ca_cmd, ATCAPacket *packet)

  ATCACommand KDF method.
- bool atIsSHAFamily (ATCADeviceType device_type)

  determines if a given device type is a SHA device or a superset of a SHA device
- bool atIsECCFamily (ATCADeviceType device_type)

  determines if a given device type is an ECC device or a superset of a ECC device
- ATCA_STATUS isATCAError (uint8_t *data)

  checks for basic error frame in data
- void atCRC (size_t length, const uint8_t *data, uint8_t *crc_le)

  Calculates CRC over the given raw data and returns the CRC in little-endian byte order.
- void atCalcCrc (ATCAPacket *pkt)

  This function calculates CRC and adds it to the correct offset in the packet data.
- ATCA_STATUS atCheckCrc (const uint8_t *response)

  This function checks the consistency of a response.

Opcodes for Crypto Authentication device commands

- #define ATCA_CHECKMAC ((uint8_t)0x28)

  CheckMac command op-code.
- #define ATCA_DERIVE_KEY ((uint8_t)0x1C)

  DeriveKey command op-code.
- #define ATCA_INFO ((uint8_t)0x30)

  Info command op-code.
- #define ATCA_GENDIG ((uint8_t)0x15)

  GenDig command op-code.
- #define ATCA_GENKEY ((uint8_t)0x40)

  GenKey command op-code.
- #define ATCA_HMAC ((uint8_t)0x11)

  HMAC command op-code.
- #define ATCA_LOCK ((uint8_t)0x17)

  Lock command op-code.
- #define ATCA_MAC ((uint8_t)0x08)

  MAC command op-code.
- #define ATCA_NONCE ((uint8_t)0x16)

  Nonce command op-code.
- #define ATCA_PAUSE ((uint8_t)0x01)

  Pause command op-code.
- #define ATCA_PRIVWRITE ((uint8_t)0x46)

  PrivWrite command op-code.
- #define ATCA_RANDOM ((uint8_t)0x1B)

  Random command op-code.
- #define ATCA_READ ((uint8_t)0x02)

  Read command op-code.
- #define ATCA_SIGN ((uint8_t)0x41)

  Sign command op-code.
- #define ATCA_UPDATE_EXTRA ((uint8_t)0x20)

  UpdateExtra command op-code.
- #define ATCA_VERIFY ((uint8_t)0x45)

  GenKey command op-code.
- #define ATCA_WRITE ((uint8_t)0x12)

  Write command op-code.
- #define ATCA_ECDH ((uint8_t)0x43)

  ECDH command op-code.
- #define ATCA_COUNTER ((uint8_t)0x24)

  Counter command op-code.
- #define ATCA_SHA ((uint8_t)0x47)

  SHA command op-code.
- #define ATCA_AES ((uint8_t)0x51)

  AES command op-code.
- #define ATCA_KDF ((uint8_t)0x56)

  KDF command op-code.
- #define ATCA_SECUREBOOT ((uint8_t)0x80)

  Secure Boot command op-code.
- #define ATCA_SELFTEST ((uint8_t)0x77)

  Self test command op-code.

Definitions of Data and Packet Sizes

- #define ATCA_BLOCK_SIZE (32)

  size of a block
- #define ATCA_WORD_SIZE (4)

  size of a word
- #define ATCA_PUB_KEY_PAD (4)

  size of the public key pad
- #define ATCA_SERIAL_NUM_SIZE (9)

  number of bytes in the device serial number
- #define ATCA_RSP_SIZE_VAL ((uint8_t)7)

  size of response packet containing four bytes of data
- #define ATCA_KEY_COUNT (16)

  number of keys
- #define ATCA_ECC_CONFIG_SIZE (128)

  size of configuration zone
- #define ATCA_SHA_CONFIG_SIZE (88)

  size of configuration zone
- #define ATCA_OTP_SIZE (64)

  size of OTP zone
- #define ATCA_DATA_SIZE (ATCA_KEY_COUNT ∗ ATCA_KEY_SIZE)

  size of data zone
- #define ATCA_AES_GFM_SIZE ATCA_BLOCK_SIZE

  size of GFM data
- #define ATCA_CHIPMODE_OFFSET (19)

  ChipMode byte offset within the configuration zone.
- #define ATCA_CHIPMODE_I2C_ADDRESS_FLAG ((uint8_t)0x01)

  ChipMode I2C Address in UserExtraAdd flag.
- #define ATCA_CHIPMODE_TTL_ENABLE_FLAG ((uint8_t)0x02)

  ChipMode TTLenable flag.
- #define ATCA_CHIPMODE_WATCHDOG_MASK ((uint8_t)0x04)

  ChipMode watchdog duration mask.
- #define ATCA_CHIPMODE_WATCHDOG_SHORT ((uint8_t)0x00)

  ChipMode short watchdog (∼1.3s)
- #define ATCA_CHIPMODE_WATCHDOG_LONG ((uint8_t)0x04)

  ChipMode long watchdog (∼13s)
- #define ATCA_CHIPMODE_CLOCK_DIV_MASK ((uint8_t)0xF8)

  ChipMode clock divider mask.
- #define ATCA_CHIPMODE_CLOCK_DIV_M0 ((uint8_t)0x00)

  ChipMode clock divider M0.
- #define ATCA_CHIPMODE_CLOCK_DIV_M1 ((uint8_t)0x28)

  ChipMode clock divider M1.
- #define ATCA_CHIPMODE_CLOCK_DIV_M2 ((uint8_t)0x68)

  ChipMode clock divider M2.
- #define ATCA_COUNT_SIZE ((uint8_t)1)

  Number of bytes in the command packet Count.
- #define ATCA_CRC_SIZE ((uint8_t)2)

  Number of bytes in the command packet CRC.
- #define ATCA_PACKET_OVERHEAD (ATCA_COUNT_SIZE + ATCA_CRC_SIZE)

  Number of bytes in the command packet.
- #define ATCA_PUB_KEY_SIZE (64)

size of a p256 public key
- #define ATCA_PRIV_KEY_SIZE (32)

    size of a p256 private key
- #define ATCA_SIG_SIZE (64)

    size of a p256 signature
- #define ATCA_KEY_SIZE (32)

    size of a symmetric SHA key
- #define RSA2048_KEY_SIZE (256)

    size of a RSA private key
- #define ATCA_RSP_SIZE_MIN ((uint8_t)4)

    minimum number of bytes in response
- #define ATCA_RSP_SIZE_4 ((uint8_t)7)

    size of response packet containing 4 bytes data
- #define ATCA_RSP_SIZE_72 ((uint8_t)75)

    size of response packet containing 64 bytes data
- #define ATCA_RSP_SIZE_64 ((uint8_t)67)

    size of response packet containing 64 bytes data
- #define ATCA_RSP_SIZE_32 ((uint8_t)35)

    size of response packet containing 32 bytes data
- #define ATCA_RSP_SIZE_16 ((uint8_t)19)

    size of response packet containing 16 bytes data
- #define ATCA_RSP_SIZE_MAX ((uint8_t)75)

    maximum size of response packet (GenKey and Verify command)
- #define OUTNONCE_SIZE (32)

    Size of the OutNonce response expected from several commands.

Definitions for Command Parameter Ranges

- #define ATCA_KEY_ID_MAX ((uint8_t)15)

    maximum value for key id
- #define ATCA_OTP_BLOCK_MAX ((uint8_t)1)

    maximum value for OTP block

Definitions for Indexes Common to All Commands

- #define ATCA_COUNT_IDX (0)

    command packet index for count
- #define ATCA_OPCODE_IDX (1)

    command packet index for op-code
- #define ATCA_PARAM1_IDX (2)

    command packet index for first parameter
- #define ATCA_PARAM2_IDX (3)

    command packet index for second parameter
- #define ATCA_DATA_IDX (5)

    command packet index for data load
- #define ATCA_RSP_DATA_IDX (1)

    buffer index of data in response

Definitions for Zone and Address Parameters

- #define ATCA_ZONE_CONFIG ((uint8_t)0x00)
  Configuration zone.
- #define ATCA_ZONE_OTP ((uint8_t)0x01)
  OTP (One Time Programming) zone.
- #define ATCA_ZONE_DATA ((uint8_t)0x02)
  Data zone.
- #define ATCA_ZONE_MASK ((uint8_t)0x03)
  Zone mask.
- #define ATCA_ZONE_ENCRYPTED ((uint8_t)0x40)
  Zone bit 6 set: Write is encrypted with an unlocked data zone.
- #define ATCA_ZONE_READWRITE_32 ((uint8_t)0x80)
  Zone bit 7 set: Access 32 bytes, otherwise 4 bytes.
- #define ATCA_ADDRESS_MASK_CONFIG (0x001F)
  Address bits 5 to 7 are 0 for Configuration zone.
- #define ATCA_ADDRESS_MASK_OTP (0x000F)
  Address bits 4 to 7 are 0 for OTP zone.
- #define ATCA_ADDRESS_MASK (0x007F)
  Address bit 7 to 15 are always 0.
- #define ATCA_TEMPKEY_KEYID (0xFFFF)
  KeyID when referencing TempKey.

Definitions for Key types

- #define ATCA_B283_KEY_TYPE 0
  B283 NIST ECC key.
- #define ATCA_K283_KEY_TYPE 1
  K283 NIST ECC key.
- #define ATCA_P256_KEY_TYPE 4
  P256 NIST ECC key.
- #define ATCA_AES_KEY_TYPE 6
  AES-128 Key.
- #define ATCA_SHA_KEY_TYPE 7
  SHA key or other data.

Definitions for the AES Command

- #define AES_MODE_IDX ATCA_PARAM1_IDX
  AES command index for mode.
- #define AES_KEYID_IDX ATCA_PARAM2_IDX
  AES command index for key id.
- #define AES_INPUT_IDX ATCA_DATA_IDX
  AES command index for input data.
- #define AES_COUNT (23)
  AES command packet size.
- #define AES_MODE_MASK ((uint8_t)0xC7)
  AES mode bits 3 to 5 are 0.
- #define AES_MODE_KEY_BLOCK_MASK ((uint8_t)0xC0)

AES mode mask for key block field.
- #define AES_MODE_OP_MASK ((uint8_t)0x07)

    AES mode operation mask.
- #define AES_MODE_ENCRYPT ((uint8_t)0x00)

    AES mode: Encrypt.
- #define AES_MODE_DECRYPT ((uint8_t)0x01)

    AES mode: Decrypt.
- #define AES_MODE_GFM ((uint8_t)0x03)

    AES mode: GFM calculation.
- #define AES_MODE_KEY_BLOCK_POS (6)

    Bit shift for key block in mode.
- #define AES_DATA_SIZE (16)

    size of AES encrypt/decrypt data
- #define AES_RSP_SIZE ATCA_RSP_SIZE_16

    AES command response packet size.

Definitions for the CheckMac Command

- #define CHECKMAC_MODE_IDX ATCA_PARAM1_IDX

    CheckMAC command index for mode.
- #define CHECKMAC_KEYID_IDX ATCA_PARAM2_IDX

    CheckMAC command index for key identifier.
- #define CHECKMAC_CLIENT_CHALLENGE_IDX ATCA_DATA_IDX

    CheckMAC command index for client challenge.
- #define CHECKMAC_CLIENT_RESPONSE_IDX (37)

    CheckMAC command index for client response.
- #define CHECKMAC_DATA_IDX (69)

    CheckMAC command index for other data.
- #define CHECKMAC_COUNT (84)

    CheckMAC command packet size.
- #define CHECKMAC_MODE_CHALLENGE ((uint8_t)0x00)

    CheckMAC mode 0: first SHA block from key id.
- #define CHECKMAC_MODE_BLOCK2_TEMPKEY ((uint8_t)0x01)

    CheckMAC mode bit 0: second SHA block from TempKey.
- #define CHECKMAC_MODE_BLOCK1_TEMPKEY ((uint8_t)0x02)

    CheckMAC mode bit 1: first SHA block from TempKey.
- #define CHECKMAC_MODE_SOURCE_FLAG_MATCH ((uint8_t)0x04)

    CheckMAC mode bit 2: match TempKey.SourceFlag.
- #define CHECKMAC_MODE_INCLUDE_OTP_64 ((uint8_t)0x20)

    CheckMAC mode bit 5: include first 64 OTP bits.
- #define CHECKMAC_MODE_MASK ((uint8_t)0x27)

    CheckMAC mode bits 3, 4, 6, and 7 are 0.
- #define CHECKMAC_CLIENT_CHALLENGE_SIZE (32)

    CheckMAC size of client challenge.
- #define CHECKMAC_CLIENT_RESPONSE_SIZE (32)

    CheckMAC size of client response.
- #define CHECKMAC_OTHER_DATA_SIZE (13)

    CheckMAC size of "other data".
- #define CHECKMAC_CLIENT_COMMAND_SIZE (4)

    CheckMAC size of client command header size inside "other data".

- #define CHECKMAC_CMD_MATCH (0)

  CheckMAC return value when there is a match.
- #define CHECKMAC_CMD_MISMATCH (1)

  CheckMAC return value when there is a mismatch.
- #define CHECKMAC_RSP_SIZE ATCA_RSP_SIZE_MIN

  CheckMAC response packet size.

Definitions for the Counter command

- #define COUNTER_COUNT ATCA_CMD_SIZE_MIN
- #define COUNTER_MODE_IDX ATCA_PARAM1_IDX

  Counter command index for mode.
- #define COUNTER_KEYID_IDX ATCA_PARAM2_IDX

  Counter command index for key id.
- #define COUNTER_MODE_MASK ((uint8_t)0x01)

  Counter mode bits 1 to 7 are 0.
- #define COUNTER_MAX_VALUE ((uint32_t)2097151)

  Counter maximum value of the counter.
- #define COUNTER_MODE_READ ((uint8_t)0x00)

  Counter command mode for reading.
- #define COUNTER_MODE_INCREMENT ((uint8_t)0x01)

  Counter command mode for incrementing.
- #define COUNTER_RSP_SIZE ATCA_RSP_SIZE_4

  Counter command response packet size.
- #define COUNTER_SIZE ATCA_RSP_SIZE_MIN

  Counter size in binary.

Definitions for the DeriveKey Command

- #define DERIVE_KEY_RANDOM_IDX ATCA_PARAM1_IDX

  DeriveKey command index for random bit.
- #define DERIVE_KEY_TARGETKEY_IDX ATCA_PARAM2_IDX

  DeriveKey command index for target slot.
- #define DERIVE_KEY_MAC_IDX ATCA_DATA_IDX

  DeriveKey command index for optional MAC.
- #define DERIVE_KEY_COUNT_SMALL ATCA_CMD_SIZE_MIN

  DeriveKey command packet size without MAC.
- #define DERIVE_KEY_MODE ((uint8_t)0x04)

  DeriveKey command mode set to 4 as in datasheet.
- #define DERIVE_KEY_COUNT_LARGE (39)

  DeriveKey command packet size with MAC.
- #define DERIVE_KEY_RANDOM_FLAG ((uint8_t)4)

  DeriveKey 1. parameter; has to match TempKey.SourceFlag.
- #define DERIVE_KEY_MAC_SIZE (32)

  DeriveKey MAC size.
- #define DERIVE_KEY_RSP_SIZE ATCA_RSP_SIZE_MIN

  DeriveKey response packet size.

Definitions for the ECDH Command

- #define ECDH_PREFIX_MODE ((uint8_t)0x00)
- #define ECDH_COUNT (ATCA_CMD_SIZE_MIN + ATCA_PUB_KEY_SIZE)
- #define ECDH_MODE_SOURCE_MASK ((uint8_t)0x01)
- #define ECDH_MODE_SOURCE_EEPROM_SLOT ((uint8_t)0x00)
- #define ECDH_MODE_SOURCE_TEMPKEY ((uint8_t)0x01)
- #define ECDH_MODE_OUTPUT_MASK ((uint8_t)0x02)
- #define ECDH_MODE_OUTPUT_CLEAR ((uint8_t)0x00)
- #define ECDH_MODE_OUTPUT_ENC ((uint8_t)0x02)
- #define ECDH_MODE_COPY_MASK ((uint8_t)0x0C)
- #define ECDH_MODE_COPY_COMPATIBLE ((uint8_t)0x00)
- #define ECDH_MODE_COPY_EEPROM_SLOT ((uint8_t)0x04)
- #define ECDH_MODE_COPY_TEMP_KEY ((uint8_t)0x08)
- #define ECDH_MODE_COPY_OUTPUT_BUFFER ((uint8_t)0x0C)
- #define ECDH_KEY_SIZE ATCA_BLOCK_SIZE

  ECDH output data size.
- #define ECDH_RSP_SIZE ATCA_RSP_SIZE_64

  ECDH command packet size.

Definitions for the GenDig Command

- #define GENDIG_ZONE_IDX ATCA_PARAM1_IDX

  GenDig command index for zone.
- #define GENDIG_KEYID_IDX ATCA_PARAM2_IDX

  GenDig command index for key id.
- #define GENDIG_DATA_IDX ATCA_DATA_IDX

  GenDig command index for optional data.
- #define GENDIG_COUNT ATCA_CMD_SIZE_MIN

  GenDig command packet size without "other data".
- #define GENDIG_ZONE_CONFIG ((uint8_t)0)

  GenDig zone id config. Use KeyID to specify any of the four 256-bit blocks of the Configuration zone.
- #define GENDIG_ZONE_OTP ((uint8_t)1)

  GenDig zone id OTP. Use KeyID to specify either the first or second 256-bit block of the OTP zone.
- #define GENDIG_ZONE_DATA ((uint8_t)2)

  GenDig zone id data. Use KeyID to specify a slot in the Data zone or a transport key in the hardware array.
- #define GENDIG_ZONE_SHARED_NONCE ((uint8_t)3)

  GenDig zone id shared nonce. KeyID specifies the location of the input value in the message generation.
- #define GENDIG_ZONE_COUNTER ((uint8_t)4)

  GenDig zone id counter. KeyID specifies the monotonic counter ID to be included in the message generation.
- #define GENDIG_ZONE_KEY_CONFIG ((uint8_t)5)

  GenDig zone id key config. KeyID specifies the slot for which the configuration information is to be included in the message generation.
- #define GENDIG_RSP_SIZE ATCA_RSP_SIZE_MIN

  GenDig command response packet size.

Definitions for the GenKey Command

- #define GENKEY_MODE_IDX ATCA_PARAM1_IDX

  GenKey command index for mode.
- #define GENKEY_KEYID_IDX ATCA_PARAM2_IDX

  GenKey command index for key id.
- #define GENKEY_DATA_IDX (5)

  GenKey command index for other data.
- #define GENKEY_COUNT ATCA_CMD_SIZE_MIN

  GenKey command packet size without "other data".
- #define GENKEY_COUNT_DATA (10)

  GenKey command packet size with "other data".
- #define GENKEY_OTHER_DATA_SIZE (3)

  GenKey size of "other data".
- #define GENKEY_MODE_MASK ((uint8_t)0x1C)

  GenKey mode bits 0 to 1 and 5 to 7 are 0.
- #define GENKEY_MODE_PRIVATE ((uint8_t)0x04)

  GenKey mode: private key generation.
- #define GENKEY_MODE_PUBLIC ((uint8_t)0x00)

  GenKey mode: public key calculation.
- #define GENKEY_MODE_DIGEST ((uint8_t)0x08)

  GenKey mode: PubKey digest will be created after the public key is calculated.
- #define GENKEY_MODE_PUBKEY_DIGEST ((uint8_t)0x10)

  GenKey mode: Calculate PubKey digest on the public key in KeyId.
- #define GENKEY_PRIVATE_TO_TEMPKEY ((uint16_t)0xFFFF)

  GenKey Create private key and store to tempkey (608 only)
- #define GENKEY_RSP_SIZE_SHORT ATCA_RSP_SIZE_MIN

  GenKey response packet size in Digest mode.
- #define GENKEY_RSP_SIZE_LONG ATCA_RSP_SIZE_64

  GenKey response packet size when returning a public key.

Definitions for the HMAC Command

- #define HMAC_MODE_IDX ATCA_PARAM1_IDX

  HMAC command index for mode.
- #define HMAC_KEYID_IDX ATCA_PARAM2_IDX

  HMAC command index for key id.
- #define HMAC_COUNT ATCA_CMD_SIZE_MIN

  HMAC command packet size.
- #define HMAC_MODE_FLAG_TK_RAND ((uint8_t)0x00)

  HMAC mode bit 2: The value of this bit must match the value in TempKey.SourceFlag or the command will return an error.
- #define HMAC_MODE_FLAG_TK_NORAND ((uint8_t)0x04)

  HMAC mode bit 2: The value of this bit must match the value in TempKey.SourceFlag or the command will return an error.
- #define HMAC_MODE_FLAG_OTP88 ((uint8_t)0x10)

  HMAC mode bit 4: Include the first 88 OTP bits (OTP[0] through OTP[10]) in the message.; otherwise, the corresponding message bits are set to zero. Not applicable for ATECC508A.
- #define HMAC_MODE_FLAG_OTP64 ((uint8_t)0x20)

  HMAC mode bit 5: Include the first 64 OTP bits (OTP[0] through OTP[7]) in the message.; otherwise, the corresponding message bits are set to zero. If Mode[4] is set, the value of this mode bit is ignored. Not applicable for ATECC508A.

- #define HMAC_MODE_FLAG_FULLSN ((uint8_t)0x40)

  HMAC mode bit 6: If set, include the 48 bits SN[2:3] and SN[4:7] in the message.; otherwise, the corresponding message bits are set to zero.
- #define HMAC_MODE_MASK ((uint8_t)0x74)

  HMAC mode bits 0, 1, 3, and 7 are 0.
- #define HMAC_DIGEST_SIZE (32)

  HMAC size of digest response.
- #define HMAC_RSP_SIZE ATCA_RSP_SIZE_32

  HMAC command response packet size.

Definitions for the Info Command

- #define INFO_PARAM1_IDX ATCA_PARAM1_IDX

  Info command index for 1. parameter.
- #define INFO_PARAM2_IDX ATCA_PARAM2_IDX

  Info command index for 2. parameter.
- #define INFO_COUNT ATCA_CMD_SIZE_MIN

  Info command packet size.
- #define INFO_MODE_REVISION ((uint8_t)0x00)

  Info mode Revision.
- #define INFO_MODE_KEY_VALID ((uint8_t)0x01)

  Info mode KeyValid.
- #define INFO_MODE_STATE ((uint8_t)0x02)

  Info mode State.
- #define INFO_MODE_GPIO ((uint8_t)0x03)

  Info mode GPIO.
- #define INFO_MODE_VOL_KEY_PERMIT ((uint8_t)0x04)

  Info mode GPIO.
- #define INFO_MODE_MAX ((uint8_t)0x03)

  Info mode maximum value.
- #define INFO_NO_STATE ((uint8_t)0x00)

  Info mode is not the state mode.
- #define INFO_OUTPUT_STATE_MASK ((uint8_t)0x01)

  Info output state mask.
- #define INFO_DRIVER_STATE_MASK ((uint8_t)0x02)

  Info driver state mask.
- #define INFO_PARAM2_SET_LATCH_STATE ((uint16_t)0x0002)

  Info param2 to set the persistent latch state.
- #define INFO_PARAM2_LATCH_SET ((uint16_t)0x0001)

  Info param2 to set the persistent latch.
- #define INFO_PARAM2_LATCH_CLEAR ((uint16_t)0x0000)

  Info param2 to clear the persistent latch.
- #define INFO_SIZE ((uint8_t)0x04)

  Info return size.
- #define INFO_RSP_SIZE ATCA_RSP_SIZE_VAL

  Info command response packet size.

Definitions for the KDF Command

- #define KDF_MODE_IDX ATCA_PARAM1_IDX

  KDF command index for mode.
- #define KDF_KEYID_IDX ATCA_PARAM2_IDX

  KDF command index for key id.
- #define KDF_DETAILS_IDX ATCA_DATA_IDX

  KDF command index for details.
- #define KDF_DETAILS_SIZE 4

  KDF details (param3) size.
- #define KDF_MESSAGE_IDX (ATCA_DATA_IDX + KDF_DETAILS_SIZE)
- #define KDF_MODE_SOURCE_MASK ((uint8_t)0x03)

  KDF mode source key mask.
- #define KDF_MODE_SOURCE_TEMPKEY ((uint8_t)0x00)

  KDF mode source key in TempKey.
- #define KDF_MODE_SOURCE_TEMPKEY_UP ((uint8_t)0x01)

  KDF mode source key in upper TempKey.
- #define KDF_MODE_SOURCE_SLOT ((uint8_t)0x02)

  KDF mode source key in a slot.
- #define KDF_MODE_SOURCE_ALTKEYBUF ((uint8_t)0x03)

  KDF mode source key in alternate key buffer.
- #define KDF_MODE_TARGET_MASK ((uint8_t)0x1C)

  KDF mode target key mask.
- #define KDF_MODE_TARGET_TEMPKEY ((uint8_t)0x00)

  KDF mode target key in TempKey.
- #define KDF_MODE_TARGET_TEMPKEY_UP ((uint8_t)0x04)

  KDF mode target key in upper TempKey.
- #define KDF_MODE_TARGET_SLOT ((uint8_t)0x08)

  KDF mode target key in slot.
- #define KDF_MODE_TARGET_ALTKEYBUF ((uint8_t)0x0C)

  KDF mode target key in alternate key buffer.
- #define KDF_MODE_TARGET_OUTPUT ((uint8_t)0x10)

  KDF mode target key in output buffer.
- #define KDF_MODE_TARGET_OUTPUT_ENC ((uint8_t)0x14)

  KDF mode target key encrypted in output buffer.
- #define KDF_MODE_ALG_MASK ((uint8_t)0x60)

  KDF mode algorithm mask.
- #define KDF_MODE_ALG_PRF ((uint8_t)0x00)

  KDF mode PRF algorithm.
- #define KDF_MODE_ALG_AES ((uint8_t)0x20)

  KDF mode AES algorithm.
- #define KDF_MODE_ALG_HKDF ((uint8_t)0x40)

  KDF mode HKDF algorithm.
- #define KDF_DETAILS_PRF_KEY_LEN_MASK ((uint32_t)0x00000003)

  KDF details for PRF, source key length mask.
- #define KDF_DETAILS_PRF_KEY_LEN_16 ((uint32_t)0x00000000)

  KDF details for PRF, source key length is 16 bytes.
- #define KDF_DETAILS_PRF_KEY_LEN_32 ((uint32_t)0x00000001)

  KDF details for PRF, source key length is 32 bytes.
- #define KDF_DETAILS_PRF_KEY_LEN_48 ((uint32_t)0x00000002)

  KDF details for PRF, source key length is 48 bytes.

- #define KDF_DETAILS_PRF_KEY_LEN_64 ((uint32_t)0x00000003)

  KDF details for PRF, source key length is 64 bytes.
- #define KDF_DETAILS_PRF_TARGET_LEN_MASK ((uint32_t)0x00000100)

  KDF details for PRF, target length mask.
- #define KDF_DETAILS_PRF_TARGET_LEN_32 ((uint32_t)0x00000000)

  KDF details for PRF, target length is 32 bytes.
- #define KDF_DETAILS_PRF_TARGET_LEN_64 ((uint32_t)0x00000100)

  KDF details for PRF, target length is 64 bytes.
- #define KDF_DETAILS_PRF_AEAD_MASK ((uint32_t)0x00000600)

  KDF details for PRF, AEAD processing mask.
- #define KDF_DETAILS_PRF_AEAD_MODE0 ((uint32_t)0x00000000)

  KDF details for PRF, AEAD no processing.
- #define KDF_DETAILS_PRF_AEAD_MODE1 ((uint32_t)0x00000200)

  KDF details for PRF, AEAD First 32 go to target, second 32 go to output buffer.
- #define KDF_DETAILS_AES_KEY_LOC_MASK ((uint32_t)0x00000003)

  KDF details for AES, key location mask.
- #define KDF_DETAILS_HKDF_MSG_LOC_MASK ((uint32_t)0x00000003)

  KDF details for HKDF, message location mask.
- #define KDF_DETAILS_HKDF_MSG_LOC_SLOT ((uint32_t)0x00000000)

  KDF details for HKDF, message location in slot.
- #define KDF_DETAILS_HKDF_MSG_LOC_TEMPKEY ((uint32_t)0x00000001)

  KDF details for HKDF, message location in TempKey.
- #define KDF_DETAILS_HKDF_MSG_LOC_INPUT ((uint32_t)0x00000002)

  KDF details for HKDF, message location in input parameter.
- #define KDF_DETAILS_HKDF_MSG_LOC_IV ((uint32_t)0x00000003)

  KDF details for HKDF, message location is a special IV function.
- #define KDF_DETAILS_HKDF_ZERO_KEY ((uint32_t)0x00000004)

  KDF details for HKDF, key is 32 bytes of zero.

Definitions for the Lock Command

- #define LOCK_ZONE_IDX ATCA_PARAM1_IDX

  Lock command index for zone.
- #define LOCK_SUMMARY_IDX ATCA_PARAM2_IDX

  Lock command index for summary.
- #define LOCK_COUNT ATCA_CMD_SIZE_MIN

  Lock command packet size.
- #define LOCK_ZONE_CONFIG ((uint8_t)0x00)

  Lock zone is Config.
- #define LOCK_ZONE_DATA ((uint8_t)0x01)

  Lock zone is OTP or Data.
- #define LOCK_ZONE_DATA_SLOT ((uint8_t)0x02)

  Lock slot of Data.
- #define LOCK_ZONE_NO_CRC ((uint8_t)0x80)

  Lock command: Ignore summary.
- #define LOCK_ZONE_MASK (0xBF)

  Lock parameter 1 bits 6 are 0.
- #define ATCA_UNLOCKED (0x55)

  Value indicating an unlocked zone.
- #define ATCA_LOCKED (0x00)

  Value indicating a locked zone.
- #define LOCK_RSP_SIZE ATCA_RSP_SIZE_MIN

  Lock command response packet size.

Definitions for the MAC Command

- #define MAC_MODE_IDX ATCA_PARAM1_IDX

  MAC command index for mode.
- #define MAC_KEYID_IDX ATCA_PARAM2_IDX

  MAC command index for key id.
- #define MAC_CHALLENGE_IDX ATCA_DATA_IDX

  MAC command index for optional challenge.
- #define MAC_COUNT_SHORT ATCA_CMD_SIZE_MIN

  MAC command packet size without challenge.
- #define MAC_COUNT_LONG (39)

  MAC command packet size with challenge.
- #define MAC_MODE_CHALLENGE ((uint8_t)0x00)

  MAC mode 0: first SHA block from data slot.
- #define MAC_MODE_BLOCK2_TEMPKEY ((uint8_t)0x01)

  MAC mode bit 0: second SHA block from TempKey.
- #define MAC_MODE_BLOCK1_TEMPKEY ((uint8_t)0x02)

  MAC mode bit 1: first SHA block from TempKey.
- #define MAC_MODE_SOURCE_FLAG_MATCH ((uint8_t)0x04)

  MAC mode bit 2: match TempKey.SourceFlag.
- #define MAC_MODE_PTNONCE_TEMPKEY ((uint8_t)0x06)

  MAC mode bit 0: second SHA block from TempKey.
- #define MAC_MODE_PASSTHROUGH ((uint8_t)0x07)

  MAC mode bit 0-2: pass-through mode.
- #define MAC_MODE_INCLUDE_OTP_88 ((uint8_t)0x10)

  MAC mode bit 4: include first 88 OTP bits.
- #define MAC_MODE_INCLUDE_OTP_64 ((uint8_t)0x20)

  MAC mode bit 5: include first 64 OTP bits.
- #define MAC_MODE_INCLUDE_SN ((uint8_t)0x40)

  MAC mode bit 6: include serial number.
- #define MAC_CHALLENGE_SIZE (32)

  MAC size of challenge.
- #define MAC_SIZE (32)

  MAC size of response.
- #define MAC_MODE_MASK ((uint8_t)0x77)

  MAC mode bits 3 and 7 are 0.
- #define MAC_RSP_SIZE ATCA_RSP_SIZE_32

  MAC command response packet size.

Definitions for the Nonce Command

- #define NONCE_MODE_IDX ATCA_PARAM1_IDX

  Nonce command index for mode.
- #define NONCE_PARAM2_IDX ATCA_PARAM2_IDX

  Nonce command index for 2. parameter.
- #define NONCE_INPUT_IDX ATCA_DATA_IDX

  Nonce command index for input data.
- #define NONCE_COUNT_SHORT (ATCA_CMD_SIZE_MIN + 20)

  Nonce command packet size for 20 bytes of NumIn.
- #define NONCE_COUNT_LONG (ATCA_CMD_SIZE_MIN + 32)

Nonce command packet size for 32 bytes of NumIn.
- #define NONCE_COUNT_LONG_64 (ATCA_CMD_SIZE_MIN + 64)

  Nonce command packet size for 64 bytes of NumIn.
- #define NONCE_MODE_MASK ((uint8_t)0x03)

  Nonce mode bits 2 to 7 are 0.
- #define NONCE_MODE_SEED_UPDATE ((uint8_t)0x00)

  Nonce mode: update seed.
- #define NONCE_MODE_NO_SEED_UPDATE ((uint8_t)0x01)

  Nonce mode: do not update seed.
- #define NONCE_MODE_INVALID ((uint8_t)0x02)

  Nonce mode 2 is invalid.
- #define NONCE_MODE_PASSTHROUGH ((uint8_t)0x03)

  Nonce mode: pass-through.
- #define NONCE_MODE_INPUT_LEN_MASK ((uint8_t)0x20)

  Nonce mode: input size mask.
- #define NONCE_MODE_INPUT_LEN_32 ((uint8_t)0x00)

  Nonce mode: input size is 32 bytes.
- #define NONCE_MODE_INPUT_LEN_64 ((uint8_t)0x20)

  Nonce mode: input size is 64 bytes.
- #define NONCE_MODE_TARGET_MASK ((uint8_t)0xC0)

  Nonce mode: target mask.
- #define NONCE_MODE_TARGET_TEMPKEY ((uint8_t)0x00)

  Nonce mode: target is TempKey.
- #define NONCE_MODE_TARGET_MSGDIGBUF ((uint8_t)0x40)

  Nonce mode: target is Message Digest Buffer.
- #define NONCE_MODE_TARGET_ALTKEYBUF ((uint8_t)0x80)

  Nonce mode: target is Alternate Key Buffer.
- #define NONCE_ZERO_CALC_MASK ((uint16_t)0x8000)

  Nonce zero (param2): calculation mode mask.
- #define NONCE_ZERO_CALC_RANDOM ((uint16_t)0x0000)

  Nonce zero (param2): calculation mode random, use RNG in calculation and return RNG output.
- #define NONCE_ZERO_CALC_TEMPKEY ((uint16_t)0x8000)

  Nonce zero (param2): calculation mode TempKey, use TempKey in calculation and return new TempKey value.
- #define NONCE_NUMIN_SIZE (20)

  Nonce NumIn size for random modes.
- #define NONCE_NUMIN_SIZE_PASSTHROUGH (32)

  Nonce NumIn size for 32-byte pass-through mode.
- #define NONCE_RSP_SIZE_SHORT ATCA_RSP_SIZE_MIN

  Nonce command response packet size with no output.
- #define NONCE_RSP_SIZE_LONG ATCA_RSP_SIZE_32

  Nonce command response packet size with output.

Definitions for the Pause Command

- #define PAUSE_SELECT_IDX ATCA_PARAM1_IDX

  Pause command index for Selector.
- #define PAUSE_PARAM2_IDX ATCA_PARAM2_IDX

  Pause command index for 2. parameter.
- #define PAUSE_COUNT ATCA_CMD_SIZE_MIN

  Pause command packet size.
- #define PAUSE_RSP_SIZE ATCA_RSP_SIZE_MIN

  Pause command response packet size.

Definitions for the PrivWrite Command

- #define PRIVWRITE_ZONE_IDX ATCA_PARAM1_IDX

  PrivWrite command index for zone.
- #define PRIVWRITE_KEYID_IDX ATCA_PARAM2_IDX

  PrivWrite command index for KeyID.
- #define PRIVWRITE_VALUE_IDX (5)

  PrivWrite command index for value.
- #define PRIVWRITE_MAC_IDX (41)

  PrivWrite command index for MAC.
- #define PRIVWRITE_COUNT (75)

  PrivWrite command packet size.
- #define PRIVWRITE_ZONE_MASK ((uint8_t)0x40)

  PrivWrite zone bits 0 to 5 and 7 are 0.
- #define PRIVWRITE_MODE_ENCRYPT ((uint8_t)0x40)

  PrivWrite mode: encrypted.
- #define PRIVWRITE_RSP_SIZE ATCA_RSP_SIZE_MIN

  PrivWrite command response packet size.

Definitions for the Random Command

- #define RANDOM_MODE_IDX ATCA_PARAM1_IDX

  Random command index for mode.
- #define RANDOM_PARAM2_IDX ATCA_PARAM2_IDX

  Random command index for 2. parameter.
- #define RANDOM_COUNT ATCA_CMD_SIZE_MIN

  Random command packet size.
- #define RANDOM_SEED_UPDATE ((uint8_t)0x00)

  Random mode for automatic seed update.
- #define RANDOM_NO_SEED_UPDATE ((uint8_t)0x01)

  Random mode for no seed update.
- #define RANDOM_NUM_SIZE ((uint8_t)32)

  Number of bytes in the data packet of a random command.
- #define RANDOM_RSP_SIZE ATCA_RSP_SIZE_32

  Random command response packet size.

Definitions for the Read Command

- #define READ_ZONE_IDX ATCA_PARAM1_IDX

  Read command index for zone.
- #define READ_ADDR_IDX ATCA_PARAM2_IDX

  Read command index for address.
- #define READ_COUNT ATCA_CMD_SIZE_MIN

  Read command packet size.
- #define READ_ZONE_MASK ((uint8_t)0x83)

  Read zone bits 2 to 6 are 0.
- #define READ_4_RSP_SIZE ATCA_RSP_SIZE_VAL

  Read command response packet size when reading 4 bytes.
- #define READ_32_RSP_SIZE ATCA_RSP_SIZE_32

  Read command response packet size when reading 32 bytes.

Definitions for the SecureBoot Command

- #define SECUREBOOT_MODE_IDX ATCA_PARAM1_IDX

  SecureBoot command index for mode.
- #define SECUREBOOT_DIGEST_SIZE (32)

  SecureBoot digest input size.
- #define SECUREBOOT_SIGNATURE_SIZE (64)

  SecureBoot signature input size.
- #define SECUREBOOT_COUNT_DIG (ATCA_CMD_SIZE_MIN + SECUREBOOT_DIGEST_SIZE)

  SecureBoot command packet size for just a digest.
- #define SECUREBOOT_COUNT_DIG_SIG (ATCA_CMD_SIZE_MIN + SECUREBOOT_DIGEST_SIZE + SECUREBOOT_SIGNATURE_SIZE)

  SecureBoot command packet size for a digest and signature.
- #define SECUREBOOT_MAC_SIZE (32)

  SecureBoot MAC output size.
- #define SECUREBOOT_RSP_SIZE_NO_MAC ATCA_RSP_SIZE_MIN

  SecureBoot response packet size for no MAC.
- #define SECUREBOOT_RSP_SIZE_MAC (ATCA_PACKET_OVERHEAD + SECUREBOOT_MAC_SIZE)

  SecureBoot response packet size with MAC.
- #define SECUREBOOT_MODE_MASK ((uint8_t)0x07)

  SecureBoot mode mask.
- #define SECUREBOOT_MODE_FULL ((uint8_t)0x05)

  SecureBoot mode Full.
- #define SECUREBOOT_MODE_FULL_STORE ((uint8_t)0x06)

  SecureBoot mode FullStore.
- #define SECUREBOOT_MODE_FULL_COPY ((uint8_t)0x07)

  SecureBoot mode FullCopy.
- #define SECUREBOOT_MODE_PROHIBIT_FLAG ((uint8_t)0x40)

  SecureBoot mode flag to prohibit SecureBoot until next power cycle.
- #define SECUREBOOT_MODE_ENC_MAC_FLAG ((uint8_t)0x80)

  SecureBoot mode flag for encrypted digest and returning validating MAC.
- #define SECUREBOOTCONFIG_OFFSET (70)

  SecureBootConfig byte offset into the configuration zone.
- #define SECUREBOOTCONFIG_MODE_MASK ((uint16_t)0x0003)

  Mask for SecureBootMode field in SecureBootConfig value.
- #define SECUREBOOTCONFIG_MODE_DISABLED ((uint16_t)0x0000)

  Disabled SecureBootMode in SecureBootConfig value.
- #define SECUREBOOTCONFIG_MODE_FULL_BOTH ((uint16_t)0x0001)

  Both digest and signature always required SecureBootMode in SecureBootConfig value.
- #define SECUREBOOTCONFIG_MODE_FULL_SIG ((uint16_t)0x0002)

  Signature stored SecureBootMode in SecureBootConfig value.
- #define SECUREBOOTCONFIG_MODE_FULL_DIG ((uint16_t)0x0003)

  Digest stored SecureBootMode in SecureBootConfig value.

Definitions for the SelfTest Command

- #define SELFTEST_MODE_IDX ATCA_PARAM1_IDX

  SelfTest command index for mode.
- #define SELFTEST_COUNT ATCA_CMD_SIZE_MIN

  SelfTest command packet size.
- #define SELFTEST_MODE_RNG ((uint8_t)0x01)

  SelfTest mode RNG DRBG function.
- #define SELFTEST_MODE_ECDSA_SIGN_VERIFY ((uint8_t)0x02)

  SelfTest mode ECDSA verify function.
- #define SELFTEST_MODE_ECDH ((uint8_t)0x08)

  SelfTest mode ECDH function.
- #define SELFTEST_MODE_AES ((uint8_t)0x10)

  SelfTest mode AES encrypt function.
- #define SELFTEST_MODE_SHA ((uint8_t)0x20)

  SelfTest mode SHA function.
- #define SELFTEST_MODE_ALL ((uint8_t)0x3B)

  SelfTest mode all algorithms.
- #define SELFTEST_RSP_SIZE ATCA_RSP_SIZE_MIN

  SelfTest command response packet size.

Definitions for the SHA Command

- #define SHA_COUNT_SHORT ATCA_CMD_SIZE_MIN
- #define SHA_COUNT_LONG ATCA_CMD_SIZE_MIN

  Just a starting size.
- #define ATCA_SHA_DIGEST_SIZE (32)
- #define SHA_DATA_MAX (64)
- #define ATCA_SHA256_BLOCK_SIZE (64)
- #define SHA_CONTEXT_MAX_SIZE (99)
- #define SHA_MODE_MASK ((uint8_t)0x07)

  Mask the bit 0-2.
- #define SHA_MODE_SHA256_START ((uint8_t)0x00)

  Initialization, does not accept a message.
- #define SHA_MODE_SHA256_UPDATE ((uint8_t)0x01)

  Add 64 bytes in the meesage to the SHA context.
- #define SHA_MODE_SHA256_END ((uint8_t)0x02)

  Complete the calculation and return the digest.
- #define SHA_MODE_SHA256_PUBLIC ((uint8_t)0x03)

  Add 64 byte ECC public key in the slot to the SHA context.
- #define SHA_MODE_HMAC_START ((uint8_t)0x04)

  Initialization, HMAC calculation.
- #define SHA_MODE_HMAC_UPDATE ((uint8_t)0x01)

  Add 64 bytes in the meesage to the SHA context.
- #define SHA_MODE_HMAC_END ((uint8_t)0x05)

  Complete the HMAC computation and return digest.
- #define SHA_MODE_608_HMAC_END ((uint8_t)0x02)

  Complete the HMAC computation and return digest... Different command on 608.
- #define SHA_MODE_READ_CONTEXT ((uint8_t)0x06)

  Read current SHA-256 context out of the device.
- #define SHA_MODE_WRITE_CONTEXT ((uint8_t)0x07)

Restore a SHA-256 context into the device.
- #define SHA_MODE_TARGET_MASK ((uint8_t)0xC0)

  Resulting digest target location mask.
- #define SHA_MODE_TARGET_TEMPKEY ((uint8_t)0x00)

  Place resulting digest both in Output buffer and TempKey.
- #define SHA_MODE_TARGET_MSGDIGBUF ((uint8_t)0x40)

  Place resulting digest both in Output buffer and Message Digest Buffer.
- #define SHA_MODE_TARGET_OUT_ONLY ((uint8_t)0xC0)

  Place resulting digest both in Output buffer ONLY.
- #define SHA_RSP_SIZE ATCA_RSP_SIZE_32

  SHA command response packet size.
- #define SHA_RSP_SIZE_SHORT ATCA_RSP_SIZE_MIN

  SHA command response packet size only status code.
- #define SHA_RSP_SIZE_LONG ATCA_RSP_SIZE_32

  SHA command response packet size.

13.2.1 Detailed Description

CryptoAuthLib command builder object, ATCACommand. Member functions for the ATCACommand object.

13.2.2 Macro Definition Documentation

13.2.2.1 AES_COUNT

```
#define AES_COUNT (23)
```

AES command packet size.

13.2.2.2 AES_DATA_SIZE

```
#define AES_DATA_SIZE (16)
```

size of AES encrypt/decrypt data

13.2.2.3 AES_INPUT_IDX

```
#define AES_INPUT_IDX ATCA_DATA_IDX
```

AES command index for input data.

13.2.2.4 AES_KEYID_IDX

```
#define AES_KEYID_IDX ATCA_PARAM2_IDX
```

AES command index for key id.

13.2.2.5 AES_MODE_DECRYPT

```
#define AES_MODE_DECRYPT ((uint8_t)0x01)
```

AES mode: Decrypt.

13.2.2.6 AES_MODE_ENCRYPT

```
#define AES_MODE_ENCRYPT ((uint8_t)0x00)
```

AES mode: Encrypt.

13.2.2.7 AES_MODE_GFM

```
#define AES_MODE_GFM ((uint8_t)0x03)
```

AES mode: GFM calculation.

13.2.2.8 AES_MODE_IDX

```
#define AES_MODE_IDX ATCA_PARAM1_IDX
```

AES command index for mode.

13.2.2.9 AES_MODE_KEY_BLOCK_MASK

```
#define AES_MODE_KEY_BLOCK_MASK ((uint8_t)0xC0)
```

AES mode mask for key block field.

13.2.2.10 AES_MODE_KEY_BLOCK_POS

```
#define AES_MODE_KEY_BLOCK_POS (6)
```

Bit shift for key block in mode.

13.2.2.11 AES_MODE_MASK

```
#define AES_MODE_MASK ((uint8_t)0xC7)
```

AES mode bits 3 to 5 are 0.

13.2.2.12 AES_MODE_OP_MASK

```
#define AES_MODE_OP_MASK ((uint8_t)0x07)
```

AES mode operation mask.

13.2.2.13 AES_RSP_SIZE

```
#define AES_RSP_SIZE ATCA_RSP_SIZE_16
```

AES command response packet size.

13.2.2.14 ATCA_ADDRESS_MASK

```
#define ATCA_ADDRESS_MASK (0x007F)
```

Address bit 7 to 15 are always 0.

13.2.2.15 ATCA_ADDRESS_MASK_CONFIG

```
#define ATCA_ADDRESS_MASK_CONFIG (0x001F)
```

Address bits 5 to 7 are 0 for Configuration zone.

13.2.2.16 ATCA_ADDRESS_MASK_OTP

```
#define ATCA_ADDRESS_MASK_OTP (0x000F)
```

Address bits 4 to 7 are 0 for OTP zone.

13.2.2.17 ATCA_AES

```
#define ATCA_AES ((uint8_t)0x51)
```

AES command op-code.

13.2.2.18 ATCA_AES_GFM_SIZE

```
#define ATCA_AES_GFM_SIZE ATCA_BLOCK_SIZE
```

size of GFM data

13.2.2.19 ATCA_AES_KEY_TYPE

```
#define ATCA_AES_KEY_TYPE 6
```

AES-128 Key.

13.2.2.20 ATCA_B283_KEY_TYPE

```
#define ATCA_B283_KEY_TYPE 0
```

B283 NIST ECC key.

13.2.2.21 ATCA_BLOCK_SIZE

```
#define ATCA_BLOCK_SIZE (32)
```

size of a block

13.2.2.22 ATCA_CHECKMAC

```
#define ATCA_CHECKMAC ((uint8_t)0x28)
```

CheckMac command op-code.

13.2.2.23 ATCA_CHIPMODE_CLOCK_DIV_M0

```
#define ATCA_CHIPMODE_CLOCK_DIV_M0 ((uint8_t)0x00)
```

ChipMode clock divider M0.

13.2.2.24 ATCA_CHIPMODE_CLOCK_DIV_M1

```
#define ATCA_CHIPMODE_CLOCK_DIV_M1 ((uint8_t)0x28)
```

ChipMode clock divider M1.

13.2.2.25 ATCA_CHIPMODE_CLOCK_DIV_M2

```
#define ATCA_CHIPMODE_CLOCK_DIV_M2 ((uint8_t)0x68)
```

ChipMode clock divider M2.

13.2.2.26 ATCA_CHIPMODE_CLOCK_DIV_MASK

```
#define ATCA_CHIPMODE_CLOCK_DIV_MASK ((uint8_t)0xF8)
```

ChipMode clock divider mask.

13.2.2.27 ATCA_CHIPMODE_I2C_ADDRESS_FLAG

```
#define ATCA_CHIPMODE_I2C_ADDRESS_FLAG ((uint8_t)0x01)
```

ChipMode I2C Address in UserExtraAdd flag.

13.2.2.28 ATCA_CHIPMODE_OFFSET

```
#define ATCA_CHIPMODE_OFFSET (19)
```

ChipMode byte offset within the configuration zone.

13.2.2.29 ATCA_CHIPMODE_TTL_ENABLE_FLAG

```
#define ATCA_CHIPMODE_TTL_ENABLE_FLAG ((uint8_t)0x02)
```

ChipMode TTLenable flag.

13.2.2.30 ATCA_CHIPMODE_WATCHDOG_LONG

```
#define ATCA_CHIPMODE_WATCHDOG_LONG ((uint8_t)0x04)
```

ChipMode long watchdog (∼13s)

13.2.2.31 ATCA_CHIPMODE_WATCHDOG_MASK

```
#define ATCA_CHIPMODE_WATCHDOG_MASK ((uint8_t)0x04)
```

ChipMode watchdog duration mask.

13.2.2.32 ATCA_CHIPMODE_WATCHDOG_SHORT

```
#define ATCA_CHIPMODE_WATCHDOG_SHORT ((uint8_t)0x00)
```

ChipMode short watchdog (∼1.3s)

13.2.2.33 ATCA_CMD_SIZE_MAX

```
#define ATCA_CMD_SIZE_MAX ((uint8_t)4 * 36 + 7)
```

maximum size of command packet (Verify)

13.2.2.34 ATCA_CMD_SIZE_MIN

```
#define ATCA_CMD_SIZE_MIN ((uint8_t)7)
```

minimum number of bytes in command (from count byte to second CRC byte)

13.2.2.35 ATCA_COUNT_IDX

```
#define ATCA_COUNT_IDX (0)
```

command packet index for count

13.2.2.36 ATCA_COUNT_SIZE

```
#define ATCA_COUNT_SIZE ((uint8_t)1)
```

Number of bytes in the command packet Count.

13.2.2.37 ATCA_COUNTER

```
#define ATCA_COUNTER ((uint8_t)0x24)
```

Counter command op-code.

13.2.2.38 ATCA_CRC_SIZE

```
#define ATCA_CRC_SIZE ((uint8_t)2)
```

Number of bytes in the command packet CRC.

13.2.2.39 ATCA_DATA_IDX

```
#define ATCA_DATA_IDX (5)
```

command packet index for data load

13.2.2.40 ATCA_DATA_SIZE

```
#define ATCA_DATA_SIZE (ATCA_KEY_COUNT * ATCA_KEY_SIZE)
```

size of data zone

13.2.2.41 ATCA_DERIVE_KEY

```
#define ATCA_DERIVE_KEY ((uint8_t)0x1C)
```

DeriveKey command op-code.

13.2.2.42 ATCA_ECC_CONFIG_SIZE

```
#define ATCA_ECC_CONFIG_SIZE (128)
```

size of configuration zone

13.2.2.43 ATCA_ECDH

```
#define ATCA_ECDH ((uint8_t)0x43)
```

ECDH command op-code.

13.2.2.44 ATCA_GENDIG

```
#define ATCA_GENDIG ((uint8_t)0x15)
```

GenDig command op-code.

13.2.2.45 ATCA_GENKEY

```
#define ATCA_GENKEY ((uint8_t)0x40)
```

GenKey command op-code.

13.2.2.46 ATCA_HMAC

```
#define ATCA_HMAC ((uint8_t)0x11)
```

HMAC command op-code.

13.2.2.47 ATCA_INFO

```
#define ATCA_INFO ((uint8_t)0x30)
```

Info command op-code.

13.2.2.48 ATCA_K283_KEY_TYPE

```
#define ATCA_K283_KEY_TYPE 1
```

K283 NIST ECC key.

13.2.2.49 ATCA_KDF

```
#define ATCA_KDF ((uint8_t)0x56)
```

KDF command op-code.

13.2.2.50 ATCA_KEY_COUNT

```
#define ATCA_KEY_COUNT (16)
```

number of keys

13.2.2.51 ATCA_KEY_ID_MAX

```
#define ATCA_KEY_ID_MAX ((uint8_t)15)
```

maximum value for key id

13.2.2.52 ATCA_KEY_SIZE

```
#define ATCA_KEY_SIZE (32)
```

size of a symmetric SHA key

13.2.2.53 ATCA_LOCK

```
#define ATCA_LOCK ((uint8_t)0x17)
```

Lock command op-code.

13.2.2.54 ATCA_LOCKED

```
#define ATCA_LOCKED (0x00)
```

Value indicating a locked zone.

13.2.2.55 ATCA_MAC

```
#define ATCA_MAC ((uint8_t)0x08)
```

MAC command op-code.

13.2.2.56 ATCA_NONCE

```
#define ATCA_NONCE ((uint8_t)0x16)
```

Nonce command op-code.

13.2.2.57 ATCA_OPCODE_IDX

```
#define ATCA_OPCODE_IDX (1)
```

command packet index for op-code

13.2.2.58 ATCA_OTP_BLOCK_MAX

```
#define ATCA_OTP_BLOCK_MAX ((uint8_t)1)
```

maximum value for OTP block

13.2.2.59 ATCA_OTP_SIZE

```
#define ATCA_OTP_SIZE (64)
```

size of OTP zone

13.2.2.60 ATCA_P256_KEY_TYPE

```
#define ATCA_P256_KEY_TYPE 4
```

P256 NIST ECC key.

13.2.2.61 ATCA_PACKET_OVERHEAD

```
#define ATCA_PACKET_OVERHEAD (ATCA_COUNT_SIZE + ATCA_CRC_SIZE)
```

Number of bytes in the command packet.

13.2.2.62 ATCA_PARAM1_IDX

```
#define ATCA_PARAM1_IDX (2)
```

command packet index for first parameter

13.2.2.63 ATCA_PARAM2_IDX

```
#define ATCA_PARAM2_IDX (3)
```

command packet index for second parameter

13.2.2.64 ATCA_PAUSE

```
#define ATCA_PAUSE ((uint8_t)0x01)
```

Pause command op-code.

13.2.2.65 ATCA_PRIV_KEY_SIZE

```
#define ATCA_PRIV_KEY_SIZE (32)
```

size of a p256 private key

13.2.2.66 ATCA_PRIVWRITE

```
#define ATCA_PRIVWRITE ((uint8_t)0x46)
```

PrivWrite command op-code.

13.2.2.67 ATCA_PUB_KEY_PAD

```
#define ATCA_PUB_KEY_PAD (4)
```

size of the public key pad

13.2.2.68 ATCA_PUB_KEY_SIZE

```
#define ATCA_PUB_KEY_SIZE (64)
```

size of a p256 public key

13.2.2.69 ATCA_RANDOM

```
#define ATCA_RANDOM ((uint8_t)0x1B)
```

Random command op-code.

13.2.2.70 ATCA_READ

```
#define ATCA_READ ((uint8_t)0x02)
```

Read command op-code.

13.2.2.71 ATCA_RSP_DATA_IDX

```
#define ATCA_RSP_DATA_IDX (1)
```

buffer index of data in response

13.2.2.72 ATCA_RSP_SIZE_16

```
#define ATCA_RSP_SIZE_16 ((uint8_t)19)
```

size of response packet containing 16 bytes data

13.2.2.73 ATCA_RSP_SIZE_32

```
#define ATCA_RSP_SIZE_32 ((uint8_t)35)
```

size of response packet containing 32 bytes data

13.2.2.74 ATCA_RSP_SIZE_4

```
#define ATCA_RSP_SIZE_4 ((uint8_t)7)
```

size of response packet containing 4 bytes data

13.2.2.75 ATCA_RSP_SIZE_64

```
#define ATCA_RSP_SIZE_64 ((uint8_t)67)
```

size of response packet containing 64 bytes data

13.2.2.76 ATCA_RSP_SIZE_72

```
#define ATCA_RSP_SIZE_72 ((uint8_t)75)
```

size of response packet containing 64 bytes data

13.2.2.77 ATCA_RSP_SIZE_MAX

```
#define ATCA_RSP_SIZE_MAX ((uint8_t)75)
```

maximum size of response packet (GenKey and Verify command)

13.2.2.78 ATCA_RSP_SIZE_MIN

```
#define ATCA_RSP_SIZE_MIN ((uint8_t)4)
```

minimum number of bytes in response

13.2.2.79 ATCA_RSP_SIZE_VAL

```
#define ATCA_RSP_SIZE_VAL ((uint8_t)7)
```

size of response packet containing four bytes of data

13.2.2.80 ATCA_SECUREBOOT

```
#define ATCA_SECUREBOOT ((uint8_t)0x80)
```

Secure Boot command op-code.

13.2.2.81 ATCA_SELFTEST

```
#define ATCA_SELFTEST ((uint8_t)0x77)
```

Self test command op-code.

13.2.2.82 ATCA_SERIAL_NUM_SIZE

```
#define ATCA_SERIAL_NUM_SIZE (9)
```

number of bytes in the device serial number

13.2.2.83 ATCA_SHA

```
#define ATCA_SHA ((uint8_t)0x47)
```

SHA command op-code.

13.2.2.84 ATCA_SHA256_BLOCK_SIZE

```
#define ATCA_SHA256_BLOCK_SIZE (64)
```

13.2.2.85 ATCA_SHA_CONFIG_SIZE

```
#define ATCA_SHA_CONFIG_SIZE (88)
```

size of configuration zone

13.2.2.86 ATCA_SHA_DIGEST_SIZE

```
#define ATCA_SHA_DIGEST_SIZE (32)
```

13.2.2.87 ATCA_SHA_KEY_TYPE

```
#define ATCA_SHA_KEY_TYPE 7
```

SHA key or other data.

13.2.2.88 ATCA_SIG_SIZE

```
#define ATCA_SIG_SIZE (64)
```

size of a p256 signature

13.2.2.89 ATCA_SIGN

```
#define ATCA_SIGN ((uint8_t)0x41)
```

Sign command op-code.

13.2.2.90 ATCA_TEMPKEY_KEYID

```
#define ATCA_TEMPKEY_KEYID (0xFFFF)
```

KeyID when referencing TempKey.

13.2.2.91 ATCA_UNLOCKED

```
#define ATCA_UNLOCKED (0x55)
```

Value indicating an unlocked zone.

13.2.2.92 ATCA_UPDATE_EXTRA

```
#define ATCA_UPDATE_EXTRA ((uint8_t)0x20)
```

UpdateExtra command op-code.

13.2.2.93 ATCA_VERIFY

```
#define ATCA_VERIFY ((uint8_t)0x45)
```

GenKey command op-code.

13.2.2.94 ATCA_WORD_SIZE

```
#define ATCA_WORD_SIZE (4)
```

size of a word

13.2.2.95 ATCA_WRITE

```
#define ATCA_WRITE ((uint8_t)0x12)
```

Write command op-code.

13.2.2.96 ATCA_ZONE_CONFIG

```
#define ATCA_ZONE_CONFIG ((uint8_t)0x00)
```

Configuration zone.

13.2.2.97 ATCA_ZONE_DATA

```
#define ATCA_ZONE_DATA ((uint8_t)0x02)
```

Data zone.

13.2.2.98 ATCA_ZONE_ENCRYPTED

```
#define ATCA_ZONE_ENCRYPTED ((uint8_t)0x40)
```

Zone bit 6 set: Write is encrypted with an unlocked data zone.

13.2.2.99 ATCA_ZONE_MASK

```
#define ATCA_ZONE_MASK ((uint8_t)0x03)
```

Zone mask.

13.2.2.100 ATCA_ZONE_OTP

```
#define ATCA_ZONE_OTP ((uint8_t)0x01)
```

OTP (One Time Programming) zone.

13.2.2.101 ATCA_ZONE_READWRITE_32

```
#define ATCA_ZONE_READWRITE_32 ((uint8_t)0x80)
```

Zone bit 7 set: Access 32 bytes, otherwise 4 bytes.

13.2.2.102 CHECKMAC_CLIENT_CHALLENGE_IDX

```
#define CHECKMAC_CLIENT_CHALLENGE_IDX ATCA_DATA_IDX
```

CheckMAC command index for client challenge.

13.2.2.103 CHECKMAC_CLIENT_CHALLENGE_SIZE

```
#define CHECKMAC_CLIENT_CHALLENGE_SIZE (32)
```

CheckMAC size of client challenge.

13.2.2.104 CHECKMAC_CLIENT_COMMAND_SIZE

```
#define CHECKMAC_CLIENT_COMMAND_SIZE (4)
```

CheckMAC size of client command header size inside "other data".

13.2.2.105 CHECKMAC_CLIENT_RESPONSE_IDX

```
#define CHECKMAC_CLIENT_RESPONSE_IDX (37)
```

CheckMAC command index for client response.

13.2.2.106 CHECKMAC_CLIENT_RESPONSE_SIZE

```
#define CHECKMAC_CLIENT_RESPONSE_SIZE (32)
```

CheckMAC size of client response.

13.2.2.107 CHECKMAC_CMD_MATCH

```
#define CHECKMAC_CMD_MATCH (0)
```

CheckMAC return value when there is a match.

13.2.2.108 CHECKMAC_CMD_MISMATCH

```
#define CHECKMAC_CMD_MISMATCH (1)
```

CheckMAC return value when there is a mismatch.

13.2.2.109 CHECKMAC_COUNT

```
#define CHECKMAC_COUNT (84)
```

CheckMAC command packet size.

13.2.2.110 CHECKMAC_DATA_IDX

```
#define CHECKMAC_DATA_IDX (69)
```

CheckMAC command index for other data.

13.2.2.111 CHECKMAC_KEYID_IDX

```
#define CHECKMAC_KEYID_IDX ATCA_PARAM2_IDX
```

CheckMAC command index for key identifier.

13.2.2.112 CHECKMAC_MODE_BLOCK1_TEMPKEY

```
#define CHECKMAC_MODE_BLOCK1_TEMPKEY ((uint8_t)0x02)
```

CheckMAC mode bit 1: first SHA block from TempKey.

13.2.2.113 CHECKMAC_MODE_BLOCK2_TEMPKEY

```
#define CHECKMAC_MODE_BLOCK2_TEMPKEY ((uint8_t)0x01)
```

CheckMAC mode bit 0: second SHA block from TempKey.

13.2.2.114 CHECKMAC_MODE_CHALLENGE

```
#define CHECKMAC_MODE_CHALLENGE ((uint8_t)0x00)
```

CheckMAC mode 0: first SHA block from key id.

13.2.2.115 CHECKMAC_MODE_IDX

```
#define CHECKMAC_MODE_IDX ATCA_PARAM1_IDX
```

CheckMAC command index for mode.

13.2.2.116 CHECKMAC_MODE_INCLUDE_OTP_64

```
#define CHECKMAC_MODE_INCLUDE_OTP_64 ((uint8_t)0x20)
```

CheckMAC mode bit 5: include first 64 OTP bits.

13.2.2.117 CHECKMAC_MODE_MASK

```
#define CHECKMAC_MODE_MASK ((uint8_t)0x27)
```

CheckMAC mode bits 3, 4, 6, and 7 are 0.

13.2.2.118 CHECKMAC_MODE_SOURCE_FLAG_MATCH

```
#define CHECKMAC_MODE_SOURCE_FLAG_MATCH ((uint8_t)0x04)
```

CheckMAC mode bit 2: match TempKey.SourceFlag.

13.2.2.119 CHECKMAC_OTHER_DATA_SIZE

```
#define CHECKMAC_OTHER_DATA_SIZE (13)
```

CheckMAC size of "other data".

13.2.2.120 CHECKMAC_RSP_SIZE

```
#define CHECKMAC_RSP_SIZE ATCA_RSP_SIZE_MIN
```

CheckMAC response packet size.

13.2.2.121 CMD_STATUS_BYTE_COMM

```
#define CMD_STATUS_BYTE_COMM ((uint8_t)0xFF)
```

communication error

13.2.2.122 CMD_STATUS_BYTE_ECC

```
#define CMD_STATUS_BYTE_ECC ((uint8_t)0x05)
```

command ECC error

13.2.2.123 CMD_STATUS_BYTE_EXEC

```
#define CMD_STATUS_BYTE_EXEC ((uint8_t)0x0F)
```

command execution error

13.2.2.124 CMD_STATUS_BYTE_PARSE

```
#define CMD_STATUS_BYTE_PARSE ((uint8_t)0x03)
```

command parse error

13.2.2.125 CMD_STATUS_SUCCESS

```
#define CMD_STATUS_SUCCESS ((uint8_t)0x00)
```

status byte for success

13.2.2.126 CMD_STATUS_WAKEUP

```
#define CMD_STATUS_WAKEUP ((uint8_t)0x11)
```

status byte after wake-up

13.2.2.127 COUNTER_COUNT

```
#define COUNTER_COUNT ATCA_CMD_SIZE_MIN
```

13.2.2.128 COUNTER_KEYID_IDX

```
#define COUNTER_KEYID_IDX ATCA_PARAM2_IDX
```

Counter command index for key id.

13.2.2.129 COUNTER_MAX_VALUE

```
#define COUNTER_MAX_VALUE ((uint32_t)2097151)
```

Counter maximum value of the counter.

13.2.2.130 COUNTER_MODE_IDX

```
#define COUNTER_MODE_IDX ATCA_PARAM1_IDX
```

Counter command index for mode.

13.2.2.131 COUNTER_MODE_INCREMENT

```
#define COUNTER_MODE_INCREMENT ((uint8_t)0x01)
```

Counter command mode for incrementing.

13.2.2.132 COUNTER_MODE_MASK

```
#define COUNTER_MODE_MASK ((uint8_t)0x01)
```

Counter mode bits 1 to 7 are 0.

13.2.2.133 COUNTER_MODE_READ

```
#define COUNTER_MODE_READ ((uint8_t)0x00)
```

Counter command mode for reading.

13.2.2.134 COUNTER_RSP_SIZE

```
#define COUNTER_RSP_SIZE ATCA_RSP_SIZE_4
```

Counter command response packet size.

13.2.2.135 COUNTER_SIZE

```
#define COUNTER_SIZE ATCA_RSP_SIZE_MIN
```

Counter size in binary.

13.2.2.136 DERIVE_KEY_COUNT_LARGE

```
#define DERIVE_KEY_COUNT_LARGE (39)
```

DeriveKey command packet size with MAC.

13.2.2.137 DERIVE_KEY_COUNT_SMALL

```
#define DERIVE_KEY_COUNT_SMALL ATCA_CMD_SIZE_MIN
```

DeriveKey command packet size without MAC.

13.2.2.138 DERIVE_KEY_MAC_IDX

```
#define DERIVE_KEY_MAC_IDX ATCA_DATA_IDX
```

DeriveKey command index for optional MAC.

13.2.2.139 DERIVE_KEY_MAC_SIZE

```
#define DERIVE_KEY_MAC_SIZE (32)
```

DeriveKey MAC size.

13.2.2.140 DERIVE_KEY_MODE

```
#define DERIVE_KEY_MODE ((uint8_t)0x04)
```

DeriveKey command mode set to 4 as in datasheet.

13.2.2.141 DERIVE_KEY_RANDOM_FLAG

```
#define DERIVE_KEY_RANDOM_FLAG ((uint8_t)4)
```

DeriveKey 1. parameter; has to match TempKey.SourceFlag.

13.2.2.142 DERIVE_KEY_RANDOM_IDX

```
#define DERIVE_KEY_RANDOM_IDX ATCA_PARAM1_IDX
```

DeriveKey command index for random bit.

13.2.2.143 DERIVE_KEY_RSP_SIZE

```
#define DERIVE_KEY_RSP_SIZE ATCA_RSP_SIZE_MIN
```

DeriveKey response packet size.

13.2.2.144 DERIVE_KEY_TARGETKEY_IDX

```
#define DERIVE_KEY_TARGETKEY_IDX ATCA_PARAM2_IDX
```

DeriveKey command index for target slot.

13.2.2.145 ECDH_COUNT

```
#define ECDH_COUNT (ATCA_CMD_SIZE_MIN + ATCA_PUB_KEY_SIZE)
```

13.2.2.146 ECDH_KEY_SIZE

```
#define ECDH_KEY_SIZE ATCA_BLOCK_SIZE
```

ECDH output data size.

13.2.2.147 ECDH_MODE_COPY_COMPATIBLE

```
#define ECDH_MODE_COPY_COMPATIBLE ((uint8_t)0x00)
```

13.2.2.148 ECDH_MODE_COPY_EEPROM_SLOT

```
#define ECDH_MODE_COPY_EEPROM_SLOT ((uint8_t)0x04)
```

13.2.2.149 ECDH_MODE_COPY_MASK

```
#define ECDH_MODE_COPY_MASK ((uint8_t)0x0C)
```

13.2.2.150 ECDH_MODE_COPY_OUTPUT_BUFFER

```
#define ECDH_MODE_COPY_OUTPUT_BUFFER ((uint8_t)0x0C)
```

13.2.2.151 ECDH_MODE_COPY_TEMP_KEY

```
#define ECDH_MODE_COPY_TEMP_KEY ((uint8_t)0x08)
```

13.2.2.152 ECDH_MODE_OUTPUT_CLEAR

```
#define ECDH_MODE_OUTPUT_CLEAR ((uint8_t)0x00)
```

13.2.2.153 ECDH_MODE_OUTPUT_ENC

```
#define ECDH_MODE_OUTPUT_ENC ((uint8_t)0x02)
```

13.2.2.154 ECDH_MODE_OUTPUT_MASK

```
#define ECDH_MODE_OUTPUT_MASK ((uint8_t)0x02)
```

13.2.2.155 ECDH_MODE_SOURCE_EEPROM_SLOT

```
#define ECDH_MODE_SOURCE_EEPROM_SLOT ((uint8_t)0x00)
```

13.2.2.156 ECDH_MODE_SOURCE_MASK

```
#define ECDH_MODE_SOURCE_MASK ((uint8_t)0x01)
```

13.2.2.157 ECDH_MODE_SOURCE_TEMPKEY

```
#define ECDH_MODE_SOURCE_TEMPKEY ((uint8_t)0x01)
```

13.2.2.158 ECDH_PREFIX_MODE

```
#define ECDH_PREFIX_MODE ((uint8_t)0x00)
```

13.2.2.159 ECDH_RSP_SIZE

```
#define ECDH_RSP_SIZE ATCA_RSP_SIZE_64
```

ECDH command packet size.

13.2.2.160 GENDIG_COUNT

```
#define GENDIG_COUNT ATCA_CMD_SIZE_MIN
```

GenDig command packet size without "other data".

13.2.2.161 GENDIG_DATA_IDX

```
#define GENDIG_DATA_IDX ATCA_DATA_IDX
```

GenDig command index for optional data.

13.2.2.162 GENDIG_KEYID_IDX

```
#define GENDIG_KEYID_IDX ATCA_PARAM2_IDX
```

GenDig command index for key id.

13.2.2.163 GENDIG_RSP_SIZE

```
#define GENDIG_RSP_SIZE ATCA_RSP_SIZE_MIN
```

GenDig command response packet size.

13.2.2.164 GENDIG_ZONE_CONFIG

```
#define GENDIG_ZONE_CONFIG ((uint8_t)0)
```

GenDig zone id config. Use KeyID to specify any of the four 256-bit blocks of the Configuration zone.

13.2.2.165 GENDIG_ZONE_COUNTER

```
#define GENDIG_ZONE_COUNTER ((uint8_t)4)
```

GenDig zone id counter. KeyID specifies the monotonic counter ID to be included in the message generation.

13.2.2.166 GENDIG_ZONE_DATA

```
#define GENDIG_ZONE_DATA ((uint8_t)2)
```

GenDig zone id data. Use KeyID to specify a slot in the Data zone or a transport key in the hardware array.

13.2.2.167 GENDIG_ZONE_IDX

```
#define GENDIG_ZONE_IDX ATCA_PARAM1_IDX
```

GenDig command index for zone.

13.2.2.168 GENDIG_ZONE_KEY_CONFIG

```
#define GENDIG_ZONE_KEY_CONFIG ((uint8_t)5)
```

GenDig zone id key config. KeyID specifies the slot for which the configuration information is to be included in the message generation.

13.2.2.169 GENDIG_ZONE_OTP

```
#define GENDIG_ZONE_OTP ((uint8_t)1)
```

GenDig zone id OTP. Use KeyID to specify either the first or second 256-bit block of the OTP zone.

13.2.2.170 GENDIG_ZONE_SHARED_NONCE

```
#define GENDIG_ZONE_SHARED_NONCE ((uint8_t)3)
```

GenDig zone id shared nonce. KeyID specifies the location of the input value in the message generation.

13.2.2.171 GENKEY_COUNT

```
#define GENKEY_COUNT ATCA_CMD_SIZE_MIN
```

GenKey command packet size without "other data".

13.2.2.172 GENKEY_COUNT_DATA

```
#define GENKEY_COUNT_DATA (10)
```

GenKey command packet size with "other data".

13.2.2.173 GENKEY_DATA_IDX

```
#define GENKEY_DATA_IDX (5)
```

GenKey command index for other data.

13.2.2.174 GENKEY_KEYID_IDX

```
#define GENKEY_KEYID_IDX ATCA_PARAM2_IDX
```

GenKey command index for key id.

13.2.2.175 GENKEY_MODE_DIGEST

```
#define GENKEY_MODE_DIGEST ((uint8_t)0x08)
```

GenKey mode: PubKey digest will be created after the public key is calculated.

13.2.2.176 GENKEY_MODE_IDX

```
#define GENKEY_MODE_IDX ATCA_PARAM1_IDX
```

GenKey command index for mode.

13.2.2.177 GENKEY_MODE_MASK

```
#define GENKEY_MODE_MASK ((uint8_t)0x1C)
```

GenKey mode bits 0 to 1 and 5 to 7 are 0.

13.2.2.178 GENKEY_MODE_PRIVATE

```
#define GENKEY_MODE_PRIVATE ((uint8_t)0x04)
```

GenKey mode: private key generation.

13.2.2.179 GENKEY_MODE_PUBKEY_DIGEST

```
#define GENKEY_MODE_PUBKEY_DIGEST ((uint8_t)0x10)
```

GenKey mode: Calculate PubKey digest on the public key in KeyId.

13.2.2.180 GENKEY_MODE_PUBLIC

```
#define GENKEY_MODE_PUBLIC ((uint8_t)0x00)
```

GenKey mode: public key calculation.

13.2.2.181 GENKEY_OTHER_DATA_SIZE

```
#define GENKEY_OTHER_DATA_SIZE (3)
```

GenKey size of "other data".

13.2.2.182 GENKEY_PRIVATE_TO_TEMPKEY

```
#define GENKEY_PRIVATE_TO_TEMPKEY ((uint16_t)0xFFFF)
```

GenKey Create private key and store to tempkey (608 only)

13.2.2.183 GENKEY_RSP_SIZE_LONG

```
#define GENKEY_RSP_SIZE_LONG ATCA_RSP_SIZE_64
```

GenKey response packet size when returning a public key.

13.2.2.184 GENKEY_RSP_SIZE_SHORT

```
#define GENKEY_RSP_SIZE_SHORT ATCA_RSP_SIZE_MIN
```

GenKey response packet size in Digest mode.

13.2.2.185 HMAC_COUNT

```
#define HMAC_COUNT ATCA_CMD_SIZE_MIN
```

HMAC command packet size.

13.2.2.186 HMAC_DIGEST_SIZE

```
#define HMAC_DIGEST_SIZE (32)
```

HMAC size of digest response.

13.2.2.187 HMAC_KEYID_IDX

```
#define HMAC_KEYID_IDX ATCA_PARAM2_IDX
```

HMAC command index for key id.

13.2.2.188 HMAC_MODE_FLAG_FULLSN

```
#define HMAC_MODE_FLAG_FULLSN ((uint8_t)0x40)
```

HMAC mode bit 6: If set, include the 48 bits SN[2:3] and SN[4:7] in the message.; otherwise, the corresponding message bits are set to zero.

13.2.2.189 HMAC_MODE_FLAG_OTP64

```
#define HMAC_MODE_FLAG_OTP64 ((uint8_t)0x20)
```

HMAC mode bit 5: Include the first 64 OTP bits (OTP[0] through OTP[7]) in the message.; otherwise, the corresponding message bits are set to zero. If Mode[4] is set, the value of this mode bit is ignored. Not applicable for ATECC508A.

13.2.2.190 HMAC_MODE_FLAG_OTP88

```
#define HMAC_MODE_FLAG_OTP88 ((uint8_t)0x10)
```

HMAC mode bit 4: Include the first 88 OTP bits (OTP[0] through OTP[10]) in the message.; otherwise, the corresponding message bits are set to zero. Not applicable for ATECC508A.

13.2.2.191 HMAC_MODE_FLAG_TK_NORAND

```
#define HMAC_MODE_FLAG_TK_NORAND ((uint8_t)0x04)
```

HMAC mode bit 2: The value of this bit must match the value in TempKey.SourceFlag or the command will return an error.

13.2.2.192 HMAC_MODE_FLAG_TK_RAND

```
#define HMAC_MODE_FLAG_TK_RAND ((uint8_t)0x00)
```

HMAC mode bit 2: The value of this bit must match the value in TempKey.SourceFlag or the command will return an error.

13.2.2.193 HMAC_MODE_IDX

```
#define HMAC_MODE_IDX ATCA_PARAM1_IDX
```

HMAC command index for mode.

13.2.2.194 HMAC_MODE_MASK

```
#define HMAC_MODE_MASK ((uint8_t)0x74)
```

HMAC mode bits 0, 1, 3, and 7 are 0.

13.2.2.195 HMAC_RSP_SIZE

```
#define HMAC_RSP_SIZE ATCA_RSP_SIZE_32
```

HMAC command response packet size.

13.2.2.196 INFO_COUNT

```
#define INFO_COUNT ATCA_CMD_SIZE_MIN
```

Info command packet size.

13.2.2.197 INFO_DRIVER_STATE_MASK

```
#define INFO_DRIVER_STATE_MASK ((uint8_t)0x02)
```

Info driver state mask.

13.2.2.198 INFO_MODE_GPIO

```
#define INFO_MODE_GPIO ((uint8_t)0x03)
```

Info mode GPIO.

13.2.2.199 INFO_MODE_KEY_VALID

```
#define INFO_MODE_KEY_VALID ((uint8_t)0x01)
```

Info mode KeyValid.

13.2.2.200 INFO_MODE_MAX

```
#define INFO_MODE_MAX ((uint8_t)0x03)
```

Info mode maximum value.

13.2.2.201 INFO_MODE_REVISION

```
#define INFO_MODE_REVISION ((uint8_t)0x00)
```

Info mode Revision.

13.2.2.202 INFO_MODE_STATE

```
#define INFO_MODE_STATE ((uint8_t)0x02)
```

Info mode State.

13.2.2.203 INFO_MODE_VOL_KEY_PERMIT

```
#define INFO_MODE_VOL_KEY_PERMIT ((uint8_t)0x04)
```

Info mode GPIO.

13.2.2.204 INFO_NO_STATE

```
#define INFO_NO_STATE ((uint8_t)0x00)
```

Info mode is not the state mode.

13.2.2.205 INFO_OUTPUT_STATE_MASK

```
#define INFO_OUTPUT_STATE_MASK ((uint8_t)0x01)
```

Info output state mask.

13.2.2.206 INFO_PARAM1_IDX

```
#define INFO_PARAM1_IDX ATCA_PARAM1_IDX
```

Info command index for 1. parameter.

13.2.2.207 INFO_PARAM2_IDX

```
#define INFO_PARAM2_IDX ATCA_PARAM2_IDX
```

Info command index for 2. parameter.

13.2.2.208 INFO_PARAM2_LATCH_CLEAR

```
#define INFO_PARAM2_LATCH_CLEAR ((uint16_t)0x0000)
```

Info param2 to clear the persistent latch.

13.2.2.209 INFO_PARAM2_LATCH_SET

```
#define INFO_PARAM2_LATCH_SET ((uint16_t)0x0001)
```

Info param2 to set the persistent latch.

13.2.2.210 INFO_PARAM2_SET_LATCH_STATE

```
#define INFO_PARAM2_SET_LATCH_STATE ((uint16_t)0x0002)
```

Info param2 to set the persistent latch state.

13.2.2.211 INFO_RSP_SIZE

```
#define INFO_RSP_SIZE ATCA_RSP_SIZE_VAL
```

Info command response packet size.

13.2.2.212 INFO_SIZE

```
#define INFO_SIZE ((uint8_t)0x04)
```

Info return size.

13.2.2.213 KDF_DETAILS_AES_KEY_LOC_MASK

```
#define KDF_DETAILS_AES_KEY_LOC_MASK ((uint32_t)0x00000003)
```

KDF details for AES, key location mask.

13.2.2.214 KDF_DETAILS_HKDF_MSG_LOC_INPUT

```
#define KDF_DETAILS_HKDF_MSG_LOC_INPUT ((uint32_t)0x00000002)
```

KDF details for HKDF, message location in input parameter.

13.2.2.215 KDF_DETAILS_HKDF_MSG_LOC_IV

```
#define KDF_DETAILS_HKDF_MSG_LOC_IV ((uint32_t)0x00000003)
```

KDF details for HKDF, message location is a special IV function.

13.2.2.216 KDF_DETAILS_HKDF_MSG_LOC_MASK

```
#define KDF_DETAILS_HKDF_MSG_LOC_MASK ((uint32_t)0x00000003)
```

KDF details for HKDF, message location mask.

13.2.2.217 KDF_DETAILS_HKDF_MSG_LOC_SLOT

```
#define KDF_DETAILS_HKDF_MSG_LOC_SLOT ((uint32_t)0x00000000)
```

KDF details for HKDF, message location in slot.

13.2.2.218 KDF_DETAILS_HKDF_MSG_LOC_TEMPKEY

```
#define KDF_DETAILS_HKDF_MSG_LOC_TEMPKEY ((uint32_t)0x00000001)
```

KDF details for HKDF, message location in TempKey.

13.2.2.219 KDF_DETAILS_HKDF_ZERO_KEY

```
#define KDF_DETAILS_HKDF_ZERO_KEY ((uint32_t)0x00000004)
```

KDF details for HKDF, key is 32 bytes of zero.

13.2.2.220 KDF_DETAILS_IDX

```
#define KDF_DETAILS_IDX ATCA_DATA_IDX
```

KDF command index for details.

13.2.2.221 KDF_DETAILS_PRF_AEAD_MASK

```
#define KDF_DETAILS_PRF_AEAD_MASK ((uint32_t)0x00000600)
```

KDF details for PRF, AEAD processing mask.

13.2.2.222 KDF_DETAILS_PRF_AEAD_MODE0

```
#define KDF_DETAILS_PRF_AEAD_MODE0 ((uint32_t)0x00000000)
```

KDF details for PRF, AEAD no processing.

13.2.2.223 KDF_DETAILS_PRF_AEAD_MODE1

```
#define KDF_DETAILS_PRF_AEAD_MODE1 ((uint32_t)0x00000200)
```

KDF details for PRF, AEAD First 32 go to target, second 32 go to output buffer.

13.2.2.224 KDF_DETAILS_PRF_KEY_LEN_16

```
#define KDF_DETAILS_PRF_KEY_LEN_16 ((uint32_t)0x00000000)
```

KDF details for PRF, source key length is 16 bytes.

13.2.2.225 KDF_DETAILS_PRF_KEY_LEN_32

```
#define KDF_DETAILS_PRF_KEY_LEN_32 ((uint32_t)0x00000001)
```

KDF details for PRF, source key length is 32 bytes.

13.2.2.226 KDF_DETAILS_PRF_KEY_LEN_48

```
#define KDF_DETAILS_PRF_KEY_LEN_48 ((uint32_t)0x00000002)
```

KDF details for PRF, source key length is 48 bytes.

13.2.2.227 KDF_DETAILS_PRF_KEY_LEN_64

```
#define KDF_DETAILS_PRF_KEY_LEN_64 ((uint32_t)0x00000003)
```

KDF details for PRF, source key length is 64 bytes.

13.2.2.228 KDF_DETAILS_PRF_KEY_LEN_MASK

```
#define KDF_DETAILS_PRF_KEY_LEN_MASK ((uint32_t)0x00000003)
```

KDF details for PRF, source key length mask.

13.2.2.229 KDF_DETAILS_PRF_TARGET_LEN_32

```
#define KDF_DETAILS_PRF_TARGET_LEN_32 ((uint32_t)0x00000000)
```

KDF details for PRF, target length is 32 bytes.

13.2.2.230 KDF_DETAILS_PRF_TARGET_LEN_64

```
#define KDF_DETAILS_PRF_TARGET_LEN_64 ((uint32_t)0x00000100)
```

KDF details for PRF, target length is 64 bytes.

13.2.2.231 KDF_DETAILS_PRF_TARGET_LEN_MASK

```
#define KDF_DETAILS_PRF_TARGET_LEN_MASK ((uint32_t)0x00000100)
```

KDF details for PRF, target length mask.

13.2.2.232 KDF_DETAILS_SIZE

```
#define KDF_DETAILS_SIZE 4
```

KDF details (param3) size.

13.2.2.233 KDF_KEYID_IDX

```
#define KDF_KEYID_IDX ATCA_PARAM2_IDX
```

KDF command index for key id.

13.2.2.234 KDF_MESSAGE_IDX

```
#define KDF_MESSAGE_IDX (ATCA_DATA_IDX + KDF_DETAILS_SIZE)
```

13.2.2.235 KDF_MODE_ALG_AES

```
#define KDF_MODE_ALG_AES ((uint8_t)0x20)
```

KDF mode AES algorithm.

13.2.2.236 KDF_MODE_ALG_HKDF

```
#define KDF_MODE_ALG_HKDF ((uint8_t)0x40)
```

KDF mode HKDF algorithm.

13.2.2.237 KDF_MODE_ALG_MASK

```
#define KDF_MODE_ALG_MASK ((uint8_t)0x60)
```

KDF mode algorithm mask.

13.2.2.238 KDF_MODE_ALG_PRF

```
#define KDF_MODE_ALG_PRF ((uint8_t)0x00)
```

KDF mode PRF algorithm.

13.2.2.239 KDF_MODE_IDX

```
#define KDF_MODE_IDX ATCA_PARAM1_IDX
```

KDF command index for mode.

13.2.2.240 KDF_MODE_SOURCE_ALTKEYBUF

```
#define KDF_MODE_SOURCE_ALTKEYBUF ((uint8_t)0x03)
```

KDF mode source key in alternate key buffer.

13.2.2.241 KDF_MODE_SOURCE_MASK

```
#define KDF_MODE_SOURCE_MASK ((uint8_t)0x03)
```

KDF mode source key mask.

13.2.2.242 KDF_MODE_SOURCE_SLOT

```
#define KDF_MODE_SOURCE_SLOT ((uint8_t)0x02)
```

KDF mode source key in a slot.

13.2.2.243 KDF_MODE_SOURCE_TEMPKEY

```
#define KDF_MODE_SOURCE_TEMPKEY ((uint8_t)0x00)
```

KDF mode source key in TempKey.

13.2.2.244 KDF_MODE_SOURCE_TEMPKEY_UP

```
#define KDF_MODE_SOURCE_TEMPKEY_UP ((uint8_t)0x01)
```

KDF mode source key in upper TempKey.

13.2.2.245 KDF_MODE_TARGET_ALTKEYBUF

```
#define KDF_MODE_TARGET_ALTKEYBUF ((uint8_t)0x0C)
```

KDF mode target key in alternate key buffer.

13.2.2.246 KDF_MODE_TARGET_MASK

```
#define KDF_MODE_TARGET_MASK ((uint8_t)0x1C)
```

KDF mode target key mask.

13.2.2.247 KDF_MODE_TARGET_OUTPUT

```
#define KDF_MODE_TARGET_OUTPUT ((uint8_t)0x10)
```

KDF mode target key in output buffer.

13.2.2.248 KDF_MODE_TARGET_OUTPUT_ENC

```
#define KDF_MODE_TARGET_OUTPUT_ENC ((uint8_t)0x14)
```

KDF mode target key encrypted in output buffer.

13.2.2.249 KDF_MODE_TARGET_SLOT

```
#define KDF_MODE_TARGET_SLOT ((uint8_t)0x08)
```

KDF mode target key in slot.

13.2.2.250 KDF_MODE_TARGET_TEMPKEY

```
#define KDF_MODE_TARGET_TEMPKEY ((uint8_t)0x00)
```

KDF mode target key in TempKey.

13.2.2.251 KDF_MODE_TARGET_TEMPKEY_UP

```
#define KDF_MODE_TARGET_TEMPKEY_UP ((uint8_t)0x04)
```

KDF mode target key in upper TempKey.

13.2.2.252 LOCK_COUNT

```
#define LOCK_COUNT ATCA_CMD_SIZE_MIN
```

Lock command packet size.

13.2.2.253 LOCK_RSP_SIZE

```
#define LOCK_RSP_SIZE ATCA_RSP_SIZE_MIN
```

Lock command response packet size.

13.2.2.254 LOCK_SUMMARY_IDX

```
#define LOCK_SUMMARY_IDX ATCA_PARAM2_IDX
```

Lock command index for summary.

13.2.2.255 LOCK_ZONE_CONFIG

```
#define LOCK_ZONE_CONFIG ((uint8_t)0x00)
```

Lock zone is Config.

13.2.2.256 LOCK_ZONE_DATA

```
#define LOCK_ZONE_DATA ((uint8_t)0x01)
```

Lock zone is OTP or Data.

13.2.2.257 LOCK_ZONE_DATA_SLOT

```
#define LOCK_ZONE_DATA_SLOT ((uint8_t)0x02)
```

Lock slot of Data.

13.2.2.258 LOCK_ZONE_IDX

```
#define LOCK_ZONE_IDX ATCA_PARAM1_IDX
```

Lock command index for zone.

13.2.2.259 LOCK_ZONE_MASK

```
#define LOCK_ZONE_MASK (0xBF)
```

Lock parameter 1 bits 6 are 0.

13.2.2.260 LOCK_ZONE_NO_CRC

```
#define LOCK_ZONE_NO_CRC ((uint8_t)0x80)
```

Lock command: Ignore summary.

13.2.2.261 MAC_CHALLENGE_IDX

```
#define MAC_CHALLENGE_IDX ATCA_DATA_IDX
```

MAC command index for optional challenge.

13.2.2.262 MAC_CHALLENGE_SIZE

```
#define MAC_CHALLENGE_SIZE (32)
```

MAC size of challenge.

13.2.2.263 MAC_COUNT_LONG

```
#define MAC_COUNT_LONG (39)
```

MAC command packet size with challenge.

13.2.2.264 MAC_COUNT_SHORT

```
#define MAC_COUNT_SHORT ATCA_CMD_SIZE_MIN
```

MAC command packet size without challenge.

13.2.2.265 MAC_KEYID_IDX

```
#define MAC_KEYID_IDX ATCA_PARAM2_IDX
```

MAC command index for key id.

13.2.2.266 MAC_MODE_BLOCK1_TEMPKEY

```
#define MAC_MODE_BLOCK1_TEMPKEY ((uint8_t)0x02)
```

MAC mode bit 1: first SHA block from TempKey.

13.2.2.267 MAC_MODE_BLOCK2_TEMPKEY

```
#define MAC_MODE_BLOCK2_TEMPKEY ((uint8_t)0x01)
```

MAC mode bit 0: second SHA block from TempKey.

13.2.2.268 MAC_MODE_CHALLENGE

```
#define MAC_MODE_CHALLENGE ((uint8_t)0x00)
```

MAC mode 0: first SHA block from data slot.

13.2.2.269 MAC_MODE_IDX

```
#define MAC_MODE_IDX ATCA_PARAM1_IDX
```

MAC command index for mode.

13.2.2.270 MAC_MODE_INCLUDE_OTP_64

```
#define MAC_MODE_INCLUDE_OTP_64 ((uint8_t)0x20)
```

MAC mode bit 5: include first 64 OTP bits.

13.2.2.271 MAC_MODE_INCLUDE_OTP_88

```
#define MAC_MODE_INCLUDE_OTP_88 ((uint8_t)0x10)
```

MAC mode bit 4: include first 88 OTP bits.

13.2.2.272 MAC_MODE_INCLUDE_SN

```
#define MAC_MODE_INCLUDE_SN ((uint8_t)0x40)
```

MAC mode bit 6: include serial number.

13.2.2.273 MAC_MODE_MASK

```
#define MAC_MODE_MASK ((uint8_t)0x77)
```

MAC mode bits 3 and 7 are 0.

13.2.2.274 MAC_MODE_PASSTHROUGH

```
#define MAC_MODE_PASSTHROUGH ((uint8_t)0x07)
```

MAC mode bit 0-2: pass-through mode.

13.2.2.275 MAC_MODE_PTNONCE_TEMPKEY

```
#define MAC_MODE_PTNONCE_TEMPKEY ((uint8_t)0x06)
```

MAC mode bit 0: second SHA block from TempKey.

13.2.2.276 MAC_MODE_SOURCE_FLAG_MATCH

```
#define MAC_MODE_SOURCE_FLAG_MATCH ((uint8_t)0x04)
```

MAC mode bit 2: match TempKey.SourceFlag.

13.2.2.277 MAC_RSP_SIZE

```
#define MAC_RSP_SIZE ATCA_RSP_SIZE_32
```

MAC command response packet size.

13.2.2.278 MAC_SIZE

```
#define MAC_SIZE (32)
```

MAC size of response.

13.2.2.279 NONCE_COUNT_LONG

```
#define NONCE_COUNT_LONG (ATCA_CMD_SIZE_MIN + 32)
```

Nonce command packet size for 32 bytes of NumIn.

13.2.2.280 NONCE_COUNT_LONG_64

```
#define NONCE_COUNT_LONG_64 (ATCA_CMD_SIZE_MIN + 64)
```

Nonce command packet size for 64 bytes of NumIn.

13.2.2.281 NONCE_COUNT_SHORT

```
#define NONCE_COUNT_SHORT (ATCA_CMD_SIZE_MIN + 20)
```

Nonce command packet size for 20 bytes of NumIn.

13.2.2.282 NONCE_INPUT_IDX

```
#define NONCE_INPUT_IDX ATCA_DATA_IDX
```

Nonce command index for input data.

13.2.2.283 NONCE_MODE_IDX

```
#define NONCE_MODE_IDX ATCA_PARAM1_IDX
```

Nonce command index for mode.

13.2.2.284 NONCE_MODE_INPUT_LEN_32

```
#define NONCE_MODE_INPUT_LEN_32 ((uint8_t)0x00)
```

Nonce mode: input size is 32 bytes.

13.2.2.285 NONCE_MODE_INPUT_LEN_64

```
#define NONCE_MODE_INPUT_LEN_64 ((uint8_t)0x20)
```

Nonce mode: input size is 64 bytes.

13.2.2.286 NONCE_MODE_INPUT_LEN_MASK

```
#define NONCE_MODE_INPUT_LEN_MASK ((uint8_t)0x20)
```

Nonce mode: input size mask.

13.2.2.287 NONCE_MODE_INVALID

```
#define NONCE_MODE_INVALID ((uint8_t)0x02)
```

Nonce mode 2 is invalid.

13.2.2.288 NONCE_MODE_MASK

```
#define NONCE_MODE_MASK ((uint8_t)0x03)
```

Nonce mode bits 2 to 7 are 0.

13.2.2.289 NONCE_MODE_NO_SEED_UPDATE

```
#define NONCE_MODE_NO_SEED_UPDATE ((uint8_t)0x01)
```

Nonce mode: do not update seed.

13.2.2.290 NONCE_MODE_PASSTHROUGH

```
#define NONCE_MODE_PASSTHROUGH ((uint8_t)0x03)
```

Nonce mode: pass-through.

13.2.2.291 NONCE_MODE_SEED_UPDATE

```
#define NONCE_MODE_SEED_UPDATE ((uint8_t)0x00)
```

Nonce mode: update seed.

13.2.2.292 NONCE_MODE_TARGET_ALTKEYBUF

```
#define NONCE_MODE_TARGET_ALTKEYBUF ((uint8_t)0x80)
```

Nonce mode: target is Alternate Key Buffer.

13.2.2.293 NONCE_MODE_TARGET_MASK

```
#define NONCE_MODE_TARGET_MASK ((uint8_t)0xC0)
```

Nonce mode: target mask.

13.2.2.294 NONCE_MODE_TARGET_MSGDIGBUF

```
#define NONCE_MODE_TARGET_MSGDIGBUF ((uint8_t)0x40)
```

Nonce mode: target is Message Digest Buffer.

13.2.2.295 NONCE_MODE_TARGET_TEMPKEY

```
#define NONCE_MODE_TARGET_TEMPKEY ((uint8_t)0x00)
```

Nonce mode: target is TempKey.

13.2.2.296 NONCE_NUMIN_SIZE

```
#define NONCE_NUMIN_SIZE (20)
```

Nonce NumIn size for random modes.

13.2.2.297 NONCE_NUMIN_SIZE_PASSTHROUGH

```
#define NONCE_NUMIN_SIZE_PASSTHROUGH (32)
```

Nonce NumIn size for 32-byte pass-through mode.

13.2.2.298 NONCE_PARAM2_IDX

```
#define NONCE_PARAM2_IDX ATCA_PARAM2_IDX
```

Nonce command index for 2. parameter.

13.2.2.299 NONCE_RSP_SIZE_LONG

```
#define NONCE_RSP_SIZE_LONG ATCA_RSP_SIZE_32
```

Nonce command response packet size with output.

13.2.2.300 NONCE_RSP_SIZE_SHORT

```
#define NONCE_RSP_SIZE_SHORT ATCA_RSP_SIZE_MIN
```

Nonce command response packet size with no output.

13.2.2.301 NONCE_ZERO_CALC_MASK

```
#define NONCE_ZERO_CALC_MASK ((uint16_t)0x8000)
```

Nonce zero (param2): calculation mode mask.

13.2.2.302 NONCE_ZERO_CALC_RANDOM

```
#define NONCE_ZERO_CALC_RANDOM ((uint16_t)0x0000)
```

Nonce zero (param2): calculation mode random, use RNG in calculation and return RNG output.

13.2.2.303 NONCE_ZERO_CALC_TEMPKEY

```
#define NONCE_ZERO_CALC_TEMPKEY ((uint16_t)0x8000)
```

Nonce zero (param2): calculation mode TempKey, use TempKey in calculation and return new TempKey value.

13.2.2.304 OUTNONCE_SIZE

```
#define OUTNONCE_SIZE (32)
```

Size of the OutNonce response expected from several commands.

13.2.2.305 PAUSE_COUNT

```
#define PAUSE_COUNT ATCA_CMD_SIZE_MIN
```

Pause command packet size.

13.2.2.306 PAUSE_PARAM2_IDX

```
#define PAUSE_PARAM2_IDX ATCA_PARAM2_IDX
```

Pause command index for 2. parameter.

13.2.2.307 PAUSE_RSP_SIZE

```
#define PAUSE_RSP_SIZE ATCA_RSP_SIZE_MIN
```

Pause command response packet size.

13.2.2.308 PAUSE_SELECT_IDX

```
#define PAUSE_SELECT_IDX ATCA_PARAM1_IDX
```

Pause command index for Selector.

13.2.2.309 PRIVWRITE_COUNT

```
#define PRIVWRITE_COUNT (75)
```

PrivWrite command packet size.

13.2.2.310 PRIVWRITE_KEYID_IDX

```
#define PRIVWRITE_KEYID_IDX ATCA_PARAM2_IDX
```

PrivWrite command index for KeyID.

13.2.2.311 PRIVWRITE_MAC_IDX

```
#define PRIVWRITE_MAC_IDX (41)
```

PrivWrite command index for MAC.

13.2.2.312 PRIVWRITE_MODE_ENCRYPT

```
#define PRIVWRITE_MODE_ENCRYPT ((uint8_t)0x40)
```

PrivWrite mode: encrypted.

13.2.2.313 PRIVWRITE_RSP_SIZE

```
#define PRIVWRITE_RSP_SIZE ATCA_RSP_SIZE_MIN
```

PrivWrite command response packet size.

13.2.2.314 PRIVWRITE_VALUE_IDX

```
#define PRIVWRITE_VALUE_IDX ( 5)
```

PrivWrite command index for value.

13.2.2.315 PRIVWRITE_ZONE_IDX

```
#define PRIVWRITE_ZONE_IDX ATCA_PARAM1_IDX
```

PrivWrite command index for zone.

13.2.2.316 PRIVWRITE_ZONE_MASK

```
#define PRIVWRITE_ZONE_MASK ((uint8_t)0x40)
```

PrivWrite zone bits 0 to 5 and 7 are 0.

13.2.2.317 RANDOM_COUNT

```
#define RANDOM_COUNT ATCA_CMD_SIZE_MIN
```

Random command packet size.

13.2.2.318 RANDOM_MODE_IDX

```
#define RANDOM_MODE_IDX ATCA_PARAM1_IDX
```

Random command index for mode.

13.2.2.319 RANDOM_NO_SEED_UPDATE

```
#define RANDOM_NO_SEED_UPDATE ((uint8_t)0x01)
```

Random mode for no seed update.

13.2.2.320 RANDOM_NUM_SIZE

```
#define RANDOM_NUM_SIZE ((uint8_t)32)
```

Number of bytes in the data packet of a random command.

13.2.2.321 RANDOM_PARAM2_IDX

```
#define RANDOM_PARAM2_IDX ATCA_PARAM2_IDX
```

Random command index for 2. parameter.

13.2.2.322 RANDOM_RSP_SIZE

```
#define RANDOM_RSP_SIZE ATCA_RSP_SIZE_32
```

Random command response packet size.

13.2.2.323 RANDOM_SEED_UPDATE

```
#define RANDOM_SEED_UPDATE ((uint8_t)0x00)
```

Random mode for automatic seed update.

13.2.2.324 READ_32_RSP_SIZE

```
#define READ_32_RSP_SIZE ATCA_RSP_SIZE_32
```

Read command response packet size when reading 32 bytes.

13.2.2.325 READ_4_RSP_SIZE

```
#define READ_4_RSP_SIZE ATCA_RSP_SIZE_VAL
```

Read command response packet size when reading 4 bytes.

13.2.2.326 READ_ADDR_IDX

```
#define READ_ADDR_IDX ATCA_PARAM2_IDX
```

Read command index for address.

13.2.2.327 READ_COUNT

```
#define READ_COUNT ATCA_CMD_SIZE_MIN
```

Read command packet size.

13.2.2.328 READ_ZONE_IDX

```
#define READ_ZONE_IDX ATCA_PARAM1_IDX
```

Read command index for zone.

13.2.2.329 READ_ZONE_MASK

```
#define READ_ZONE_MASK ((uint8_t)0x83)
```

Read zone bits 2 to 6 are 0.

13.2.2.330 RSA2048_KEY_SIZE

```
#define RSA2048_KEY_SIZE (256)
```

size of a RSA private key

13.2.2.331 SECUREBOOT_COUNT_DIG

```
#define SECUREBOOT_COUNT_DIG (ATCA_CMD_SIZE_MIN + SECUREBOOT_DIGEST_SIZE)
```

SecureBoot command packet size for just a digest.

13.2.2.332 SECUREBOOT_COUNT_DIG_SIG

```
#define SECUREBOOT_COUNT_DIG_SIG (ATCA_CMD_SIZE_MIN + SECUREBOOT_DIGEST_SIZE + SECUREBOOT_SIGNATURE_SIZE)
```

SecureBoot command packet size for a digest and signature.

13.2.2.333 SECUREBOOT_DIGEST_SIZE

```
#define SECUREBOOT_DIGEST_SIZE (32)
```

SecureBoot digest input size.

13.2.2.334 SECUREBOOT_MAC_SIZE

```
#define SECUREBOOT_MAC_SIZE (32)
```

SecureBoot MAC output size.

13.2.2.335 SECUREBOOT_MODE_ENC_MAC_FLAG

```
#define SECUREBOOT_MODE_ENC_MAC_FLAG ((uint8_t)0x80)
```

SecureBoot mode flag for encrypted digest and returning validating MAC.

13.2.2.336 SECUREBOOT_MODE_FULL

```
#define SECUREBOOT_MODE_FULL ((uint8_t)0x05)
```

SecureBoot mode Full.

13.2.2.337 SECUREBOOT_MODE_FULL_COPY

```
#define SECUREBOOT_MODE_FULL_COPY ((uint8_t)0x07)
```

SecureBoot mode FullCopy.

13.2.2.338 SECUREBOOT_MODE_FULL_STORE

```
#define SECUREBOOT_MODE_FULL_STORE ((uint8_t)0x06)
```

SecureBoot mode FullStore.

13.2.2.339 SECUREBOOT_MODE_IDX

```
#define SECUREBOOT_MODE_IDX ATCA_PARAM1_IDX
```

SecureBoot command index for mode.

13.2.2.340 SECUREBOOT_MODE_MASK

```
#define SECUREBOOT_MODE_MASK ((uint8_t)0x07)
```

SecureBoot mode mask.

13.2.2.341 SECUREBOOT_MODE_PROHIBIT_FLAG

```
#define SECUREBOOT_MODE_PROHIBIT_FLAG ((uint8_t)0x40)
```

SecureBoot mode flag to prohibit SecureBoot until next power cycle.

13.2.2.342 SECUREBOOT_RSP_SIZE_MAC

```
#define SECUREBOOT_RSP_SIZE_MAC (ATCA_PACKET_OVERHEAD + SECUREBOOT_MAC_SIZE)
```

SecureBoot response packet size with MAC.

13.2.2.343 SECUREBOOT_RSP_SIZE_NO_MAC

```
#define SECUREBOOT_RSP_SIZE_NO_MAC ATCA_RSP_SIZE_MIN
```

SecureBoot response packet size for no MAC.

13.2.2.344 SECUREBOOT_SIGNATURE_SIZE

```
#define SECUREBOOT_SIGNATURE_SIZE (64)
```

SecureBoot signature input size.

13.2.2.345 SECUREBOOTCONFIG_MODE_DISABLED

```
#define SECUREBOOTCONFIG_MODE_DISABLED ((uint16_t)0x0000)
```

Disabled SecureBootMode in SecureBootConfig value.

13.2.2.346 SECUREBOOTCONFIG_MODE_FULL_BOTH

```
#define SECUREBOOTCONFIG_MODE_FULL_BOTH ((uint16_t)0x0001)
```

Both digest and signature always required SecureBootMode in SecureBootConfig value.

13.2.2.347 SECUREBOOTCONFIG_MODE_FULL_DIG

```
#define SECUREBOOTCONFIG_MODE_FULL_DIG ((uint16_t)0x0003)
```

Digest stored SecureBootMode in SecureBootConfig value.

13.2.2.348 SECUREBOOTCONFIG_MODE_FULL_SIG

```
#define SECUREBOOTCONFIG_MODE_FULL_SIG ((uint16_t)0x0002)
```

Signature stored SecureBootMode in SecureBootConfig value.

13.2.2.349 SECUREBOOTCONFIG_MODE_MASK

```
#define SECUREBOOTCONFIG_MODE_MASK ((uint16_t)0x0003)
```

Mask for SecureBootMode field in SecureBootConfig value.

13.2.2.350 SECUREBOOTCONFIG_OFFSET

```
#define SECUREBOOTCONFIG_OFFSET (70)
```

SecureBootConfig byte offset into the configuration zone.

13.2.2.351 SELFTEST_COUNT

```
#define SELFTEST_COUNT ATCA_CMD_SIZE_MIN
```

SelfTest command packet size.

13.2.2.352 SELFTEST_MODE_AES

```
#define SELFTEST_MODE_AES ((uint8_t)0x10)
```

SelfTest mode AES encrypt function.

13.2.2.353 SELFTEST_MODE_ALL

```
#define SELFTEST_MODE_ALL ((uint8_t)0x3B)
```

SelfTest mode all algorithms.

13.2.2.354 SELFTEST_MODE_ECDH

```
#define SELFTEST_MODE_ECDH ((uint8_t)0x08)
```

SelfTest mode ECDH function.

13.2.2.355 SELFTEST_MODE_ECDSA_SIGN_VERIFY

```
#define SELFTEST_MODE_ECDSA_SIGN_VERIFY ((uint8_t)0x02)
```

SelfTest mode ECDSA verify function.

13.2.2.356 SELFTEST_MODE_IDX

```
#define SELFTEST_MODE_IDX ATCA_PARAM1_IDX
```

SelfTest command index for mode.

13.2.2.357 SELFTEST_MODE_RNG

```
#define SELFTEST_MODE_RNG ((uint8_t)0x01)
```

SelfTest mode RNG DRBG function.

13.2.2.358 SELFTEST_MODE_SHA

```
#define SELFTEST_MODE_SHA ((uint8_t)0x20)
```

SelfTest mode SHA function.

13.2.2.359 SELFTEST_RSP_SIZE

```
#define SELFTEST_RSP_SIZE ATCA_RSP_SIZE_MIN
```

SelfTest command response packet size.

13.2.2.360 SHA_CONTEXT_MAX_SIZE

```
#define SHA_CONTEXT_MAX_SIZE (99)
```

13.2.2.361 SHA_COUNT_LONG

```
#define SHA_COUNT_LONG ATCA_CMD_SIZE_MIN
```

Just a starting size.

13.2.2.362 SHA_COUNT_SHORT

```
#define SHA_COUNT_SHORT ATCA_CMD_SIZE_MIN
```

13.2.2.363 SHA_DATA_MAX

```
#define SHA_DATA_MAX (64)
```

13.2.2.364 SHA_MODE_608_HMAC_END

```
#define SHA_MODE_608_HMAC_END ((uint8_t)0x02)
```

Complete the HMAC computation and return digest... Different command on 608.

13.2.2.365 SHA_MODE_HMAC_END

```
#define SHA_MODE_HMAC_END ((uint8_t)0x05)
```

Complete the HMAC computation and return digest.

13.2.2.366 SHA_MODE_HMAC_START

```
#define SHA_MODE_HMAC_START ((uint8_t)0x04)
```

Initialization, HMAC calculation.

13.2.2.367 SHA_MODE_HMAC_UPDATE

```
#define SHA_MODE_HMAC_UPDATE ((uint8_t)0x01)
```

Add 64 bytes in the meesage to the SHA context.

13.2.2.368 SHA_MODE_MASK

```
#define SHA_MODE_MASK ((uint8_t)0x07)
```

Mask the bit 0-2.

13.2.2.369 SHA_MODE_READ_CONTEXT

```
#define SHA_MODE_READ_CONTEXT ((uint8_t)0x06)
```

Read current SHA-256 context out of the device.

13.2.2.370 SHA_MODE_SHA256_END

```
#define SHA_MODE_SHA256_END ((uint8_t)0x02)
```

Complete the calculation and return the digest.

13.2.2.371 SHA_MODE_SHA256_PUBLIC

```
#define SHA_MODE_SHA256_PUBLIC ((uint8_t)0x03)
```

Add 64 byte ECC public key in the slot to the SHA context.

13.2.2.372 SHA_MODE_SHA256_START

```
#define SHA_MODE_SHA256_START ((uint8_t)0x00)
```

Initialization, does not accept a message.

13.2.2.373 SHA_MODE_SHA256_UPDATE

```
#define SHA_MODE_SHA256_UPDATE ((uint8_t)0x01)
```

Add 64 bytes in the meesage to the SHA context.

13.2.2.374 SHA_MODE_TARGET_MASK

```
#define SHA_MODE_TARGET_MASK ((uint8_t)0xC0)
```

Resulting digest target location mask.

13.2.2.375 SHA_MODE_TARGET_MSGDIGBUF

```
#define SHA_MODE_TARGET_MSGDIGBUF ((uint8_t)0x40)
```

Place resulting digest both in Output buffer and Message Digest Buffer.

13.2.2.376 SHA_MODE_TARGET_OUT_ONLY

```
#define SHA_MODE_TARGET_OUT_ONLY ((uint8_t)0xC0)
```

Place resulting digest both in Output buffer ONLY.

13.2.2.377 SHA_MODE_TARGET_TEMPKEY

```
#define SHA_MODE_TARGET_TEMPKEY ((uint8_t)0x00)
```

Place resulting digest both in Output buffer and TempKey.

13.2.2.378 SHA_MODE_WRITE_CONTEXT

```
#define SHA_MODE_WRITE_CONTEXT ((uint8_t)0x07)
```

Restore a SHA-256 context into the device.

13.2.2.379 SHA_RSP_SIZE

```
#define SHA_RSP_SIZE ATCA_RSP_SIZE_32
```

SHA command response packet size.

13.2.2.380 SHA_RSP_SIZE_LONG

```
#define SHA_RSP_SIZE_LONG ATCA_RSP_SIZE_32
```

SHA command response packet size.

13.2.2.381 SHA_RSP_SIZE_SHORT

```
#define SHA_RSP_SIZE_SHORT ATCA_RSP_SIZE_MIN
```

SHA command response packet size only status code.

13.2.3 Typedef Documentation

13.2.3.1 ATCACommand

```
typedef struct atca_command* ATCACommand
```

13.2.4 Function Documentation

13.2.4.1 atAES()

```
ATCA_STATUS atAES (
            ATCACommand ca_cmd,
            ATCAPacket * packet )
```

ATCACommand AES method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

13.2.4.2 atCalcCrc()

```
void atCalcCrc (
            ATCAPacket * packet )
```

This function calculates CRC and adds it to the correct offset in the packet data.

Parameters

in	*packet*	Packet to calculate CRC data for

13.2.4.3 atCheckCrc()

```
ATCA_STATUS atCheckCrc (
            const uint8_t * response )
```

This function checks the consistency of a response.

Parameters

in	*response*	pointer to response

Returns

ATCA_SUCCESS on success, otherwise ATCA_RX_CRC_ERROR

13.2.4.4 atCheckMAC()

```
ATCA_STATUS atCheckMAC (
            ATCACommand ca_cmd,
            ATCAPacket * packet )
```

ATCACommand CheckMAC method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

13.2.4.5 atCounter()

```
ATCA_STATUS atCounter (
            ATCACommand ca_cmd,
            ATCAPacket * packet )
```

ATCACommand Counter method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.2.4.6 atCRC()

```
void atCRC (
            size_t length,
            const uint8_t * data,
            uint8_t * crc_le )
```

Calculates CRC over the given raw data and returns the CRC in little-endian byte order.

Parameters

in	*length*	Size of data not including the CRC byte positions
in	*data*	Pointer to the data over which to compute the CRC
out	*crc←_le*	Pointer to the place where the two-bytes of CRC will be returned in little-endian byte order.

13.2.4.7 atDeriveKey()

```
ATCA_STATUS atDeriveKey (
            ATCACommand ca_cmd,
            ATCAPacket * packet,
            bool has_mac )
```

ATCACommand DeriveKey method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built
in	*has_mac*	hasMAC determines if MAC data is present in the packet input

Returns

ATCA_SUCCESS

13.2.4.8 atECDH()

```
ATCA_STATUS atECDH (
            ATCACommand ca_cmd,
            ATCAPacket * packet )
```

ATCACommand ECDH method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

13.2.4.9 atGenDig()

```
ATCA_STATUS atGenDig (
            ATCACommand ca_cmd,
            ATCAPacket * packet,
            bool is_no_mac_key )
```

ATCACommand Generate Digest method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built
in	*is_no_mac_key*	Should be true if GenDig is being run on a slot that has its SlotConfig.NoMac bit set

Returns

ATCA_SUCCESS

13.2.4.10 atGenKey()

```
ATCA_STATUS atGenKey (
            ATCACommand ca_cmd,
            ATCAPacket * packet )
```

ATCACommand Generate Key method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

13.2.4.11 atHMAC()

```
ATCA_STATUS atHMAC (
            ATCACommand ca_cmd,
            ATCAPacket * packet )
```

ATCACommand HMAC method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

13.2.4.12 atInfo()

```
ATCA_STATUS atInfo (
            ATCACommand ca_cmd,
            ATCAPacket * packet )
```

ATCACommand Info method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

13.2.4.13 atIsECCFamily()

```
bool atIsECCFamily (
            ATCADeviceType device_type )
```

determines if a given device type is an ECC device or a superset of a ECC device

Parameters

in	*device_type*	Type of device to check for family type

Returns

boolean indicating whether the given device is an ECC family device.

13.2.4.14 atIsSHAFamily()

```
bool atIsSHAFamily (
            ATCADeviceType device_type )
```

determines if a given device type is a SHA device or a superset of a SHA device

Parameters

in	*device_type*	Type of device to check for family type

Returns

boolean indicating whether the given device is a SHA family device.

13.2.4.15 atKDF()

```
ATCA_STATUS atKDF (
            ATCACommand ca_cmd,
            ATCAPacket * packet )
```

ATCACommand KDF method.

Parameters

in	*ca_cmd*	Instance
in	*packet*	Pointer to the packet containing the command being built.

Returns

ATCA_SUCCESS

13.2.4.16 atLock()

```
ATCA_STATUS atLock (
            ATCACommand ca_cmd,
            ATCAPacket * packet )
```

ATCACommand Lock method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

13.2.4.17 atMAC()

```
ATCA_STATUS atMAC (
            ATCACommand ca_cmd,
            ATCAPacket * packet )
```

ATCACommand MAC method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

13.2.4.18 atNonce()

```
ATCA_STATUS atNonce (
            ATCACommand ca_cmd,
            ATCAPacket * packet )
```

ATCACommand Nonce method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.2.4.19 atPause()

```
ATCA_STATUS atPause (
            ATCACommand ca_cmd,
            ATCAPacket * packet )
```

ATCACommand Pause method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

13.2.4.20 atPrivWrite()

```
ATCA_STATUS atPrivWrite (
            ATCACommand ca_cmd,
            ATCAPacket * packet )
```

ATCACommand PrivWrite method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

13.2.4.21 atRandom()

```
ATCA_STATUS atRandom (
            ATCACommand ca_cmd,
            ATCAPacket * packet )
```

ATCACommand Random method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

13.2.4.22 atRead()

```
ATCA_STATUS atRead (
            ATCACommand ca_cmd,
            ATCAPacket * packet )
```

ATCACommand Read method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

13.2.4.23 atSecureBoot()

```
ATCA_STATUS atSecureBoot (
            ATCACommand ca_cmd,
            ATCAPacket * packet )
```

ATCACommand SecureBoot method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

13.2.4.24 atSelfTest()

```
ATCA_STATUS atSelfTest (
            ATCACommand ca_cmd,
            ATCAPacket * packet )
```

ATCACommand AES method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

13.2.4.25 atSHA()

```
ATCA_STATUS atSHA (
            ATCACommand ca_cmd,
            ATCAPacket * packet,
            uint16_t write_context_size )
```

ATCACommand SHA method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built
in	*write_context_size*	the length of the sha write_context data

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.2.4.26 atSign()

```
ATCA_STATUS atSign (
            ATCACommand ca_cmd,
            ATCAPacket * packet )
```

ATCACommand Sign method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

13.2.4.27 atUpdateExtra()

```
ATCA_STATUS atUpdateExtra (
            ATCACommand ca_cmd,
            ATCAPacket * packet )
```

ATCACommand UpdateExtra method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS

13.2.4.28 atVerify()

```
ATCA_STATUS atVerify (
            ATCACommand ca_cmd,
            ATCAPacket * packet )
```

ATCACommand ECDSA Verify method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.2.4.29 atWrite()

```
ATCA_STATUS atWrite (
            ATCACommand ca_cmd,
            ATCAPacket * packet,
            bool has_mac )
```

ATCACommand Write method.

Parameters

in	*ca_cmd*	instance
in	*packet*	pointer to the packet containing the command being built
in	*has_mac*	Flag to indicate whether a mac is present or not

Returns

ATCA_SUCCESS

13.2.4.30 deleteATCACommand()

```
void deleteATCACommand (
            ATCACommand * ca_cmd )
```

ATCACommand destructor.

Parameters

in	*ca_cmd*	instance of a command object

13.2.4.31 initATCACommand()

```
ATCA_STATUS initATCACommand (
```

```
            ATCADeviceType device_type,
            ATCACommand ca_cmd )
```

Initializer for ATCACommand.

Parameters

in	*device_type*	Specifies which set of commands and execution times should be associated with this command object.
in	*ca_cmd*	Pre-allocated command structure to initialize.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.2.4.32 isATCAError()

```
ATCA_STATUS isATCAError (
            uint8_t * data )
```

checks for basic error frame in data

Parameters

in	*data*	pointer to received data - expected to be in the form of a CA device response frame

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.2.4.33 newATCACommand()

```
ATCACommand newATCACommand (
            ATCADeviceType device_type )
```

constructor for ATCACommand

Parameters

in	*device_type*	Specifies which set of commands and execution times should be associated with this command object.

Returns

Initialized object on success. NULL on failure.

13.3 ATCADevice (atca_)

ATCADevice object - composite of command and interface objects.

Data Structures

- struct atca_device

    atca_device is the C object backing ATCADevice. See the atca_device.h file for details on the ATCADevice methods.

Typedefs

- typedef struct atca_device ∗ ATCADevice

Enumerations

- enum ATCADeviceType {
  ATSHA204A, ATECC108A, ATECC508A, ATECC608A,
  ATCA_DEV_UNKNOWN = 0x20 }

    The supported Device type in Cryptoauthlib library.

Functions

- ATCADevice newATCADevice (ATCAIfaceCfg ∗cfg)

    constructor for a Microchip CryptoAuth device
- void deleteATCADevice (ATCADevice ∗ca_dev)

    destructor for a device NULLs reference after object is freed
- ATCA_STATUS initATCADevice (ATCAIfaceCfg ∗cfg, ATCADevice ca_dev)

    Initializer for an Microchip CryptoAuth device.
- ATCACommand atGetCommands (ATCADevice dev)

    returns a reference to the ATCACommand object for the device
- ATCAIface atGetIFace (ATCADevice dev)

    returns a reference to the ATCAIface interface object for the device
- ATCA_STATUS releaseATCADevice (ATCADevice ca_dev)

    Release any resources associated with the device.

13.3.1 Detailed Description

ATCADevice object - composite of command and interface objects.

13.3.2 Typedef Documentation

13.3.2.1 ATCADevice

```
typedef struct atca_device* ATCADevice
```

13.3.3 Enumeration Type Documentation

13.3.3.1 ATCADeviceType

```
enum ATCADeviceType
```

The supported Device type in Cryptoauthlib library.

Enumerator

ATSHA204A	
ATECC108A	
ATECC508A	
ATECC608A	
ATCA_DEV_UNKNOWN	

13.3.4 Function Documentation

13.3.4.1 atGetCommands()

```
ATCACommand atGetCommands (
            ATCADevice dev )
```

returns a reference to the ATCACommand object for the device

Parameters

in	*dev*	reference to a device

Returns

reference to the ATCACommand object for the device

13.3.4.2 atGetIFace()

```
ATCAIface atGetIFace (
            ATCADevice dev )
```

returns a reference to the ATCAIface interface object for the device

Parameters

in	*dev*	reference to a device

Returns

reference to the ATCAIface object for the device

13.3.4.3 deleteATCADevice()

```
void deleteATCADevice (
            ATCADevice * ca_dev )
```

destructor for a device NULLs reference after object is freed

Parameters

in	*ca_dev*	pointer to a reference to a device

13.3.4.4 initATCADevice()

```
ATCA_STATUS initATCADevice (
            ATCAIfaceCfg * cfg,
            ATCADevice ca_dev )
```

Initializer for an Microchip CryptoAuth device.

Parameters

in	*cfg*	pointer to an interface configuration object
in,out	*ca_dev*	As input, pre-allocated structure to be initialized. mCommands and mIface members should point to existing structures to be initialized.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.3.4.5 newATCADevice()

```
ATCADevice newATCADevice (
            ATCAIfaceCfg * cfg )
```

constructor for a Microchip CryptoAuth device

Parameters

in	*cfg*	Interface configuration object

Returns

Reference to a new ATCADevice on success. NULL on failure.

13.3.4.6 releaseATCADevice()

```
ATCA_STATUS releaseATCADevice (
            ATCADevice ca_dev )
```

Release any resources associated with the device.

Parameters

in	*ca_dev*	Device to release

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.4 ATCAIface (atca_)

Abstract interface to all CryptoAuth device types. This interface connects to the HAL implementation and abstracts the physical details of the device communication from all the upper layers of CryptoAuthLib.

Data Structures

- struct ATCAIfaceCfg
- struct atca_iface

  atca_iface is the C object backing ATCAIface. See the atca_iface.h file for details on the ATCAIface methods

Macros

- #define ATCA_POST_DELAY_MSEC 25

Typedefs

- typedef struct atca_iface ∗ ATCAIface

Enumerations

- enum ATCAIfaceType {
  ATCA_I2C_IFACE, ATCA_SWI_IFACE, ATCA_UART_IFACE, ATCA_SPI_IFACE,
  ATCA_HID_IFACE, ATCA_CUSTOM_IFACE, ATCA_UNKNOWN_IFACE }
- enum ATCAKitType { ATCA_KIT_AUTO_IFACE, ATCA_KIT_I2C_IFACE, ATCA_KIT_SWI_IFACE, ATCA_KIT_UNKNOWN_IFACE }

Functions

- ATCA_STATUS _atinit (ATCAIface ca_iface, ATCAHAL_t ∗hal)
- ATCA_STATUS initATCAIface (ATCAIfaceCfg ∗cfg, ATCAIface ca_iface)

  Initializer for ATCAIface objects.
- ATCAIface newATCAIface (ATCAIfaceCfg ∗cfg)

  Constructor for ATCAIface objects.
- ATCA_STATUS atinit (ATCAIface ca_iface)

  Performs the HAL initialization by calling intermediate HAL wrapper function. If using the basic API, the atcab_init() function should be called instead.
- ATCA_STATUS atsend (ATCAIface ca_iface, uint8_t ∗txdata, int txlength)

  Sends the data to the device by calling intermediate HAL wrapper function.
- ATCA_STATUS atreceive (ATCAIface ca_iface, uint8_t ∗rxdata, uint16_t ∗rxlength)

  Receives data from the device by calling intermediate HAL wrapper function.
- ATCA_STATUS atwake (ATCAIface ca_iface)

  Wakes up the device by calling intermediate HAL wrapper function. If using the basic API, the atcab_wakeup() function should be used instead.
- ATCA_STATUS atidle (ATCAIface ca_iface)

  Puts the device into idle state by calling intermediate HAL wrapper function. If using the basic API, the atcab_idle() function should be used instead.
- ATCA_STATUS atsleep (ATCAIface ca_iface)

Puts the device into sleep state by calling intermediate HAL wrapper function. If using the basic API, the atcab_sleep() function should be used instead.

- ATCAIfaceCfg * atgetifacecfg (ATCAIface ca_iface)

  Returns the logical interface configuration for the device.
- void * atgetifacehaldat (ATCAIface ca_iface)

  Returns the HAL data pointer for the device.
- ATCA_STATUS releaseATCAIface (ATCAIface ca_iface)

  Instruct the HAL driver to release any resources associated with this interface.
- void deleteATCAIface (ATCAIface *ca_iface)

  Instruct the HAL driver to release any resources associated with this interface, then delete the object.
- ATCA_STATUS atpostinit (ATCAIface ca_iface)

13.4.1 Detailed Description

Abstract interface to all CryptoAuth device types. This interface connects to the HAL implementation and abstracts the physical details of the device communication from all the upper layers of CryptoAuthLib.

13.4.2 Macro Definition Documentation

13.4.2.1 ATCA_POST_DELAY_MSEC

```
#define ATCA_POST_DELAY_MSEC 25
```

13.4.3 Typedef Documentation

13.4.3.1 ATCAIface

```
typedef struct atca_iface* ATCAIface
```

13.4.4 Enumeration Type Documentation

13.4.4.1 ATCAIfaceType

```
enum ATCAIfaceType
```

Enumerator

ATCA_I2C_IFACE	
ATCA_SWI_IFACE	
ATCA_UART_IFACE	
ATCA_SPI_IFACE	
ATCA_HID_IFACE	
ATCA_CUSTOM_IFACE	
ATCA_UNKNOWN_IFACE	

13.4.4.2 ATCAKitType

```
enum ATCAKitType
```

Enumerator

ATCA_KIT_AUTO_IFACE	
ATCA_KIT_I2C_IFACE	
ATCA_KIT_SWI_IFACE	
ATCA_KIT_UNKNOWN_IFACE	

13.4.5 Function Documentation

13.4.5.1 _atinit()

```
ATCA_STATUS _atinit (
             ATCAIface ca_iface,
             ATCAHAL_t * hal )
```

13.4.5.2 atgetifacecfg()

```
ATCAIfaceCfg * atgetifacecfg (
             ATCAIface ca_iface )
```

Returns the logical interface configuration for the device.

Parameters

in	*ca_iface*	Device interface.

Returns

Logical interface configuration.

13.4.5.3 atgetifacehaldat()

```
void * atgetifacehaldat (
            ATCAIface ca_iface )
```

Returns the HAL data pointer for the device.

Parameters

in	*ca_iface*	Device interface.

Returns

HAL data pointer.

13.4.5.4 atidle()

```
ATCA_STATUS atidle (
            ATCAIface ca_iface )
```

Puts the device into idle state by calling intermediate HAL wrapper function. If using the basic API, the atcab_idle() function should be used instead.

Parameters

in	*ca_iface*	Device to interact with.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.4.5.5 atinit()

```
ATCA_STATUS atinit (
            ATCAIface ca_iface )
```

Performs the HAL initialization by calling intermediate HAL wrapper function. If using the basic API, the atcab_init() function should be called instead.

Parameters

in	*ca_iface*	Device to interact with.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.4.5.6 atpostinit()

```
ATCA_STATUS atpostinit (
            ATCAIface ca_iface )
```

13.4.5.7 atreceive()

```
ATCA_STATUS atreceive (
            ATCAIface ca_iface,
            uint8_t * rxdata,
            uint16_t * rxlength )
```

Receives data from the device by calling intermediate HAL wrapper function.

Parameters

in	*ca_iface*	Device to interact with.
out	*rxdata*	Data received will be returned here.
in,out	*rxlength*	As input, the size of the rxdata buffer. As output, the number of bytes received.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.4.5.8 atsend()

```
ATCA_STATUS atsend (
            ATCAIface ca_iface,
            uint8_t * txdata,
            int txlength )
```

Sends the data to the device by calling intermediate HAL wrapper function.

Parameters

in	*ca_iface*	Device to interact with.
in	*txdata*	Data to be transmitted to the device.
in	*txlength*	Number of bytes to be transmitted to the device.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.4.5.9 atsleep()

```
ATCA_STATUS atsleep (
            ATCAIface ca_iface )
```

Puts the device into sleep state by calling intermediate HAL wrapper function. If using the basic API, the atcab_sleep() function should be used instead.

Parameters

in	*ca_iface*	Device to interact with.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.4.5.10 atwake()

```
ATCA_STATUS atwake (
            ATCAIface ca_iface )
```

Wakes up the device by calling intermediate HAL wrapper function. If using the basic API, the atcab_wakeup() function should be used instead.

Parameters

in	*ca_iface*	Device to interact with.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.4.5.11 deleteATCAIface()

```
void deleteATCAIface (
            ATCAIface * ca_iface )
```

Instruct the HAL driver to release any resources associated with this interface, then delete the object.

Parameters

in	*ca_iface*	Device interface.

13.4.5.12 initATCAIface()

```
ATCA_STATUS initATCAIface (
            ATCAIfaceCfg * cfg,
            ATCAIface ca_iface )
```

Initializer for ATCAIface objects.

Parameters

in	*cfg*	Logical configuration for the interface
in	*ca_iface*	Interface structure to initialize.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.4.5.13 newATCAIface()

```
ATCAIface newATCAIface (
            ATCAIfaceCfg * cfg )
```

Constructor for ATCAIface objects.

Parameters

in	*cfg*	Logical configuration for the interface

Returns

New interface instance on success. NULL on failure.

13.4.5.14 releaseATCAIface()

```
ATCA_STATUS releaseATCAIface (
            ATCAIface ca_iface )
```

Instruct the HAL driver to release any resources associated with this interface.

Parameters

in	*ca_iface*	Device interface.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.5 Certificate manipulation methods (atcacert_)

These methods provide convenient ways to perform certification I/O with CryptoAuth chips and perform certificate manipulation in memory.

Data Structures

- struct atcacert_tm_utc_s
- struct atcacert_device_loc_s
- struct atcacert_cert_loc_s
- struct atcacert_cert_element_s
- struct atcacert_def_s
- struct atcacert_build_state_s

Macros

- #define FALSE (0)
- #define TRUE (1)
- #define ATCACERT_E_SUCCESS 0

  Operation completed successfully.
- #define ATCACERT_E_ERROR 1

  General error.
- #define ATCACERT_E_BAD_PARAMS 2

  Invalid/bad parameter passed to function.
- #define ATCACERT_E_BUFFER_TOO_SMALL 3

  Supplied buffer for output is too small to hold the result.
- #define ATCACERT_E_DECODING_ERROR 4

  Data being decoded/parsed has an invalid format.
- #define ATCACERT_E_INVALID_DATE 5

  Date is invalid.
- #define ATCACERT_E_UNIMPLEMENTED 6

  Function is unimplemented for the current configuration.
- #define ATCACERT_E_UNEXPECTED_ELEM_SIZE 7

  A certificate element size was not what was expected.
- #define ATCACERT_E_ELEM_MISSING 8

  The certificate element isn't defined for the certificate definition.
- #define ATCACERT_E_ELEM_OUT_OF_BOUNDS 9

  Certificate element is out of bounds for the given certificate.
- #define ATCACERT_E_BAD_CERT 10

  Certificate structure is bad in some way.
- #define ATCACERT_E_WRONG_CERT_DEF 11
- #define ATCACERT_E_VERIFY_FAILED 12

  Certificate or challenge/response verification failed.
- #define ATCACERT_E_INVALID_TRANSFORM 13

  Invalid transform passed to function.
- #define DATEFMT_ISO8601_SEP_SIZE (20)
- #define DATEFMT_RFC5280_UTC_SIZE (13)
- #define DATEFMT_POSIX_UINT32_BE_SIZE (4)
- #define DATEFMT_POSIX_UINT32_LE_SIZE (4)
- #define DATEFMT_RFC5280_GEN_SIZE (15)
- #define DATEFMT_MAX_SIZE DATEFMT_ISO8601_SEP_SIZE
- #define ATCACERT_DATE_FORMAT_SIZES_COUNT 5

Typedefs

- typedef struct atcacert_tm_utc_s atcacert_tm_utc_t
- typedef enum atcacert_date_format_e atcacert_date_format_t
- typedef enum atcacert_cert_type_e atcacert_cert_type_t
- typedef enum atcacert_cert_sn_src_e atcacert_cert_sn_src_t
- typedef enum atcacert_device_zone_e atcacert_device_zone_t
- typedef enum atcacert_transform_e atcacert_transform_t

  How to transform the data from the device to the certificate.
- typedef enum atcacert_std_cert_element_e atcacert_std_cert_element_t
- typedef struct atcacert_device_loc_s atcacert_device_loc_t
- typedef struct atcacert_cert_loc_s atcacert_cert_loc_t
- typedef struct atcacert_cert_element_s atcacert_cert_element_t
- typedef struct atcacert_def_s atcacert_def_t
- typedef struct atcacert_build_state_s atcacert_build_state_t

Enumerations

- enum atcacert_date_format_e {
  DATEFMT_ISO8601_SEP, DATEFMT_RFC5280_UTC, DATEFMT_POSIX_UINT32_BE, DATEFMT_POSIX_UINT32_LE,
  DATEFMT_RFC5280_GEN }
- enum atcacert_cert_type_e { CERTTYPE_X509, CERTTYPE_CUSTOM }
- enum atcacert_cert_sn_src_e {
  SNSRC_STORED = 0x0, SNSRC_STORED_DYNAMIC = 0x7, SNSRC_DEVICE_SN = 0x8, SNSRC_SIGNER_ID = 0x9,
  SNSRC_PUB_KEY_HASH = 0xA, SNSRC_DEVICE_SN_HASH = 0xB, SNSRC_PUB_KEY_HASH_POS = 0xC, SNSRC_DEVICE_SN_HASH_POS = 0xD,
  SNSRC_PUB_KEY_HASH_RAW = 0xE, SNSRC_DEVICE_SN_HASH_RAW = 0xF }
- enum atcacert_device_zone_e { DEVZONE_CONFIG = 0x00, DEVZONE_OTP = 0x01, DEVZONE_DATA = 0x02, DEVZONE_NONE = 0x07 }
- enum atcacert_transform_e {
  TF_NONE, TF_REVERSE, TF_BIN2HEX_UC, TF_BIN2HEX_LC,
  TF_HEX2BIN_UC, TF_HEX2BIN_LC, TF_BIN2HEX_SPACE_UC, TF_BIN2HEX_SPACE_LC,
  TF_HEX2BIN_SPACE_UC, TF_HEX2BIN_SPACE_LC }

  How to transform the data from the device to the certificate.
- enum atcacert_std_cert_element_e {
  STDCERT_PUBLIC_KEY, STDCERT_SIGNATURE, STDCERT_ISSUE_DATE, STDCERT_EXPIRE_DATE,
  STDCERT_SIGNER_ID, STDCERT_CERT_SN, STDCERT_AUTH_KEY_ID, STDCERT_SUBJ_KEY_ID,
  STDCERT_NUM_ELEMENTS }

Functions

- int atcacert_read_device_loc (const atcacert_device_loc_t ∗device_loc, uint8_t ∗data)

  Read the data from a device location.
- int atcacert_read_cert (const atcacert_def_t ∗cert_def, const uint8_t ca_public_key[64], uint8_t ∗cert, size_t ∗cert_size)

  Reads the certificate specified by the certificate definition from the ATECC508A device.
- int atcacert_write_cert (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size)

  Take a full certificate and write it to the ATECC508A device according to the certificate definition.
- int atcacert_create_csr (const atcacert_def_t ∗csr_def, uint8_t ∗csr, size_t ∗csr_size)

  Creates a CSR specified by the CSR definition from the ATECC508A device. This process involves reading the dynamic CSR data from the device and combining it with the template found in the CSR definition, then signing it. Return the CSR int der format.

- int atcacert_create_csr_pem (const atcacert_def_t ∗csr_def, char ∗csr, size_t ∗csr_size)

  Creates a CSR specified by the CSR definition from the ATECC508A device. This process involves reading the dynamic CSR data from the device and combining it with the template found in the CSR definition, then signing it. Return the CSR int der format.
- int atcacert_get_response (uint8_t device_private_key_slot, const uint8_t challenge[32], uint8_t response[64])

  Calculates the response to a challenge sent from the host.
- int atcacert_date_enc (atcacert_date_format_t format, const atcacert_tm_utc_t ∗timestamp, uint8_t ∗formatted_date, size_t ∗formatted_date_size)

  Format a timestamp according to the format type.
- int atcacert_date_dec (atcacert_date_format_t format, const uint8_t ∗formatted_date, size_t formatted_date_size, atcacert_tm_utc_t ∗timestamp)

  Parse a formatted timestamp according to the specified format.
- int atcacert_date_enc_compcert (const atcacert_tm_utc_t ∗issue_date, uint8_t expire_years, uint8_t enc_dates[3])

  Encode the issue and expire dates in the format used by the compressed certificate.
- int atcacert_date_dec_compcert (const uint8_t enc_dates[3], atcacert_date_format_t expire_date_format, atcacert_tm_utc_t ∗issue_date, atcacert_tm_utc_t ∗expire_date)

  Decode the issue and expire dates from the format used by the compressed certificate.
- int atcacert_date_get_max_date (atcacert_date_format_t format, atcacert_tm_utc_t ∗timestamp)

  Return the maximum date available for the given format.
- int atcacert_date_enc_iso8601_sep (const atcacert_tm_utc_t ∗timestamp, uint8_t formatted_date[DATEFMT_ISO8601_SEP_S
- int atcacert_date_dec_iso8601_sep (const uint8_t formatted_date[DATEFMT_ISO8601_SEP_SIZE], atcacert_tm_utc_t ∗timestamp)
- int atcacert_date_enc_rfc5280_utc (const atcacert_tm_utc_t ∗timestamp, uint8_t formatted_date[DATEFMT_RFC5280_UTC_S
- int atcacert_date_dec_rfc5280_utc (const uint8_t formatted_date[DATEFMT_RFC5280_UTC_SIZE], atcacert_tm_utc_t ∗timestamp)
- int atcacert_date_enc_rfc5280_gen (const atcacert_tm_utc_t ∗timestamp, uint8_t formatted_date[DATEFMT_RFC5280_GEN_
- int atcacert_date_dec_rfc5280_gen (const uint8_t formatted_date[DATEFMT_RFC5280_GEN_SIZE], atcacert_tm_utc_t ∗timestamp)
- int atcacert_date_enc_posix_uint32_be (const atcacert_tm_utc_t ∗timestamp, uint8_t formatted_date[DATEFMT_POSIX_UINT32_BE_SIZE])
- int atcacert_date_dec_posix_uint32_be (const uint8_t formatted_date[DATEFMT_POSIX_UINT32_BE_SIZE], atcacert_tm_utc_t ∗timestamp)
- int atcacert_date_enc_posix_uint32_le (const atcacert_tm_utc_t ∗timestamp, uint8_t formatted_date[DATEFMT_POSIX_UINT32_LE_SIZE])
- int atcacert_date_dec_posix_uint32_le (const uint8_t formatted_date[DATEFMT_POSIX_UINT32_LE_SIZE], atcacert_tm_utc_t ∗timestamp)
- int atcacert_get_device_locs (const atcacert_def_t ∗cert_def, atcacert_device_loc_t ∗device_locs, size_t ∗device_locs_count, size_t device_locs_max_count, size_t block_size)

  Add all the device locations required to rebuild the specified certificate (cert_def) to a device locations list.
- int atcacert_cert_build_start (atcacert_build_state_t ∗build_state, const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t ∗cert_size, const uint8_t ca_public_key[64])

  Starts the certificate rebuilding process.
- int atcacert_cert_build_process (atcacert_build_state_t ∗build_state, const atcacert_device_loc_t ∗device_loc, const uint8_t ∗device_data)

  Process information read from the ATECC device. If it contains information for the certificate, it will be incorporated into the certificate.
- int atcacert_cert_build_finish (atcacert_build_state_t ∗build_state)

  Completes any final certificate processing required after all data from the device has been incorporated.
- int atcacert_get_device_data (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, const atcacert_device_loc_t ∗device_loc, uint8_t ∗device_data)

  Gets the dynamic data that would be saved to the specified device location. This function is primarily used to break down a full certificate into the dynamic components to be saved to a device.

- int atcacert_set_subj_public_key (const atcacert_def_t *cert_def, uint8_t *cert, size_t cert_size, const uint8_t subj_public_key[64])

    Sets the subject public key and subject key ID in a certificate.
- int atcacert_get_subj_public_key (const atcacert_def_t *cert_def, const uint8_t *cert, size_t cert_size, uint8_t subj_public_key[64])

    Gets the subject public key from a certificate.
- int atcacert_get_subj_key_id (const atcacert_def_t *cert_def, const uint8_t *cert, size_t cert_size, uint8_t subj_key_id[20])

    Gets the subject key ID from a certificate.
- int atcacert_set_signature (const atcacert_def_t *cert_def, uint8_t *cert, size_t *cert_size, size_t max_cert_size, const uint8_t signature[64])

    Sets the signature in a certificate. This may alter the size of the X.509 certificates.
- int atcacert_get_signature (const atcacert_def_t *cert_def, const uint8_t *cert, size_t cert_size, uint8_t signature[64])

    Gets the signature from a certificate.
- int atcacert_set_issue_date (const atcacert_def_t *cert_def, uint8_t *cert, size_t cert_size, const atcacert_tm_utc_t *timestamp)

    Sets the issue date (notBefore) in a certificate. Will be formatted according to the date format specified in the certificate definition.
- int atcacert_get_issue_date (const atcacert_def_t *cert_def, const uint8_t *cert, size_t cert_size, atcacert_tm_utc_t *timestamp)

    Gets the issue date from a certificate. Will be parsed according to the date format specified in the certificate definition.
- int atcacert_set_expire_date (const atcacert_def_t *cert_def, uint8_t *cert, size_t cert_size, const atcacert_tm_utc_t *timestamp)

    Sets the expire date (notAfter) in a certificate. Will be formatted according to the date format specified in the certificate definition.
- int atcacert_get_expire_date (const atcacert_def_t *cert_def, const uint8_t *cert, size_t cert_size, atcacert_tm_utc_t *timestamp)

    Gets the expire date from a certificate. Will be parsed according to the date format specified in the certificate definition.
- int atcacert_set_signer_id (const atcacert_def_t *cert_def, uint8_t *cert, size_t cert_size, const uint8_t signer_id[2])

    Sets the signer ID in a certificate. Will be formatted as 4 upper-case hex digits.
- int atcacert_get_signer_id (const atcacert_def_t *cert_def, const uint8_t *cert, size_t cert_size, uint8_t signer_id[2])

    Gets the signer ID from a certificate. Will be parsed as 4 upper-case hex digits.
- int atcacert_set_cert_sn (const atcacert_def_t *cert_def, uint8_t *cert, size_t *cert_size, size_t max_cert_size, const uint8_t *cert_sn, size_t cert_sn_size)

    Sets the certificate serial number in a certificate.
- int atcacert_gen_cert_sn (const atcacert_def_t *cert_def, uint8_t *cert, size_t cert_size, const uint8_t device_sn[9])

    Sets the certificate serial number by generating it from other information in the certificate using the scheme specified by sn_source in cert_def. See the.
- int atcacert_get_cert_sn (const atcacert_def_t *cert_def, const uint8_t *cert, size_t cert_size, uint8_t *cert_sn, size_t *cert_sn_size)

    Gets the certificate serial number from a certificate.
- int atcacert_set_auth_key_id (const atcacert_def_t *cert_def, uint8_t *cert, size_t cert_size, const uint8_t auth_public_key[64])

    Sets the authority key ID in a certificate. Note that this takes the actual public key creates a key ID from it.
- int atcacert_set_auth_key_id_raw (const atcacert_def_t *cert_def, uint8_t *cert, size_t cert_size, const uint8_t *auth_key_id)

    Sets the authority key ID in a certificate.
- int atcacert_get_auth_key_id (const atcacert_def_t *cert_def, const uint8_t *cert, size_t cert_size, uint8_t auth_key_id[20])

Gets the authority key ID from a certificate.

- int atcacert_set_comp_cert (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t ∗cert_size, size_t max_cert_size, const uint8_t comp_cert[72])

    Sets the signature, issue date, expire date, and signer ID found in the compressed certificate. This also checks fields common between the cert_def and the compressed certificate to make sure they match.

- int atcacert_get_comp_cert (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_t comp_cert[72])

    Generate the compressed certificate for the given certificate.

- int atcacert_get_tbs (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, const uint8_t ∗∗tbs, size_t ∗tbs_size)

    Get a pointer to the TBS data in a certificate.

- int atcacert_get_tbs_digest (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_t tbs_digest[32])

    Get the SHA256 digest of certificate's TBS data.

- int atcacert_set_cert_element (const atcacert_def_t ∗cert_def, const atcacert_cert_loc_t ∗cert_loc, uint8_t ∗cert, size_t cert_size, const uint8_t ∗data, size_t data_size)

    Sets an element in a certificate. The data_size must match the size in cert_loc.

- int atcacert_get_cert_element (const atcacert_def_t ∗cert_def, const atcacert_cert_loc_t ∗cert_loc, const uint8_t ∗cert, size_t cert_size, uint8_t ∗data, size_t data_size)

    Gets an element from a certificate.

- int atcacert_get_key_id (const uint8_t public_key[64], uint8_t key_id[20])

    Calculates the key ID for a given public ECC P256 key.

- int atcacert_merge_device_loc (atcacert_device_loc_t ∗device_locs, size_t ∗device_locs_count, size_t device_locs_max_count, const atcacert_device_loc_t ∗device_loc, size_t block_size)

    Merge a new device location into a list of device locations. If the new location overlaps with an existing location, the existing one will be modified to encompass both. Otherwise the new location is appended to the end of the list.

- int atcacert_is_device_loc_overlap (const atcacert_device_loc_t ∗device_loc1, const atcacert_device_loc_t ∗device_loc2)

    Determines if the two device locations overlap.

- void atcacert_public_key_add_padding (const uint8_t raw_key[64], uint8_t padded_key[72])

    Takes a raw P256 ECC public key and converts it to the padded version used by ATECC devices. Input and output buffers can point to the same location to do an in-place transform.

- void atcacert_public_key_remove_padding (const uint8_t padded_key[72], uint8_t raw_key[64])

    Takes a padded public key used by ATECC devices and converts it to a raw P256 ECC public key. Input and output buffers can point to the same location to do an in-place transform.

- int atcacert_transform_data (atcacert_transform_t transform, const uint8_t ∗data, size_t data_size, uint8_t ∗destination, size_t ∗destination_size)

    Apply the specified transform to the specified data.

- int atcacert_max_cert_size (const atcacert_def_t ∗cert_def, size_t ∗max_cert_size)

    Return the maximum possible certificate size in bytes for a given cert def. Certificate can be variable size, so this gives an appropriate buffer size when reading the certificates.

- int atcacert_der_enc_length (uint32_t length, uint8_t ∗der_length, size_t ∗der_length_size)

    Encode a length in DER format.

- int atcacert_der_dec_length (const uint8_t ∗der_length, size_t ∗der_length_size, uint32_t ∗length)

    Decode a DER format length.

- int atcacert_der_adjust_length (uint8_t ∗der_length, size_t ∗der_length_size, int delta_length, uint32_t ∗new_length)
- int atcacert_der_enc_integer (const uint8_t ∗int_data, size_t int_data_size, uint8_t is_unsigned, uint8_t ∗der_int, size_t ∗der_int_size)

    Encode an ASN.1 integer in DER format, including tag and length fields.

- int atcacert_der_dec_integer (const uint8_t ∗der_int, size_t ∗der_int_size, uint8_t ∗int_data, size_t ∗int_data_size)

    Decode an ASN.1 DER encoded integer.

- int atcacert_der_enc_ecdsa_sig_value (const uint8_t raw_sig[64], uint8_t ∗der_sig, size_t ∗der_sig_size)

    Formats a raw ECDSA P256 signature in the DER encoding found in X.509 certificates.
- int atcacert_der_dec_ecdsa_sig_value (const uint8_t ∗der_sig, size_t ∗der_sig_size, uint8_t raw_sig[64])

    Parses an ECDSA P256 signature in the DER encoding as found in X.509 certificates.
- int atcacert_verify_cert_hw (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, const uint8_t ca_public_key[64])

    Verify a certificate against its certificate authority's public key using the host's ATECC device for crypto functions.
- int atcacert_gen_challenge_hw (uint8_t challenge[32])

    Generate a random challenge to be sent to the client using the RNG on the host's ATECC device.
- int atcacert_verify_response_hw (const uint8_t device_public_key[64], const uint8_t challenge[32], const uint8_t response[64])

    Verify a client's response to a challenge using the host's ATECC device for crypto functions.
- int atcacert_verify_cert_sw (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, const uint8_t ca_public_key[64])

    Verify a certificate against its certificate authority's public key using software crypto functions.The function is currently not implemented.
- int atcacert_gen_challenge_sw (uint8_t challenge[32])

    Generate a random challenge to be sent to the client using a software PRNG.The function is currently not implemented.
- int atcacert_verify_response_sw (const uint8_t device_public_key[64], const uint8_t challenge[32], const uint8_t response[64])

    Verify a client's response to a challenge using software crypto functions.The function is currently not implemented.

Variables

- const size_t ATCACERT_DATE_FORMAT_SIZES [ATCACERT_DATE_FORMAT_SIZES_COUNT]

13.5.1 Detailed Description

These methods provide convenient ways to perform certification I/O with CryptoAuth chips and perform certificate manipulation in memory.

13.5.2 Macro Definition Documentation

13.5.2.1 ATCACERT_DATE_FORMAT_SIZES_COUNT

```
#define ATCACERT_DATE_FORMAT_SIZES_COUNT 5
```

13.5.2.2 ATCACERT_E_BAD_CERT

```
#define ATCACERT_E_BAD_CERT 10
```

Certificate structure is bad in some way.

13.5.2.3 ATCACERT_E_BAD_PARAMS

```
#define ATCACERT_E_BAD_PARAMS 2
```

Invalid/bad parameter passed to function.

13.5.2.4 ATCACERT_E_BUFFER_TOO_SMALL

```
#define ATCACERT_E_BUFFER_TOO_SMALL 3
```

Supplied buffer for output is too small to hold the result.

13.5.2.5 ATCACERT_E_DECODING_ERROR

```
#define ATCACERT_E_DECODING_ERROR 4
```

Data being decoded/parsed has an invalid format.

13.5.2.6 ATCACERT_E_ELEM_MISSING

```
#define ATCACERT_E_ELEM_MISSING 8
```

The certificate element isn't defined for the certificate definition.

13.5.2.7 ATCACERT_E_ELEM_OUT_OF_BOUNDS

```
#define ATCACERT_E_ELEM_OUT_OF_BOUNDS 9
```

Certificate element is out of bounds for the given certificate.

13.5.2.8 ATCACERT_E_ERROR

```
#define ATCACERT_E_ERROR 1
```

General error.

13.5.2.9 ATCACERT_E_INVALID_DATE

```
#define ATCACERT_E_INVALID_DATE 5
```

Date is invalid.

13.5.2.10 ATCACERT_E_INVALID_TRANSFORM

```
#define ATCACERT_E_INVALID_TRANSFORM 13
```

Invalid transform passed to function.

13.5.2.11 ATCACERT_E_SUCCESS

```
#define ATCACERT_E_SUCCESS 0
```

Operation completed successfully.

13.5.2.12 ATCACERT_E_UNEXPECTED_ELEM_SIZE

```
#define ATCACERT_E_UNEXPECTED_ELEM_SIZE 7
```

A certificate element size was not what was expected.

13.5.2.13 ATCACERT_E_UNIMPLEMENTED

```
#define ATCACERT_E_UNIMPLEMENTED 6
```

Function is unimplemented for the current configuration.

13.5.2.14 ATCACERT_E_VERIFY_FAILED

```
#define ATCACERT_E_VERIFY_FAILED 12
```

Certificate or challenge/response verification failed.

13.5.2.15 ATCACERT_E_WRONG_CERT_DEF

```
#define ATCACERT_E_WRONG_CERT_DEF 11
```

13.5.2.16 DATEFMT_ISO8601_SEP_SIZE

```
#define DATEFMT_ISO8601_SEP_SIZE (20)
```

13.5.2.17 DATEFMT_MAX_SIZE

```
#define DATEFMT_MAX_SIZE DATEFMT_ISO8601_SEP_SIZE
```

13.5.2.18 DATEFMT_POSIX_UINT32_BE_SIZE

```
#define DATEFMT_POSIX_UINT32_BE_SIZE (4)
```

13.5.2.19 DATEFMT_POSIX_UINT32_LE_SIZE

```
#define DATEFMT_POSIX_UINT32_LE_SIZE (4)
```

13.5.2.20 DATEFMT_RFC5280_GEN_SIZE

```
#define DATEFMT_RFC5280_GEN_SIZE (15)
```

13.5.2.21 DATEFMT_RFC5280_UTC_SIZE

```
#define DATEFMT_RFC5280_UTC_SIZE (13)
```

13.5.2.22 FALSE

```
#define FALSE (0)
```

13.5.2.23 TRUE

```
#define TRUE (1)
```

13.5.3 Typedef Documentation

13.5.3.1 atcacert_build_state_t

```
typedef struct atcacert_build_state_s atcacert_build_state_t
```

Tracks the state of a certificate as it's being rebuilt from device information.

13.5.3.2 atcacert_cert_element_t

```
typedef struct atcacert_cert_element_s atcacert_cert_element_t
```

Defines a generic dynamic element for a certificate including the device and template locations.

13.5.3.3 atcacert_cert_loc_t

```
typedef struct atcacert_cert_loc_s atcacert_cert_loc_t
```

Defines a chunk of data in a certificate template.

13.5.3.4 atcacert_cert_sn_src_t

```
typedef enum atcacert_cert_sn_src_e atcacert_cert_sn_src_t
```

Sources for the certificate serial number.

13.5.3.5 atcacert_cert_type_t

```
typedef enum atcacert_cert_type_e atcacert_cert_type_t
```

Types of certificates.

13.5.3.6 atcacert_date_format_t

```
typedef enum atcacert_date_format_e atcacert_date_format_t
```

Date formats.

13.5.3.7 atcacert_def_t

```
typedef struct atcacert_def_s atcacert_def_t
```

Defines a certificate and all the pieces to work with it.

If any of the standard certificate elements (std_cert_elements) are not a part of the certificate definition, set their count to 0 to indicate their absence.

13.5.3.8 atcacert_device_loc_t

```
typedef struct atcacert_device_loc_s atcacert_device_loc_t
```

Defines a chunk of data in an ATECC device.

13.5.3.9 atcacert_device_zone_t

```
typedef enum atcacert_device_zone_e atcacert_device_zone_t
```

ATECC device zones. The values match the Zone Encodings as specified in the datasheet.

13.5.3.10 atcacert_std_cert_element_t

```
typedef enum atcacert_std_cert_element_e atcacert_std_cert_element_t
```

Standard dynamic certificate elements.

13.5.3.11 atcacert_tm_utc_t

```
typedef struct atcacert_tm_utc_s atcacert_tm_utc_t
```

Holds a broken-down date in UTC. Mimics atcacert_tm_utc_t from time.h.

13.5.3.12 atcacert_transform_t

```
typedef enum atcacert_transform_e atcacert_transform_t
```

How to transform the data from the device to the certificate.

13.5.4 Enumeration Type Documentation

13.5.4.1 atcacert_cert_sn_src_e

```
enum atcacert_cert_sn_src_e
```

Sources for the certificate serial number.

Enumerator

SNSRC_STORED	Cert serial is stored on the device.
SNSRC_STORED_DYNAMIC	Cert serial is stored on the device with the first byte being the DER size (X509 certs only).
SNSRC_DEVICE_SN	Cert serial number is 0x40(MSB) + 9-byte device serial number. Only applies to device certificates.
SNSRC_SIGNER_ID	Cert serial number is 0x40(MSB) + 2-byte signer ID. Only applies to signer certificates.
SNSRC_PUB_KEY_HASH	Cert serial number is the SHA256(Subject public key + Encoded dates), with uppermost 2 bits set to 01.
SNSRC_DEVICE_SN_HASH	Cert serial number is the SHA256(Device SN + Encoded dates), with uppermost 2 bits set to 01. Only applies to device certificates.
SNSRC_PUB_KEY_HASH_POS	Depreciated, don't use. Cert serial number is the SHA256(Subject public key + Encoded dates), with MSBit set to 0 to ensure it's positive.
SNSRC_DEVICE_SN_HASH_POS	Depreciated, don't use. Cert serial number is the SHA256(Device SN + Encoded dates), with MSBit set to 0 to ensure it's positive. Only applies to device certificates.
SNSRC_PUB_KEY_HASH_RAW	Depreciated, don't use. Cert serial number is the SHA256(Subject public key + Encoded dates).
SNSRC_DEVICE_SN_HASH_RAW	Depreciated, don't use. Cert serial number is the SHA256(Device SN + Encoded dates). Only applies to device certificates.

13.5.4.2 atcacert_cert_type_e

```
enum atcacert_cert_type_e
```

Types of certificates.

Enumerator

CERTTYPE_X509	Standard X509 certificate.
CERTTYPE_CUSTOM	Custom format.

13.5.4.3 atcacert_date_format_e

```
enum atcacert_date_format_e
```

Date formats.

Enumerator

DATEFMT_ISO8601_SEP	ISO8601 full date YYYY-MM-DDThh:mm:ssZ.
DATEFMT_RFC5280_UTC	RFC 5280 (X.509) 4.1.2.5.1 UTCTime format YYMMDDhhmmssZ.
DATEFMT_POSIX_UINT32_BE	POSIX (aka UNIX) date format. Seconds since Jan 1, 1970. 32 bit unsigned integer, big endian.

Enumerator

DATEFMT_POSIX_UINT32_LE	POSIX (aka UNIX) date format. Seconds since Jan 1, 1970. 32 bit unsigned integer, little endian.
DATEFMT_RFC5280_GEN	RFC 5280 (X.509) 4.1.2.5.2 GeneralizedTime format YYYYMMDDhhmmssZ.

13.5.4.4 atcacert_device_zone_e

```
enum atcacert_device_zone_e
```

ATECC device zones. The values match the Zone Encodings as specified in the datasheet.

Enumerator

DEVZONE_CONFIG	Configuration zone.
DEVZONE_OTP	One Time Programmable zone.
DEVZONE_DATA	Data zone (slots).
DEVZONE_NONE	Special value used to indicate there is no device location.

13.5.4.5 atcacert_std_cert_element_e

```
enum atcacert_std_cert_element_e
```

Standard dynamic certificate elements.

Enumerator

STDCERT_PUBLIC_KEY	
STDCERT_SIGNATURE	
STDCERT_ISSUE_DATE	
STDCERT_EXPIRE_DATE	
STDCERT_SIGNER_ID	
STDCERT_CERT_SN	
STDCERT_AUTH_KEY_ID	
STDCERT_SUBJ_KEY_ID	
STDCERT_NUM_ELEMENTS	Special item to give the number of elements in this enum.

13.5.4.6 atcacert_transform_e

```
enum atcacert_transform_e
```

How to transform the data from the device to the certificate.

Enumerator

TF_NONE	No transform, data is used byte for byte.
TF_REVERSE	Reverse the bytes (e.g. change endianness)
TF_BIN2HEX_UC	Convert raw binary into ASCII hex, uppercase.
TF_BIN2HEX_LC	Convert raw binary into ASCII hex, lowercase.
TF_HEX2BIN_UC	Convert ASCII hex, uppercase to binary.
TF_HEX2BIN_LC	Convert ASCII hex, lowercase to binary.
TF_BIN2HEX_SPACE_UC	Convert raw binary into ASCII hex, uppercase space between bytes.
TF_BIN2HEX_SPACE_LC	Convert raw binary into ASCII hex, lowercase space between bytes.
TF_HEX2BIN_SPACE_UC	Convert ASCII hex, uppercase with spaces between bytes to binary.
TF_HEX2BIN_SPACE_LC	Convert ASCII hex, lowercase with spaces between bytes to binary.

13.5.5 Function Documentation

13.5.5.1 atcacert_cert_build_finish()

```
int atcacert_cert_build_finish (
            atcacert_build_state_t * build_state )
```

Completes any final certificate processing required after all data from the device has been incorporated.

The final certificate and its size in bytes are contained in the cert and cert_size elements of the build_state structure. This will be the same buffers as supplied to the atcacert_cert_build_start function at the beginning of the certificate rebuilding process.

Parameters

in	*build_state*	Current certificate build state.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.2 atcacert_cert_build_process()

```
int atcacert_cert_build_process (
            atcacert_build_state_t * build_state,
            const atcacert_device_loc_t * device_loc,
            const uint8_t * device_data )
```

Process information read from the ATECC device. If it contains information for the certificate, it will be incorporated into the certificate.

Parameters

in	*build_state*	Current certificate building state.
in	*device_loc*	Device location structure describing where on the device the following data came from.
in	*device_data*	Actual data from the device. It should represent the offset and byte count specified in the device_loc parameter.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.3 atcacert_cert_build_start()

```
int atcacert_cert_build_start (
            atcacert_build_state_t * build_state,
            const atcacert_def_t * cert_def,
            uint8_t * cert,
            size_t * cert_size,
            const uint8_t ca_public_key[64] )
```

Starts the certificate rebuilding process.

Parameters

out	*build_state*	Structure is initialized to start the certificate building process. Will be passed to the other certificate building functions.
in	*cert_def*	Certificate definition for the certificate being built.
in	*cert*	Buffer to contain the rebuilt certificate.
in	*cert_size*	As input, the size of the cert buffer in bytes. This value will be adjusted to the current/final size of the certificate through the building process.
in	*ca_public_key*	ECC P256 public key of the certificate authority (issuer) for the certificate being built. Set to NULL if the authority key id is not needed, set properly in the cert_def template, or stored on the device as specified in the cert_def cert_elements.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.4 atcacert_create_csr()

```
int atcacert_create_csr (
            const atcacert_def_t * csr_def,
            uint8_t * csr,
            size_t * csr_size )
```

Creates a CSR specified by the CSR definition from the ATECC508A device. This process involves reading the dynamic CSR data from the device and combining it with the template found in the CSR definition, then signing it. Return the CSR int der format.

Parameters

in	*csr_def*	CSR definition describing where to find the dynamic CSR information on the device and how to incorporate it into the template.
out	*csr*	Buffer to receive the CSR.
in,out	*csr_size*	As input, the size of the CSR buffer in bytes. As output, the size of the CSR returned in cert in bytes.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.5.5.5 atcacert_create_csr_pem()

```
int atcacert_create_csr_pem (
            const atcacert_def_t * csr_def,
            char * csr,
            size_t * csr_size )
```

Creates a CSR specified by the CSR definition from the ATECC508A device. This process involves reading the dynamic CSR data from the device and combining it with the template found in the CSR definition, then signing it. Return the CSR int der format.

Parameters

in	*csr_def*	CSR definition describing where to find the dynamic CSR information on the device and how to incorporate it into the template.
out	*csr*	Buffer to received the CSR formatted as PEM.
in,out	*csr_size*	As input, the size of the CSR buffer in bytes. As output, the size of the CSR as PEM returned in cert in bytes.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.5.5.6 atcacert_date_dec()

```
int atcacert_date_dec (
            atcacert_date_format_t format,
            const uint8_t * formatted_date,
            size_t formatted_date_size,
            atcacert_tm_utc_t * timestamp )
```

Parse a formatted timestamp according to the specified format.

Parameters

in	*format*	Format to parse the formatted date as.
in	*formatted_date*	Formatted date to be parsed.
in	*formatted_date_size*	Size of the formatted date in bytes.
out	*timestamp*	Parsed timestamp is returned here.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.7 atcacert_date_dec_compcert()

```
int atcacert_date_dec_compcert (
            const uint8_t enc_dates[3],
            atcacert_date_format_t expire_date_format,
            atcacert_tm_utc_t * issue_date,
            atcacert_tm_utc_t * expire_date )
```

Decode the issue and expire dates from the format used by the compressed certificate.

Parameters

in	*enc_dates*	Encoded date from the compressed certificate. 3 bytes.
in	*expire_date_format*	Expire date format. Only used to determine max date when no expiration date is specified by the encoded date.
out	*issue_date*	Decoded issue date is returned here.
out	*expire_date*	Decoded expire date is returned here. If there is no expiration date, the expire date will be set to a maximum value for the given expire_date_format.

Returns

0 on success

13.5.5.8 atcacert_date_dec_iso8601_sep()

```
int atcacert_date_dec_iso8601_sep (
            const uint8_t formatted_date[DATEFMT_ISO8601_SEP_SIZE],
            atcacert_tm_utc_t * timestamp )
```

13.5.5.9 atcacert_date_dec_posix_uint32_be()

```
int atcacert_date_dec_posix_uint32_be (
            const uint8_t formatted_date[DATEFMT_POSIX_UINT32_BE_SIZE],
            atcacert_tm_utc_t * timestamp )
```

13.5.5.10 atcacert_date_dec_posix_uint32_le()

```
int atcacert_date_dec_posix_uint32_le (
            const uint8_t formatted_date[DATEFMT_POSIX_UINT32_LE_SIZE],
            atcacert_tm_utc_t * timestamp )
```

13.5.5.11 atcacert_date_dec_rfc5280_gen()

```
int atcacert_date_dec_rfc5280_gen (
            const uint8_t formatted_date[DATEFMT_RFC5280_GEN_SIZE],
            atcacert_tm_utc_t * timestamp )
```

13.5.5.12 atcacert_date_dec_rfc5280_utc()

```
int atcacert_date_dec_rfc5280_utc (
            const uint8_t formatted_date[DATEFMT_RFC5280_UTC_SIZE],
            atcacert_tm_utc_t * timestamp )
```

13.5.5.13 atcacert_date_enc()

```
int atcacert_date_enc (
            atcacert_date_format_t format,
            const atcacert_tm_utc_t * timestamp,
            uint8_t * formatted_date,
            size_t * formatted_date_size )
```

Format a timestamp according to the format type.

Parameters

in	*format*	Format to use.
in	*timestamp*	Timestamp to format.
out	*formatted_date*	Formatted date will be returned in this buffer.
in,out	*formatted_date_size*	As input, the size of the formatted_date buffer. As output, the size of the returned formatted_date.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.14 atcacert_date_enc_compcert()

```
int atcacert_date_enc_compcert (
            const atcacert_tm_utc_t * issue_date,
            uint8_t expire_years,
            uint8_t enc_dates[3] )
```

Encode the issue and expire dates in the format used by the compressed certificate.

Parameters

in	*issue_date*	Issue date to encode. Note that minutes and seconds will be ignored.
in	*expire_years*	Expire date is expressed as a number of years past the issue date. 0 should be used if there is no expire date.
out	*enc_dates*	Encoded dates for use in the compressed certificate is returned here. 3 bytes.

Returns

0 on success

13.5.5.15 atcacert_date_enc_iso8601_sep()

```
int atcacert_date_enc_iso8601_sep (
            const atcacert_tm_utc_t * timestamp,
            uint8_t formatted_date[DATEFMT_ISO8601_SEP_SIZE] )
```

13.5.5.16 atcacert_date_enc_posix_uint32_be()

```
int atcacert_date_enc_posix_uint32_be (
            const atcacert_tm_utc_t * timestamp,
            uint8_t formatted_date[DATEFMT_POSIX_UINT32_BE_SIZE] )
```

13.5.5.17 atcacert_date_enc_posix_uint32_le()

```
int atcacert_date_enc_posix_uint32_le (
            const atcacert_tm_utc_t * timestamp,
            uint8_t formatted_date[DATEFMT_POSIX_UINT32_LE_SIZE] )
```

13.5.5.18 atcacert_date_enc_rfc5280_gen()

```
int atcacert_date_enc_rfc5280_gen (
            const atcacert_tm_utc_t * timestamp,
            uint8_t formatted_date[DATEFMT_RFC5280_GEN_SIZE] )
```

13.5.5.19 atcacert_date_enc_rfc5280_utc()

```
int atcacert_date_enc_rfc5280_utc (
            const atcacert_tm_utc_t * timestamp,
            uint8_t formatted_date[DATEFMT_RFC5280_UTC_SIZE] )
```

13.5.5.20 atcacert_date_get_max_date()

```
int atcacert_date_get_max_date (
            atcacert_date_format_t format,
            atcacert_tm_utc_t * timestamp )
```

Return the maximum date available for the given format.

Parameters

in	*format*	Format to get the max date for.
out	*timestamp*	Max date is returned here.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.21 atcacert_der_adjust_length()

```
int atcacert_der_adjust_length (
            uint8_t * der_length,
            size_t * der_length_size,
            int delta_length,
            uint32_t * new_length )
```

13.5.5.22 atcacert_der_dec_ecdsa_sig_value()

```
int atcacert_der_dec_ecdsa_sig_value (
            const uint8_t * der_sig,
            size_t * der_sig_size,
            uint8_t raw_sig[64] )
```

Parses an ECDSA P256 signature in the DER encoding as found in X.509 certificates.

This will parse the DER encoding of the signatureValue field as found in an X.509 certificate (RFC 5280). x509_sig should include the tag, length, and value. The value of the signatureValue is the DER encoding of the ECDSA-Sig-Value as specified by RFC 5480 and SECG SEC1.

Parameters

in	*der_sig*	X.509 format signature (TLV of signatureValue) to be parsed.
in,out	*der_sig_size*	As input, size of the der_sig buffer in bytes. As output, size of the DER x.509 signature parsed from the buffer.
out	*raw_sig*	Parsed P256 ECDSA signature will be returned in this buffer. Formatted as R and S integers concatenated together. 64 bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.23 atcacert_der_dec_integer()

```
int atcacert_der_dec_integer (
            const uint8_t * der_int,
            size_t * der_int_size,
            uint8_t * int_data,
            size_t * int_data_size )
```

Decode an ASN.1 DER encoded integer.

X.680 (http://www.itu.int/rec/T-REC-X.680/en) section 19.8, for tag value X.690 (http://www.itu.int/rec/T-REC-X.690/en) section 8.3, for encoding

Parameters

in	*der_int*	DER encoded ASN.1 integer, including the tag and length fields.
in,out	*der_int_size*	As input, the size of the der_int buffer in bytes. As output, the size of the DER integer decoded in bytes.
out	*int_data*	Decode integer is returned in this buffer in a signed big-endian format.
in,out	*int_data_size*	As input, the size of int_data in bytes. As output, the size of the decoded integer in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.24 atcacert_der_dec_length()

```
int atcacert_der_dec_length (
            const uint8_t * der_length,
            size_t * der_length_size,
            uint32_t * length )
```

Decode a DER format length.

X.690 (http://www.itu.int/rec/T-REC-X.690/en) section 8.1.3, for encoding

Parameters

in	*der_length*	DER encoded length.
in,out	*der_length_size*	As input, the size of the der_length buffer in bytes. As output, the size of the DER encoded length that was decoded.
out	*length*	Decoded length is returned here.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.25 atcacert_der_enc_ecdsa_sig_value()

```
int atcacert_der_enc_ecdsa_sig_value (
            const uint8_t raw_sig[64],
            uint8_t * der_sig,
            size_t * der_sig_size )
```

Formats a raw ECDSA P256 signature in the DER encoding found in X.509 certificates.

This will return the DER encoding of the signatureValue field as found in an X.509 certificate (RFC 5280). This include the tag, length, and value. The value of the signatureValue is the DER encoding of the ECDSA-Sig-Value as specified by RFC 5480 and SECG SEC1.

Parameters

in	*raw_sig*	P256 ECDSA signature to be formatted. Input format is R and S integers concatenated together. 64 bytes.
out	*der_sig*	X.509 format signature (TLV of signatureValue) will be returned in this buffer.
in,out	*der_sig_size*	As input, the size of the x509_sig buffer in bytes. As output, the size of the returned X.509 signature in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.26 atcacert_der_enc_integer()

```
int atcacert_der_enc_integer (
            const uint8_t * int_data,
            size_t int_data_size,
            uint8_t is_unsigned,
            uint8_t * der_int,
            size_t * der_int_size )
```

Encode an ASN.1 integer in DER format, including tag and length fields.

X.680 (http://www.itu.int/rec/T-REC-X.680/en) section 19.8, for tag value X.690 (http://www.itu.int/rec/T-REC-X.690/en) section 8.3, for encoding

Parameters

in	*int_data*	Raw integer in big-endian format.
in	*int_data_size*	Size of the raw integer in bytes.
in	*is_unsigned*	Indicate whether the input integer should be treated as unsigned.
out	*der_int*	DER encoded integer is returned in this buffer.
in,out	*der_int_size*	As input, the size of the der_int buffer in bytes. As output, the size of the DER integer returned in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.27 atcacert_der_enc_length()

```
int atcacert_der_enc_length (
            uint32_t length,
            uint8_t * der_length,
            size_t * der_length_size )
```

Encode a length in DER format.

X.690 (http://www.itu.int/rec/T-REC-X.690/en) section 8.1.3, for encoding

Parameters

in	*length*	Length to be encoded.
out	*der_length*	DER encoded length will returned in this buffer.
in,out	*der_length_size*	As input, size of der_length buffer in bytes. As output, the size of the DER length encoding in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.28 atcacert_gen_cert_sn()

```
int atcacert_gen_cert_sn (
            const atcacert_def_t * cert_def,
            uint8_t * cert,
            size_t cert_size,
            const uint8_t device_sn[9] )
```

Sets the certificate serial number by generating it from other information in the certificate using the scheme specified by sn_source in cert_def. See the.

This method requires certain elements in the certificate be set properly as they're used for generating the serial number. See atcacert_cert_sn_src_t for what elements should be set in the certificate beforehand. If the sn_source is set to SNSRC_STORED or SNSRC_STORED_DYNAMIC, the function will return ATCACERT_E_SUCCESS without making any changes to the certificate.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in,out	*cert*	Certificate to update.
in	*cert_size*	Size of the certificate (cert) in bytes.
in	*device_sn*	Device serial number, only used if required by the sn_source scheme. Can be set to NULL, if not required.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.29 atcacert_gen_challenge_hw()

```
int atcacert_gen_challenge_hw (
            uint8_t challenge[32] )
```

Generate a random challenge to be sent to the client using the RNG on the host's ATECC device.

Parameters

out	*challenge*	Random challenge is return here. 32 bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.30 atcacert_gen_challenge_sw()

```
int atcacert_gen_challenge_sw (
            uint8_t challenge[32] )
```

Generate a random challenge to be sent to the client using a software PRNG.The function is currently not implemented.

Parameters

out	*challenge*	Random challenge is return here. 32 bytes.

Returns

ATCA_UNIMPLEMENTED , as the function is currently not implemented.

13.5.5.31 atcacert_get_auth_key_id()

```
int atcacert_get_auth_key_id (
            const atcacert_def_t * cert_def,
            const uint8_t * cert,
            size_t cert_size,
            uint8_t auth_key_id[20] )
```

Gets the authority key ID from a certificate.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in	*cert*	Certificate to get element from.
in	*cert_size*	Size of the certificate (cert) in bytes.
out	*auth_key_id*	Authority key ID is returned in this buffer. 20 bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.32 atcacert_get_cert_element()

```
int atcacert_get_cert_element (
            const atcacert_def_t * cert_def,
```

```
            const atcacert_cert_loc_t * cert_loc,
            const uint8_t * cert,
            size_t cert_size,
            uint8_t * data,
            size_t data_size )
```

Gets an element from a certificate.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in	*cert_loc*	Certificate location for this element.
in	*cert*	Certificate to get element from.
in	*cert_size*	Size of the certificate (cert) in bytes.
out	*data*	Element data will be returned in this buffer. This buffer must be large enough to hold cert_loc.count bytes.
in	*data_size*	Expected size of the cert element data.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.33 atcacert_get_cert_sn()

```
int atcacert_get_cert_sn (
            const atcacert_def_t * cert_def,
            const uint8_t * cert,
            size_t cert_size,
            uint8_t * cert_sn,
            size_t * cert_sn_size )
```

Gets the certificate serial number from a certificate.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in	*cert*	Certificate to get element from.
in	*cert_size*	Size of the certificate (cert) in bytes.
out	*cert_sn*	Certificate SN will be returned in this buffer.
in,out	*cert_sn_size*	As input, the size of the cert_sn buffer. As output, the size of the certificate SN (cert_sn) in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.34 atcacert_get_comp_cert()

```
int atcacert_get_comp_cert (
            const atcacert_def_t * cert_def,
            const uint8_t * cert,
            size_t cert_size,
            uint8_t comp_cert[72] )
```

Generate the compressed certificate for the given certificate.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in	*cert*	Certificate to generate the compressed certificate for.
in	*cert_size*	Size of the certificate (cert) in bytes.
out	*comp_cert*	Compressed certificate is returned in this buffer. 72 bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.35 atcacert_get_device_data()

```
int atcacert_get_device_data (
            const atcacert_def_t * cert_def,
            const uint8_t * cert,
            size_t cert_size,
            const atcacert_device_loc_t * device_loc,
            uint8_t * device_data )
```

Gets the dynamic data that would be saved to the specified device location. This function is primarily used to break down a full certificate into the dynamic components to be saved to a device.

The atcacert_add_device_locs function can be used to generate a list of device locations a particular certificate definition requires.

Parameters

in	*cert_def*	Certificate definition for the certificate we're getting data from.
in	*cert*	Certificate to get the device data from.
in	*cert_size*	Size of the certificate in bytes.
in	*device_loc*	Device location to request data for.
out	*device_data*	Buffer that represents the device data in device_loc. Required to be at least device_loc.count in size.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.36 atcacert_get_device_locs()

```
int atcacert_get_device_locs (
            const atcacert_def_t * cert_def,
            atcacert_device_loc_t * device_locs,
            size_t * device_locs_count,
            size_t device_locs_max_count,
            size_t block_size )
```

Add all the device locations required to rebuild the specified certificate (cert_def) to a device locations list.

The block_size parameter will adjust all added device locations to have a offset and count that aligns with that block size. This allows one to generate a list of device locations that matches specific read or write semantics (e.g. 4 byte or 32 byte reads).

Parameters

in	*cert_def*	Certificate definition containing all the device locations to add to the list.
in,out	*device_locs*	List of device locations to add to.
in,out	*device_locs_count*	As input, existing size of the device locations list. As output, the new size of the device locations list.
in	*device_locs_max_count*	Maximum number of elements device_locs can hold.
in	*block_size*	Block size to align all offsets and counts to when adding device locations.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.37 atcacert_get_expire_date()

```
int atcacert_get_expire_date (
            const atcacert_def_t * cert_def,
            const uint8_t * cert,
            size_t cert_size,
            atcacert_tm_utc_t * timestamp )
```

Gets the expire date from a certificate. Will be parsed according to the date format specified in the certificate definition.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in	*cert*	Certificate to get element from.
in	*cert_size*	Size of the certificate (cert) in bytes.
out	*timestamp*	Expire date is returned in this structure.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.38 atcacert_get_issue_date()

```
int atcacert_get_issue_date (
            const atcacert_def_t * cert_def,
            const uint8_t * cert,
            size_t cert_size,
            atcacert_tm_utc_t * timestamp )
```

Gets the issue date from a certificate. Will be parsed according to the date format specified in the certificate definition.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in	*cert*	Certificate to get element from.
in	*cert_size*	Size of the certificate (cert) in bytes.
out	*timestamp*	Issue date is returned in this structure.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.39 atcacert_get_key_id()

```
int atcacert_get_key_id (
            const uint8_t public_key[64],
            uint8_t key_id[20] )
```

Calculates the key ID for a given public ECC P256 key.

Uses method 1 for calculating the keyIdentifier as specified by RFC 5280, section 4.2.1.2: (1) The keyIdentifier is composed of the 160-bit SHA-1 hash of the value of the BIT STRING subjectPublicKey (excluding the tag, length, and number of unused bits).

Parameters

in	*public_key*	ECC P256 public key to calculate key key ID for. Formatted as the X and Y integers concatenated together. 64 bytes.
in	*key_id*	Calculated key ID will be returned in this buffer. 20 bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.40 atcacert_get_response()

```
int atcacert_get_response (
            uint8_t device_private_key_slot,
            const uint8_t challenge[32],
            uint8_t response[64] )
```

Calculates the response to a challenge sent from the host.

The challenge-response protocol is an ECDSA Sign and Verify. This performs the ECDSA Sign on the challenge and returns the signature as the response.

Parameters

in	*device_private_key_slot*	Slot number for the device's private key. This must be the same slot used to generate the public key included in the device's certificate.
in	*challenge*	Challenge to generate the response for. Must be 32 bytes.
out	*response*	Response will be returned in this buffer. 64 bytes.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.5.5.41 atcacert_get_signature()

```
int atcacert_get_signature (
            const atcacert_def_t * cert_def,
            const uint8_t * cert,
            size_t cert_size,
            uint8_t signature[64] )
```

Gets the signature from a certificate.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in	*cert*	Certificate to get element from.
in	*cert_size*	Size of the certificate (cert) in bytes.
out	*signature*	Signature is returned in this buffer. Formatted at R and S integers concatenated together. 64 bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.42 atcacert_get_signer_id()

```
int atcacert_get_signer_id (
            const atcacert_def_t * cert_def,
            const uint8_t * cert,
            size_t cert_size,
            uint8_t signer_id[2] )
```

Gets the signer ID from a certificate. Will be parsed as 4 upper-case hex digits.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in	*cert*	Certificate to get element from.
in	*cert_size*	Size of the certificate (cert) in bytes.
out	*signer_id*	Signer ID will be returned in this buffer. 2 bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.43 atcacert_get_subj_key_id()

```
int atcacert_get_subj_key_id (
            const atcacert_def_t * cert_def,
            const uint8_t * cert,
            size_t cert_size,
            uint8_t subj_key_id[20] )
```

Gets the subject key ID from a certificate.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in	*cert*	Certificate to get element from.
in	*cert_size*	Size of the certificate (cert) in bytes.
out	*subj_key_id*	Subject key ID is returned in this buffer. 20 bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.44 atcacert_get_subj_public_key()

```
int atcacert_get_subj_public_key (
            const atcacert_def_t * cert_def,
            const uint8_t * cert,
            size_t cert_size,
            uint8_t subj_public_key[64] )
```

Gets the subject public key from a certificate.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in	*cert*	Certificate to get element from.
in	*cert_size*	Size of the certificate (cert) in bytes.
out	*subj_public_key*	Subject public key is returned in this buffer. Formatted at X and Y integers concatenated together. 64 bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.45 atcacert_get_tbs()

```
int atcacert_get_tbs (
            const atcacert_def_t * cert_def,
            const uint8_t * cert,
            size_t cert_size,
            const uint8_t ** tbs,
            size_t * tbs_size )
```

Get a pointer to the TBS data in a certificate.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in	*cert*	Certificate to get the TBS data pointer for.
in	*cert_size*	Size of the certificate (cert) in bytes.
out	*tbs*	Pointer to a const pointer that will be set the start of the TBS data.
out	*tbs_size*	Size of the TBS data will be returned here.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.46 atcacert_get_tbs_digest()

```
int atcacert_get_tbs_digest (
            const atcacert_def_t * cert_def,
            const uint8_t * cert,
            size_t cert_size,
            uint8_t tbs_digest[32] )
```

Get the SHA256 digest of certificate's TBS data.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in	*cert*	Certificate to get the TBS data pointer for.
in	*cert_size*	Size of the certificate (cert) in bytes.
out	*tbs_digest*	TBS data digest will be returned here. 32 bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.47 atcacert_is_device_loc_overlap()

```
int atcacert_is_device_loc_overlap (
            const atcacert_device_loc_t * device_loc1,
            const atcacert_device_loc_t * device_loc2 )
```

Determines if the two device locations overlap.

Parameters

in	*device_loc1*	First device location to check.
in	*device_loc2*	Second device location o check.

Returns

0 (false) if they don't overlap, non-zero if the do overlap.

13.5.5.48 atcacert_max_cert_size()

```
int atcacert_max_cert_size (
            const atcacert_def_t * cert_def,
            size_t * max_cert_size )
```

Return the maximum possible certificate size in bytes for a given cert def. Certificate can be variable size, so this gives an appropriate buffer size when reading the certificates.

Parameters

in	*cert_def*	Certificate definition to find a max size for.
out	*max_cert_size*	Maximum certificate size will be returned here in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.49 atcacert_merge_device_loc()

```
int atcacert_merge_device_loc (
            atcacert_device_loc_t * device_locs,
            size_t * device_locs_count,
            size_t device_locs_max_count,
            const atcacert_device_loc_t * device_loc,
            size_t block_size )
```

Merge a new device location into a list of device locations. If the new location overlaps with an existing location, the existing one will be modified to encompass both. Otherwise the new location is appended to the end of the list.

The block_size parameter will adjust all added device locations to have an offset and count that aligns with that block size. This allows one to generate a list of device locations that matches specific read/write semantics (e.g. 4 byte or 32 byte reads). Note that this block_size only applies to the device_loc being added. Existing device locations in the list won't be modified to match the block size.

Parameters

in,out	*device_locs*	Existing device location list to merge the new device location into.
in,out	*device_locs_count*	As input, the existing number of items in the device_locs list. As output, the new size of the device_locs list.
in	*device_locs_max_count*	Maximum number of items the device_locs list can hold.
in	*device_loc*	New device location to be merged into the device_locs list.
in	*block_size*	Block size to align all offsets and counts to when adding device location.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.50 atcacert_public_key_add_padding()

```
void atcacert_public_key_add_padding (
            const uint8_t raw_key[64],
            uint8_t padded_key[72] )
```

Takes a raw P256 ECC public key and converts it to the padded version used by ATECC devices. Input and output buffers can point to the same location to do an in-place transform.

Parameters

in	*raw_key*	Public key as X and Y integers concatenated together. 64 bytes.
out	*padded_key*	Padded key is returned in this buffer. X and Y integers are padded with 4 bytes of 0 in the MSB. 72 bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.51 atcacert_public_key_remove_padding()

```
void atcacert_public_key_remove_padding (
            const uint8_t padded_key[72],
            uint8_t raw_key[64] )
```

Takes a padded public key used by ATECC devices and converts it to a raw P256 ECC public key. Input and output buffers can point to the same location to do an in-place transform.

Parameters

out	*padded_key*	X and Y integers are padded with 4 bytes of 0 in the MSB. 72 bytes.
in	*raw_key*	Raw key is returned in this buffer. Public key as X and Y integers concatenated together. 64 bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.52 atcacert_read_cert()

```
int atcacert_read_cert (
            const atcacert_def_t * cert_def,
            const uint8_t ca_public_key[64],
            uint8_t * cert,
            size_t * cert_size )
```

Reads the certificate specified by the certificate definition from the ATECC508A device.

This process involves reading the dynamic cert data from the device and combining it with the template found in the certificate definition.

Parameters

in	*cert_def*	Certificate definition describing where to find the dynamic certificate information on the device and how to incorporate it into the template.
in	*ca_public_key*	The ECC P256 public key of the certificate authority that signed this certificate. Formatted as the 32 byte X and Y integers concatenated together (64 bytes total). Set to NULL if the authority key id is not needed, set properly in the cert_def template, or stored on the device as specifed in the cert_def cert_elements.
out	*cert*	Buffer to received the certificate.
in,out	*cert_size*	As input, the size of the cert buffer in bytes. As output, the size of the certificate returned in cert in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.53 atcacert_read_device_loc()

```
int atcacert_read_device_loc (
            const atcacert_device_loc_t * device_loc,
            uint8_t * data )
```

Read the data from a device location.

Parameters

in	*device_loc*	Device location to read data from.
out	*data*	Data read is returned here.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.54 atcacert_set_auth_key_id()

```
int atcacert_set_auth_key_id (
            const atcacert_def_t * cert_def,
            uint8_t * cert,
            size_t cert_size,
            const uint8_t auth_public_key[64] )
```

Sets the authority key ID in a certificate. Note that this takes the actual public key creates a key ID from it.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in,out	*cert*	Certificate to update.
in	*cert_size*	Size of the certificate (cert) in bytes.
in	*auth_public_key*	Authority public key as X and Y integers concatenated together. 64 bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.55 atcacert_set_auth_key_id_raw()

```
int atcacert_set_auth_key_id_raw (
            const atcacert_def_t * cert_def,
            uint8_t * cert,
            size_t cert_size,
            const uint8_t * auth_key_id )
```

Sets the authority key ID in a certificate.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in,out	*cert*	Certificate to update.
in	*cert_size*	Size of the certificate (cert) in bytes.
in	*auth_key_id*	Authority key ID. Same size as defined in the cert_def.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.56 atcacert_set_cert_element()

```
int atcacert_set_cert_element (
            const atcacert_def_t * cert_def,
            const atcacert_cert_loc_t * cert_loc,
            uint8_t * cert,
            size_t cert_size,
            const uint8_t * data,
            size_t data_size )
```

Sets an element in a certificate. The data_size must match the size in cert_loc.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in	*cert_loc*	Certificate location for this element.
in,out	*cert*	Certificate to update.
in	*cert_size*	Size of the certificate (cert) in bytes.
in	*data*	Element data to insert into the certificate. Buffer must contain cert_loc.count bytes to be copied into the certificate.
in	*data_size*	Size of the data in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.57 atcacert_set_cert_sn()

```
int atcacert_set_cert_sn (
            const atcacert_def_t * cert_def,
            uint8_t * cert,
            size_t * cert_size,
            size_t max_cert_size,
            const uint8_t * cert_sn,
            size_t cert_sn_size )
```

Sets the certificate serial number in a certificate.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in,out	*cert*	Certificate to update.
in,out	*cert_size*	Size of the certificate (cert) in bytes.
in	*max_cert_size*	Maximum size of the cert buffer.
in	*cert_sn*	Certificate serial number.
in	*cert_sn_size*	Size of the certificate serial number in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.58 atcacert_set_comp_cert()

```
int atcacert_set_comp_cert (
            const atcacert_def_t * cert_def,
            uint8_t * cert,
            size_t * cert_size,
```

```
size_t max_cert_size,
const uint8_t comp_cert[72] )
```

Sets the signature, issue date, expire date, and signer ID found in the compressed certificate. This also checks fields common between the cert_def and the compressed certificate to make sure they match.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in,out	*cert*	Certificate to update.
in,out	*cert_size*	As input, size of the certificate (cert) in bytes. As output, the new size of the certificate.
in	*max_cert_size*	Maximum size of the cert buffer.
in	*comp_cert*	Compressed certificate. 72 bytes.

Returns

ATCACERT_E_SUCCESS on success. ATCACERT_E_WRONG_CERT_DEF if the template ID, chain ID, and/or SN source don't match between the cert_def and the compressed certificate.

13.5.5.59 atcacert_set_expire_date()

```
int atcacert_set_expire_date (
            const atcacert_def_t * cert_def,
            uint8_t * cert,
            size_t cert_size,
            const atcacert_tm_utc_t * timestamp )
```

Sets the expire date (notAfter) in a certificate. Will be formatted according to the date format specified in the certificate definition.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in,out	*cert*	Certificate to update.
in	*cert_size*	Size of the certificate (cert) in bytes.
in	*timestamp*	Expire date.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.60 atcacert_set_issue_date()

```
int atcacert_set_issue_date (
            const atcacert_def_t * cert_def,
```

```
            uint8_t * cert,
            size_t cert_size,
            const atcacert_tm_utc_t * timestamp )
```

Sets the issue date (notBefore) in a certificate. Will be formatted according to the date format specified in the certificate definition.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in,out	*cert*	Certificate to update.
in	*cert_size*	Size of the certificate (cert) in bytes.
in	*timestamp*	Issue date.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.61 atcacert_set_signature()

```
int atcacert_set_signature (
            const atcacert_def_t * cert_def,
            uint8_t * cert,
            size_t * cert_size,
            size_t max_cert_size,
            const uint8_t signature[64] )
```

Sets the signature in a certificate. This may alter the size of the X.509 certificates.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in,out	*cert*	Certificate to update.
in,out	*cert_size*	As input, size of the certificate (cert) in bytes. As output, the new size of the certificate.
in	*max_cert_size*	Maximum size of the cert buffer.
in	*signature*	Signature as R and S integers concatenated together. 64 bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.62 atcacert_set_signer_id()

```
int atcacert_set_signer_id (
            const atcacert_def_t * cert_def,
```

```
            uint8_t * cert,
            size_t cert_size,
            const uint8_t signer_id[2] )
```

Sets the signer ID in a certificate. Will be formatted as 4 upper-case hex digits.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in,out	*cert*	Certificate to update.
in	*cert_size*	Size of the certificate (cert) in bytes.
in	*signer_id*	Signer ID.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.63 atcacert_set_subj_public_key()

```
int atcacert_set_subj_public_key (
            const atcacert_def_t * cert_def,
            uint8_t * cert,
            size_t cert_size,
            const uint8_t subj_public_key[64] )
```

Sets the subject public key and subject key ID in a certificate.

Parameters

in	*cert_def*	Certificate definition for the certificate.
in,out	*cert*	Certificate to update.
in	*cert_size*	Size of the certificate (cert) in bytes.
in	*subj_public_key*	Subject public key as X and Y integers concatenated together. 64 bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.64 atcacert_transform_data()

```
int atcacert_transform_data (
            atcacert_transform_t transform,
            const uint8_t * data,
            size_t data_size,
```

```
uint8_t * destination,
size_t * destination_size )
```

Apply the specified transform to the specified data.

Parameters

in	*transform*	Transform to be performed.
in	*data*	Input data to be transformed.
in	*data_size*	Size of the input data in bytes.
out	*destination*	Destination buffer to hold the transformed data.
in,out	*destination_size*	As input, the size of the destination buffer. As output the size of the transformed data.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.5.65 atcacert_verify_cert_hw()

```
int atcacert_verify_cert_hw (
            const atcacert_def_t * cert_def,
            const uint8_t * cert,
            size_t cert_size,
            const uint8_t ca_public_key[64] )
```

Verify a certificate against its certificate authority's public key using the host's ATECC device for crypto functions.

Parameters

in	*cert_def*	Certificate definition describing how to extract the TBS and signature components from the certificate specified.
in	*cert*	Certificate to verify.
in	*cert_size*	Size of the certificate (cert) in bytes.
in	*ca_public_key*	The ECC P256 public key of the certificate authority that signed this certificate. Formatted as the 32 byte X and Y integers concatenated together (64 bytes total).

Returns

ATCACERT_E_SUCCESS if the verify succeeds, ATCACERT_VERIFY_FAILED or ATCA_EXECUTION_↩ ERROR if it fails to verify. ATCA_EXECUTION_ERROR may occur when the public key is invalid and doesn't fall on the P256 curve.

13.5.5.66 atcacert_verify_cert_sw()

```
int atcacert_verify_cert_sw (
            const atcacert_def_t * cert_def,
            const uint8_t * cert,
            size_t cert_size,
            const uint8_t ca_public_key[64] )
```

Verify a certificate against its certificate authority's public key using software crypto functions.The function is currently not implemented.

Parameters

in	*cert_def*	Certificate definition describing how to extract the TBS and signature components from the certificate specified.
in	*cert*	Certificate to verify.
in	*cert_size*	Size of the certificate (cert) in bytes.
in	*ca_public_key*	The ECC P256 public key of the certificate authority that signed this certificate. Formatted as the 32 byte X and Y integers concatenated together (64 bytes total).

Returns

ATCA_UNIMPLEMENTED , as the function is currently not implemented.

13.5.5.67 atcacert_verify_response_hw()

```
int atcacert_verify_response_hw (
            const uint8_t device_public_key[64],
            const uint8_t challenge[32],
            const uint8_t response[64] )
```

Verify a client's response to a challenge using the host's ATECC device for crypto functions.

The challenge-response protocol is an ECDSA Sign and Verify. This performs an ECDSA verify on the response returned by the client, verifying the client has the private key counter-part to the public key returned in its certificate.

Parameters

in	*device_public_key*	Device public key as read from its certificate. Formatted as the X and Y integers concatenated together. 64 bytes.
in	*challenge*	Challenge that was sent to the client. 32 bytes.
in	*response*	Response returned from the client to be verified. 64 bytes.

Returns

ATCACERT_E_SUCCESS if the verify succeeds, ATCACERT_VERIFY_FAILED or ATCA_EXECUTION_↩ ERROR if it fails to verify. ATCA_EXECUTION_ERROR may occur when the public key is invalid and doesn't fall on the P256 curve.

13.5.5.68 atcacert_verify_response_sw()

```
int atcacert_verify_response_sw (
            const uint8_t device_public_key[64],
            const uint8_t challenge[32],
            const uint8_t response[64] )
```

Verify a client's response to a challenge using software crypto functions.The function is currently not implemented.

The challenge-response protocol is an ECDSA Sign and Verify. This performs an ECDSA verify on the response returned by the client, verifying the client has the private key counter-part to the public key returned in its certificate.

Parameters

in	*device_public_key*	Device public key as read from its certificate. Formatted as the X and Y integers concatenated together. 64 bytes.
in	*challenge*	Challenge that was sent to the client. 32 bytes.
in	*response*	Response returned from the client to be verified. 64 bytes.

Returns

ATCA_UNIMPLEMENTED , as the function is currently not implemented.

13.5.5.69 atcacert_write_cert()

```
int atcacert_write_cert (
            const atcacert_def_t * cert_def,
            const uint8_t * cert,
            size_t cert_size )
```

Take a full certificate and write it to the ATECC508A device according to the certificate definition.

Parameters

in	*cert_def*	Certificate definition describing where the dynamic certificate information is and how to store it on the device.
in	*cert*	Full certificate to be stored.
in	*cert_size*	Size of the full certificate in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.5.6 Variable Documentation

13.5.6.1 ATCACERT_DATE_FORMAT_SIZES

```
const size_t ATCACERT_DATE_FORMAT_SIZES[ATCACERT_DATE_FORMAT_SIZES_COUNT]
```

13.6 Basic Crypto API methods (atcab_)

These methods provide the most convenient, simple API to CryptoAuth chips.

Data Structures

- struct atca_aes_cbc_ctx
- struct atca_aes_cmac_ctx
- struct atca_aes_ctr_ctx
- struct atca_sha256_ctx

Macros

- #define BLOCK_NUMBER(a) (a / 32)
- #define WORD_OFFSET(a) ((a % 32) / 4)
- #define ATCA_AES_GCM_IV_STD_LENGTH 12

Typedefs

- typedef struct atca_aes_cbc_ctx atca_aes_cbc_ctx_t
- typedef struct atca_aes_cmac_ctx atca_aes_cmac_ctx_t
- typedef struct atca_aes_ctr_ctx atca_aes_ctr_ctx_t
- typedef struct atca_sha256_ctx atca_sha256_ctx_t
- typedef atca_sha256_ctx_t atca_hmac_sha256_ctx_t

Functions

- ATCA_STATUS atcab_version (char ∗ver_str)

  basic API methods are all prefixed with atcab_ (CryptoAuthLib Basic) the fundamental premise of the basic API is it is based on a single interface instance and that instance is global, so all basic API commands assume that one global device is the one to operate on.
- ATCA_STATUS atcab_init (ATCAIfaceCfg ∗cfg)

  Creates a global ATCADevice object used by Basic API.
- ATCA_STATUS atcab_init_device (ATCADevice ca_device)

  Initialize the global ATCADevice object to point to one of your choosing for use with all the atcab_ basic API.
- ATCA_STATUS atcab_release (void)

  release (free) the global ATCADevice instance. This must be called in order to release or free up the interface.
- ATCADevice atcab_get_device (void)

  Get the global device object.
- ATCADeviceType atcab_get_device_type (void)

  Get the current device type.
- ATCA_STATUS _atcab_exit (void)

  common cleanup code which idles the device after any operation
- ATCA_STATUS atcab_wakeup (void)

  wakeup the CryptoAuth device
- ATCA_STATUS atcab_idle (void)

  idle the CryptoAuth device
- ATCA_STATUS atcab_sleep (void)

  invoke sleep on the CryptoAuth device

- ATCA_STATUS atcab_cfg_discover (ATCAIfaceCfg cfg_array[ ], int max)

  auto discovery of crypto auth devices
- ATCA_STATUS atcab_get_addr (uint8_t zone, uint16_t slot, uint8_t block, uint8_t offset, uint16_t ∗addr)

  Compute the address given the zone, slot, block, and offset.
- ATCA_STATUS atcab_get_zone_size (uint8_t zone, uint16_t slot, size_t ∗size)

  Gets the size of the specified zone in bytes.
- ATCA_STATUS atcab_aes (uint8_t mode, uint16_t key_id, const uint8_t ∗aes_in, uint8_t ∗aes_out)

  Compute the AES-128 encrypt, decrypt, or GFM calculation.
- ATCA_STATUS atcab_aes_encrypt (uint16_t key_id, uint8_t key_block, const uint8_t ∗plaintext, uint8_t ∗ciphertext)

  Perform an AES-128 encrypt operation with a key in the device.
- ATCA_STATUS atcab_aes_decrypt (uint16_t key_id, uint8_t key_block, const uint8_t ∗ciphertext, uint8_t ∗plaintext)

  Perform an AES-128 decrypt operation with a key in the device.
- ATCA_STATUS atcab_aes_gfm (const uint8_t ∗h, const uint8_t ∗input, uint8_t ∗output)

  Perform a Galois Field Multiply (GFM) operation.
- ATCA_STATUS atcab_aes_cbc_init (atca_aes_cbc_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, const uint8_t ∗iv)

  Initialize context for AES CBC operation.
- ATCA_STATUS atcab_aes_cbc_encrypt_block (atca_aes_cbc_ctx_t ∗ctx, const uint8_t ∗plaintext, uint8_t ∗ciphertext)

  Encrypt a block of data using CBC mode and a key within the ATECC608A. atcab_aes_cbc_init() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_cbc_decrypt_block (atca_aes_cbc_ctx_t ∗ctx, const uint8_t ∗ciphertext, uint8_t ∗plaintext)

  Decrypt a block of data using CBC mode and a key within the ATECC608A. atcab_aes_cbc_init() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_cmac_init (atca_aes_cmac_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block)

  Initialize a CMAC calculation using an AES-128 key in the ATECC608A.
- ATCA_STATUS atcab_aes_cmac_update (atca_aes_cmac_ctx_t ∗ctx, const uint8_t ∗data, uint32_t data_size)

  Add data to an initialized CMAC calculation.
- ATCA_STATUS atcab_aes_cmac_finish (atca_aes_cmac_ctx_t ∗ctx, uint8_t ∗cmac, uint32_t cmac_size)

  Finish a CMAC operation returning the CMAC value.
- ATCA_STATUS atcab_aes_ctr_init (atca_aes_ctr_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, uint8_t counter_size, const uint8_t ∗iv)

  Initialize context for AES CTR operation with an existing IV, which is common when start a decrypt operation.
- ATCA_STATUS atcab_aes_ctr_init_rand (atca_aes_ctr_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, uint8_t counter_size, uint8_t ∗iv)

  Initialize context for AES CTR operation with a random nonce and counter set to 0 as the IV, which is common when starting an encrypt operation.
- ATCA_STATUS atcab_aes_ctr_block (atca_aes_ctr_ctx_t ∗ctx, const uint8_t ∗input, uint8_t ∗output)

  Process a block of data using CTR mode and a key within the ATECC608A device. atcab_aes_ctr_init() or atcab_aes_ctr_init_rand() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_ctr_encrypt_block (atca_aes_ctr_ctx_t ∗ctx, const uint8_t ∗plaintext, uint8_t ∗ciphertext)

  Encrypt a block of data using CTR mode and a key within the ATECC608A device. atcab_aes_ctr_init() or atcab_aes_ctr_init_rand() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_ctr_decrypt_block (atca_aes_ctr_ctx_t ∗ctx, const uint8_t ∗ciphertext, uint8_t ∗plaintext)

  Decrypt a block of data using CTR mode and a key within the ATECC608A device. atcab_aes_ctr_init() or atcab_aes_ctr_init_rand() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_ctr_increment (atca_aes_ctr_ctx_t ∗ctx)

Increments AES CTR counter value.

- ATCA_STATUS atcab_checkmac (uint8_t mode, uint16_t key_id, const uint8_t *challenge, const uint8_t *response, const uint8_t *other_data)

  Compares a MAC response with input values.
- ATCA_STATUS atcab_counter (uint8_t mode, uint16_t counter_id, uint32_t *counter_value)

  Compute the Counter functions.
- ATCA_STATUS atcab_counter_increment (uint16_t counter_id, uint32_t *counter_value)

  Increments one of the device's monotonic counters.
- ATCA_STATUS atcab_counter_read (uint16_t counter_id, uint32_t *counter_value)

  Read one of the device's monotonic counters.
- ATCA_STATUS atcab_derivekey (uint8_t mode, uint16_t key_id, const uint8_t *mac)

  Executes the DeviveKey command for deriving a new key from a nonce (TempKey) and an existing key.
- ATCA_STATUS atcab_ecdh_base (uint8_t mode, uint16_t key_id, const uint8_t *public_key, uint8_t *pms, uint8_t *out_nonce)

  Base function for generating premaster secret key using ECDH.
- ATCA_STATUS atcab_ecdh (uint16_t key_id, const uint8_t *public_key, uint8_t *pms)

  ECDH command with a private key in a slot and the premaster secret is returned in the clear.
- ATCA_STATUS atcab_ecdh_enc (uint16_t key_id, const uint8_t *public_key, uint8_t *pms, const uint8_t *read_key, uint16_t read_key_id)

  ECDH command with a private key in a slot and the premaster secret is read from the next slot.
- ATCA_STATUS atcab_ecdh_ioenc (uint16_t key_id, const uint8_t *public_key, uint8_t *pms, const uint8_t *io_key)

  ECDH command with a private key in a slot and the premaster secret is returned encrypted using the IO protection key.
- ATCA_STATUS atcab_ecdh_tempkey (const uint8_t *public_key, uint8_t *pms)

  ECDH command with a private key in TempKey and the premaster secret is returned in the clear.
- ATCA_STATUS atcab_ecdh_tempkey_ioenc (const uint8_t *public_key, uint8_t *pms, const uint8_t *io_key)

  ECDH command with a private key in TempKey and the premaster secret is returned encrypted using the IO protection key.
- ATCA_STATUS atcab_gendig (uint8_t zone, uint16_t key_id, const uint8_t *other_data, uint8_t other_data_size)

  Issues a GenDig command, which performs a SHA256 hash on the source data indicated by zone with the contents of TempKey. See the CryptoAuth datasheet for your chip to see what the values of zone correspond to.
- ATCA_STATUS atcab_genkey_base (uint8_t mode, uint16_t key_id, const uint8_t *other_data, uint8_t *public_key)

  Issues GenKey command, which can generate a private key, compute a public key, nd/or compute a digest of a public key.
- ATCA_STATUS atcab_genkey (uint16_t key_id, uint8_t *public_key)

  Issues GenKey command, which generates a new random private key in slot and returns the public key.
- ATCA_STATUS atcab_get_pubkey (uint16_t key_id, uint8_t *public_key)

  Uses GenKey command to calculate the public key from an existing private key in a slot.
- ATCA_STATUS atcab_hmac (uint8_t mode, uint16_t key_id, uint8_t *digest)

  Issues a HMAC command, which computes an HMAC/SHA-256 digest of a key stored in the device, a challenge, and other information on the device.
- ATCA_STATUS atcab_info_base (uint8_t mode, uint16_t param2, uint8_t *out_data)

  Issues an Info command, which return internal device information and can control GPIO and the persistent latch.
- ATCA_STATUS atcab_info (uint8_t *revision)

  Use the Info command to get the device revision (DevRev).
- ATCA_STATUS atcab_info_set_latch (bool state)

  Use the Info command to set the persistent latch state for an ATECC608A device.
- ATCA_STATUS atcab_info_get_latch (bool *state)

  Use the Info command to get the persistent latch current state for an ATECC608A device.

- ATCA_STATUS atcab_kdf (uint8_t mode, uint16_t key_id, const uint32_t details, const uint8_t ∗message, uint8_t ∗out_data, uint8_t ∗out_nonce)

    Executes the KDF command, which derives a new key in PRF, AES, or HKDF modes.
- ATCA_STATUS atcab_lock (uint8_t mode, uint16_t summary_crc)

    The Lock command prevents future modifications of the Configuration and/or Data and OTP zones. If the device is so configured, then this command can be used to lock individual data slots. This command fails if the designated area is already locked.
- ATCA_STATUS atcab_lock_config_zone (void)

    Unconditionally (no CRC required) lock the config zone.
- ATCA_STATUS atcab_lock_config_zone_crc (uint16_t summary_crc)

    Lock the config zone with summary CRC.
- ATCA_STATUS atcab_lock_data_zone (void)

    Unconditionally (no CRC required) lock the data zone (slots and OTP).
- ATCA_STATUS atcab_lock_data_zone_crc (uint16_t summary_crc)

    Lock the data zone (slots and OTP) with summary CRC.
- ATCA_STATUS atcab_lock_data_slot (uint16_t slot)

    Lock an individual slot in the data zone on an ATECC device. Not available for ATSHA devices. Slot must be configured to be slot lockable (KeyConfig.Lockable=1).
- ATCA_STATUS atcab_mac (uint8_t mode, uint16_t key_id, const uint8_t ∗challenge, uint8_t ∗digest)

    Executes MAC command, which computes a SHA-256 digest of a key stored in the device, a challenge, and other information on the device.
- ATCA_STATUS atcab_nonce_base (uint8_t mode, uint16_t zero, const uint8_t ∗num_in, uint8_t ∗rand_out)

    Executes Nonce command, which loads a random or fixed nonce/data into the device for use by subsequent commands.
- ATCA_STATUS atcab_nonce (const uint8_t ∗num_in)

    Execute a Nonce command in pass-through mode to initialize TempKey to a specified value.
- ATCA_STATUS atcab_nonce_load (uint8_t target, const uint8_t ∗num_in, uint16_t num_in_size)

    Execute a Nonce command in pass-through mode to load one of the device's internal buffers with a fixed value.
- ATCA_STATUS atcab_nonce_rand (const uint8_t ∗num_in, uint8_t ∗rand_out)

    Execute a Nonce command to generate a random nonce combining a host nonce (num_in) and a device random number.
- ATCA_STATUS atcab_challenge (const uint8_t ∗num_in)

    Execute a Nonce command in pass-through mode to initialize TempKey to a specified value.
- ATCA_STATUS atcab_challenge_seed_update (const uint8_t ∗num_in, uint8_t ∗rand_out)

    Execute a Nonce command to generate a random challenge combining a host nonce (num_in) and a device random number.
- ATCA_STATUS atcab_priv_write (uint16_t key_id, const uint8_t priv_key[36], uint16_t write_key_id, const uint8_t write_key[32])

    Executes PrivWrite command, to write externally generated ECC private keys into the device.
- ATCA_STATUS atcab_random (uint8_t ∗rand_out)

    Executes Random command, which generates a 32 byte random number from the CryptoAuth device.
- ATCA_STATUS atcab_read_zone (uint8_t zone, uint16_t slot, uint8_t block, uint8_t offset, uint8_t ∗data, uint8_t len)

    Executes Read command, which reads either 4 or 32 bytes of data from a given slot, configuration zone, or the OTP zone.
- ATCA_STATUS atcab_is_locked (uint8_t zone, bool ∗is_locked)

    Executes Read command, which reads the configuration zone to see if the specified zone is locked.
- ATCA_STATUS atcab_is_slot_locked (uint16_t slot, bool ∗is_locked)

    Executes Read command, which reads the configuration zone to see if the specified slot is locked.
- ATCA_STATUS atcab_read_bytes_zone (uint8_t zone, uint16_t slot, size_t offset, uint8_t ∗data, size_↩ t length)

    Used to read an arbitrary number of bytes from any zone configured for clear reads.
- ATCA_STATUS atcab_read_serial_number (uint8_t ∗serial_number)

Executes Read command, which reads the 9 byte serial number of the device from the config zone.

- ATCA_STATUS atcab_read_pubkey (uint16_t slot, uint8_t *public_key)

  Executes Read command to read an ECC P256 public key from a slot configured for clear reads.
- ATCA_STATUS atcab_read_sig (uint16_t slot, uint8_t *sig)

  Executes Read command to read a 64 byte ECDSA P256 signature from a slot configured for clear reads.
- ATCA_STATUS atcab_read_config_zone (uint8_t *config_data)

  Executes Read command to read the complete device configuration zone.
- ATCA_STATUS atcab_cmp_config_zone (uint8_t *config_data, bool *same_config)

  Compares a specified configuration zone with the configuration zone currently on the device.
- ATCA_STATUS atcab_read_enc (uint16_t key_id, uint8_t block, uint8_t *data, const uint8_t *enc_key, const uint16_t enc_key_id)

  Executes Read command on a slot configured for encrypted reads and decrypts the data to return it as plaintext.
- ATCA_STATUS atcab_secureboot (uint8_t mode, uint16_t param2, const uint8_t *digest, const uint8_t *signature, uint8_t *mac)

  Executes Secure Boot command, which provides support for secure boot of an external MCU or MPU.
- ATCA_STATUS atcab_secureboot_mac (uint8_t mode, const uint8_t *digest, const uint8_t *signature, const uint8_t *num_in, const uint8_t *io_key, bool *is_verified)

  Executes Secure Boot command with encrypted digest and validated MAC response using the IO protection key.
- ATCA_STATUS atcab_selftest (uint8_t mode, uint16_t param2, uint8_t *result)

  Executes the SelfTest command, which performs a test of one or more of the cryptographic engines within the ATECC608A chip.
- ATCA_STATUS atcab_sha_base (uint8_t mode, uint16_t length, const uint8_t *data_in, uint8_t *data_out, uint16_t *data_out_size)

  Executes SHA command, which computes a SHA-256 or HMAC/SHA-256 digest for general purpose use by the host system.
- ATCA_STATUS atcab_sha_start (void)

  Executes SHA command to initialize SHA-256 calculation engine.
- ATCA_STATUS atcab_sha_update (const uint8_t *message)

  Executes SHA command to add 64 bytes of message data to the current context.
- ATCA_STATUS atcab_sha_end (uint8_t *digest, uint16_t length, const uint8_t *message)

  Executes SHA command to complete SHA-256 or HMAC/SHA-256 operation.
- ATCA_STATUS atcab_sha_read_context (uint8_t *context, uint16_t *context_size)

  Executes SHA command to read the SHA-256 context back. Only for ATECC608A with SHA-256 contexts. HMAC not supported.
- ATCA_STATUS atcab_sha_write_context (const uint8_t *context, uint16_t context_size)

  Executes SHA command to write (restore) a SHA-256 context into the the device. Only supported for ATECC608A with SHA-256 contexts.
- ATCA_STATUS atcab_sha (uint16_t length, const uint8_t *message, uint8_t *digest)

  Use the SHA command to compute a SHA-256 digest.
- ATCA_STATUS atcab_hw_sha2_256 (const uint8_t *data, size_t data_size, uint8_t *digest)

  Use the SHA command to compute a SHA-256 digest.
- ATCA_STATUS atcab_hw_sha2_256_init (atca_sha256_ctx_t *ctx)

  Initialize a SHA context for performing a hardware SHA-256 operation on a device. Note that only one SHA operation can be run at a time.
- ATCA_STATUS atcab_hw_sha2_256_update (atca_sha256_ctx_t *ctx, const uint8_t *data, size_t data_size)

  Add message data to a SHA context for performing a hardware SHA-256 operation on a device.
- ATCA_STATUS atcab_hw_sha2_256_finish (atca_sha256_ctx_t *ctx, uint8_t *digest)

  Finish SHA-256 digest for a SHA context for performing a hardware SHA-256 operation on a device.
- ATCA_STATUS atcab_sha_hmac_init (atca_hmac_sha256_ctx_t *ctx, uint16_t key_slot)

  Executes SHA command to start an HMAC/SHA-256 operation.
- ATCA_STATUS atcab_sha_hmac_update (atca_hmac_sha256_ctx_t *ctx, const uint8_t *data, size_t data_size)

Executes SHA command to add an arbitrary amount of message data to a HMAC/SHA-256 operation.

- ATCA_STATUS atcab_sha_hmac_finish (atca_hmac_sha256_ctx_t ∗ctx, uint8_t ∗digest, uint8_t target)

  Executes SHA command to complete a HMAC/SHA-256 operation.
- ATCA_STATUS atcab_sha_hmac (const uint8_t ∗data, size_t data_size, uint16_t key_slot, uint8_t ∗digest, uint8_t target)

  Use the SHA command to compute an HMAC/SHA-256 operation.
- ATCA_STATUS atcab_sign_base (uint8_t mode, uint16_t key_id, uint8_t ∗signature)

  Executes the Sign command, which generates a signature using the ECDSA algorithm.
- ATCA_STATUS atcab_sign (uint16_t key_id, const uint8_t ∗msg, uint8_t ∗signature)

  Executes Sign command, to sign a 32-byte external message using the private key in the specified slot. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608A device or TempKey for other devices.
- ATCA_STATUS atcab_sign_internal (uint16_t key_id, bool is_invalidate, bool is_full_sn, uint8_t ∗signature)

  Executes Sign command to sign an internally generated message.
- ATCA_STATUS atcab_updateextra (uint8_t mode, uint16_t new_value)

  Executes UpdateExtra command to update the values of the two extra bytes within the Configuration zone (bytes 84 and 85).
- ATCA_STATUS atcab_verify (uint8_t mode, uint16_t key_id, const uint8_t ∗signature, const uint8_t ∗public_key, const uint8_t ∗other_data, uint8_t ∗mac)

  Executes the Verify command, which takes an ECDSA [R,S] signature and verifies that it is correctly generated from a given message and public key. In all cases, the signature is an input to the command.
- ATCA_STATUS atcab_verify_extern (const uint8_t ∗message, const uint8_t ∗signature, const uint8_t ∗public_key, bool ∗is_verified)

  Executes the Verify command, which verifies a signature (ECDSA verify operation) with all components (message, signature, and public key) supplied. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608A device or TempKey for other devices.
- ATCA_STATUS atcab_verify_extern_mac (const uint8_t ∗message, const uint8_t ∗signature, const uint8_t ∗public_key, const uint8_t ∗num_in, const uint8_t ∗io_key, bool ∗is_verified)

  Executes the Verify command with verification MAC, which verifies a signature (ECDSA verify operation) with all components (message, signature, and public key) supplied. This function is only available on the ATECC608A.
- ATCA_STATUS atcab_verify_stored (const uint8_t ∗message, const uint8_t ∗signature, uint16_t key_id, bool ∗is_verified)

  Executes the Verify command, which verifies a signature (ECDSA verify operation) with a public key stored in the device. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608A device or TempKey for other devices.
- ATCA_STATUS atcab_verify_stored_mac (const uint8_t ∗message, const uint8_t ∗signature, uint16_t key_id, const uint8_t ∗num_in, const uint8_t ∗io_key, bool ∗is_verified)

  Executes the Verify command with verification MAC, which verifies a signature (ECDSA verify operation) with a public key stored in the device. This function is only available on the ATECC608A.
- ATCA_STATUS atcab_verify_validate (uint16_t key_id, const uint8_t ∗signature, const uint8_t ∗other_data, bool ∗is_verified)

  Executes the Verify command in Validate mode to validate a public key stored in a slot.
- ATCA_STATUS atcab_verify_invalidate (uint16_t key_id, const uint8_t ∗signature, const uint8_t ∗other_data, bool ∗is_verified)

  Executes the Verify command in Invalidate mode which invalidates a previously validated public key stored in a slot.
- ATCA_STATUS atcab_write (uint8_t zone, uint16_t address, const uint8_t ∗value, const uint8_t ∗mac)

  Executes the Write command, which writes either one four byte word or a 32-byte block to one of the EEPROM zones on the device. Depending upon the value of the WriteConfig byte for this slot, the data may be required to be encrypted by the system prior to being sent to the device. This command cannot be used to write slots configured as ECC private keys.
- ATCA_STATUS atcab_write_zone (uint8_t zone, uint16_t slot, uint8_t block, uint8_t offset, const uint8_t ∗data, uint8_t len)

  Executes the Write command, which writes either 4 or 32 bytes of data into a device zone.
- ATCA_STATUS atcab_write_bytes_zone (uint8_t zone, uint16_t slot, size_t offset_bytes, const uint8_t ∗data, size_t length)

Executes the Write command, which writes data into the configuration, otp, or data zones with a given byte offset and length. Offset and length must be multiples of a word (4 bytes).

- ATCA_STATUS atcab_write_pubkey (uint16_t slot, const uint8_t ∗public_key)

  Uses the write command to write a public key to a slot in the proper format.
- ATCA_STATUS atcab_write_config_zone (const uint8_t ∗config_data)

  Executes the Write command, which writes the configuration zone.
- ATCA_STATUS atcab_write_enc (uint16_t key_id, uint8_t block, const uint8_t ∗data, const uint8_t ∗enc_key, const uint16_t enc_key_id)

  Executes the Write command, which performs an encrypted write of a 32 byte block into given slot.
- ATCA_STATUS atcab_write_config_counter (uint16_t counter_id, uint32_t counter_value)

  Initialize one of the monotonic counters in device with a specific value.

Variables

- ATCADevice _gDevice

- const char ∗ atca_basic_aes_gcm_version = "1.0"
- ATCA_STATUS atcab_aes_gcm_init (atca_aes_gcm_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, const uint8_t ∗iv, size_t iv_size)

  Initialize context for AES GCM operation with an existing IV, which is common when starting a decrypt operation.
- ATCA_STATUS atcab_aes_gcm_init_rand (atca_aes_gcm_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, size_t rand_size, const uint8_t ∗free_field, size_t free_field_size, uint8_t ∗iv)

  Initialize context for AES GCM operation with a IV composed of a random and optional fixed(free) field, which is common when starting an encrypt operation.
- ATCA_STATUS atcab_aes_gcm_aad_update (atca_aes_gcm_ctx_t ∗ctx, const uint8_t ∗aad, uint32_t aad←_size)

  Process Additional Authenticated Data (AAD) using GCM mode and a key within the ATECC608A device.
- ATCA_STATUS atcab_aes_gcm_encrypt_update (atca_aes_gcm_ctx_t ∗ctx, const uint8_t ∗plaintext, uint32_t plaintext_size, uint8_t ∗ciphertext)

  Encrypt data using GCM mode and a key within the ATECC608A device. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_gcm_encrypt_finish (atca_aes_gcm_ctx_t ∗ctx, uint8_t ∗tag, size_t tag_size)

  Complete a GCM encrypt operation returning the authentication tag.
- ATCA_STATUS atcab_aes_gcm_decrypt_update (atca_aes_gcm_ctx_t ∗ctx, const uint8_t ∗ciphertext, uint32_t ciphertext_size, uint8_t ∗plaintext)

  Decrypt data using GCM mode and a key within the ATECC608A device. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_gcm_decrypt_finish (atca_aes_gcm_ctx_t ∗ctx, const uint8_t ∗tag, size_t tag←_size, bool ∗is_verified)

  Complete a GCM decrypt operation verifying the authentication tag.

- const char ∗ atca_basic_aes_gcm_version

- ATCA_STATUS atcab_printbin (uint8_t ∗binary, size_t bin_len, bool add_space)

13.6.1 Detailed Description

These methods provide the most convenient, simple API to CryptoAuth chips.

13.6.2 Macro Definition Documentation

13.6.2.1 ATCA_AES_GCM_IV_STD_LENGTH

```
#define ATCA_AES_GCM_IV_STD_LENGTH 12
```

13.6.2.2 BLOCK_NUMBER

```
#define BLOCK_NUMBER(
              a ) (a / 32)
```

13.6.2.3 WORD_OFFSET

```
#define WORD_OFFSET(
              a ) ((a % 32) / 4)
```

13.6.3 Typedef Documentation

13.6.3.1 atca_aes_cbc_ctx_t

```
typedef struct atca_aes_cbc_ctx atca_aes_cbc_ctx_t
```

13.6.3.2 atca_aes_cmac_ctx_t

```
typedef struct atca_aes_cmac_ctx atca_aes_cmac_ctx_t
```

13.6.3.3 atca_aes_ctr_ctx_t

```
typedef struct atca_aes_ctr_ctx atca_aes_ctr_ctx_t
```

13.6.3.4 atca_hmac_sha256_ctx_t

```
typedef atca_sha256_ctx_t atca_hmac_sha256_ctx_t
```

13.6.3.5 atca_sha256_ctx_t

```
typedef struct atca_sha256_ctx atca_sha256_ctx_t
```

13.6.4 Function Documentation

13.6.4.1 _atcab_exit()

```
ATCA_STATUS _atcab_exit (
            void )
```

common cleanup code which idles the device after any operation

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.2 atcab_aes()

```
ATCA_STATUS atcab_aes (
            uint8_t mode,
            uint16_t key_id,
            const uint8_t * aes_in,
            uint8_t * aes_out )
```

Compute the AES-128 encrypt, decrypt, or GFM calculation.

Parameters

in	*mode*	The mode for the AES command.
in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*aes_in*	Input data to the AES command (16 bytes).
out	*aes_out*	Output data from the AES command is returned here (16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.3 atcab_aes_cbc_decrypt_block()

```
ATCA_STATUS atcab_aes_cbc_decrypt_block (
            atca_aes_cbc_ctx_t * ctx,
            const uint8_t * ciphertext,
            uint8_t * plaintext )
```

Decrypt a block of data using CBC mode and a key within the ATECC608A. atcab_aes_cbc_init() should be called before the first use of this function.

Parameters

in	*ctx*	AES CBC context.
in	*ciphertext*	Ciphertext to be decrypted (16 bytes).
out	*plaintext*	Decrypted data is returned here (16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.4 atcab_aes_cbc_encrypt_block()

```
ATCA_STATUS atcab_aes_cbc_encrypt_block (
            atca_aes_cbc_ctx_t * ctx,
            const uint8_t * plaintext,
            uint8_t * ciphertext )
```

Encrypt a block of data using CBC mode and a key within the ATECC608A. atcab_aes_cbc_init() should be called before the first use of this function.

Parameters

in	*ctx*	AES CBC context.
in	*plaintext*	Plaintext to be encrypted (16 bytes).
out	*ciphertext*	Encrypted data is returned here (16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.5 atcab_aes_cbc_init()

```
ATCA_STATUS atcab_aes_cbc_init (
            atca_aes_cbc_ctx_t * ctx,
            uint16_t key_id,
            uint8_t key_block,
            const uint8_t * iv )
```

Initialize context for AES CBC operation.

Parameters

in	*ctx*	AES CBC context to be initialized
in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.
in	*iv*	Initialization vector (16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.6 atcab_aes_cmac_finish()

```
ATCA_STATUS atcab_aes_cmac_finish (
            atca_aes_cmac_ctx_t * ctx,
            uint8_t * cmac,
            uint32_t cmac_size )
```

Finish a CMAC operation returning the CMAC value.

Parameters

in	*ctx*	AES-128 CMAC context.
out	*cmac*	CMAC is returned here.
in	*cmac_size*	Size of CMAC requested in bytes (max 16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.7 atcab_aes_cmac_init()

```
ATCA_STATUS atcab_aes_cmac_init (
            atca_aes_cmac_ctx_t * ctx,
            uint16_t key_id,
            uint8_t key_block )
```

Initialize a CMAC calculation using an AES-128 key in the ATECC608A.

Parameters

in	*ctx*	AES-128 CMAC context.
in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.8 atcab_aes_cmac_update()

```
ATCA_STATUS atcab_aes_cmac_update (
            atca_aes_cmac_ctx_t * ctx,
            const uint8_t * data,
            uint32_t data_size )
```

Add data to an initialized CMAC calculation.

Parameters

in	*ctx*	AES-128 CMAC context.
in	*data*	Data to be added.
in	*data_size*	Size of the data to be added in bytes.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.9 atcab_aes_ctr_block()

```
ATCA_STATUS atcab_aes_ctr_block (
            atca_aes_ctr_ctx_t * ctx,
            const uint8_t * input,
            uint8_t * output )
```

Process a block of data using CTR mode and a key within the ATECC608A device. atcab_aes_ctr_init() or atcab_aes_ctr_init_rand() should be called before the first use of this function.

Parameters

in	*ctx*	AES CTR context structure.
in	*input*	Input data to be processed (16 bytes).
out	*output*	Output data is returned here (16 bytes).

Returns

ATCA_SUCCESS on success, ATCA_INVALID_SIZE on counter overflow, otherwise an error code.

13.6.4.10 atcab_aes_ctr_decrypt_block()

```
ATCA_STATUS atcab_aes_ctr_decrypt_block (
            atca_aes_ctr_ctx_t * ctx,
            const uint8_t * ciphertext,
            uint8_t * plaintext )
```

Decrypt a block of data using CTR mode and a key within the ATECC608A device. atcab_aes_ctr_init() or atcab_aes_ctr_init_rand() should be called before the first use of this function.

Parameters

in	*ctx*	AES CTR context structure.
in	*ciphertext*	Ciphertext to be decrypted (16 bytes).
out	*plaintext*	Decrypted data is returned here (16 bytes).

Returns

ATCA_SUCCESS on success, ATCA_INVALID_SIZE on counter overflow, otherwise an error code.

13.6.4.11 atcab_aes_ctr_encrypt_block()

```
ATCA_STATUS atcab_aes_ctr_encrypt_block (
            atca_aes_ctr_ctx_t * ctx,
            const uint8_t * plaintext,
            uint8_t * ciphertext )
```

Encrypt a block of data using CTR mode and a key within the ATECC608A device. atcab_aes_ctr_init() or atcab_aes_ctr_init_rand() should be called before the first use of this function.

Parameters

in	*ctx*	AES CTR context structure.
in	*plaintext*	Plaintext to be encrypted (16 bytes).
out	*ciphertext*	Encrypted data is returned here (16 bytes).

Returns

ATCA_SUCCESS on success, ATCA_INVALID_SIZE on counter overflow, otherwise an error code.

13.6.4.12 atcab_aes_ctr_increment()

```
ATCA_STATUS atcab_aes_ctr_increment (
            atca_aes_ctr_ctx_t * ctx )
```

Increments AES CTR counter value.

Parameters

in,out	*ctx*	AES CTR context

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.13 atcab_aes_ctr_init()

```
ATCA_STATUS atcab_aes_ctr_init (
            atca_aes_ctr_ctx_t * ctx,
            uint16_t key_id,
            uint8_t key_block,
            uint8_t counter_size,
            const uint8_t * iv )
```

Initialize context for AES CTR operation with an existing IV, which is common when start a decrypt operation.

The IV is a combination of nonce (left-field) and big-endian counter (right-field). The counter_size field sets the size of the counter and the remaining bytes are assumed to be the nonce.

Parameters

in	*ctx*	AES CTR context to be initialized.
in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.
in	*counter_size*	Size of counter in IV in bytes. 4 bytes is a common size.
in	*iv*	Initialization vector (concatenation of nonce and counter) 16 bytes.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.14 atcab_aes_ctr_init_rand()

```
ATCA_STATUS atcab_aes_ctr_init_rand (
            atca_aes_ctr_ctx_t * ctx,
```

```
            uint16_t key_id,
            uint8_t key_block,
            uint8_t counter_size,
            uint8_t * iv )
```

Initialize context for AES CTR operation with a random nonce and counter set to 0 as the IV, which is common when starting an encrypt operation.

The IV is a combination of nonce (left-field) and big-endian counter (right-field). The counter_size field sets the size of the counter and the remaining bytes are assumed to be the nonce.

Parameters

in	*ctx*	AES CTR context to be initialized.
in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.
in	*counter_size*	Size of counter in IV in bytes. 4 bytes is a common size.
out	*iv*	Initialization vector (concatenation of nonce and counter) is returned here (16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.15 atcab_aes_decrypt()

```
ATCA_STATUS atcab_aes_decrypt (
            uint16_t key_id,
            uint8_t key_block,
            const uint8_t * ciphertext,
            uint8_t * plaintext )
```

Perform an AES-128 decrypt operation with a key in the device.

Parameters

in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.
in	*ciphertext*	Input ciphertext to be decrypted (16 bytes).
out	*plaintext*	Output plaintext is returned here (16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.16 atcab_aes_encrypt()

```
ATCA_STATUS atcab_aes_encrypt (
            uint16_t key_id,
            uint8_t key_block,
            const uint8_t * plaintext,
            uint8_t * ciphertext )
```

Perform an AES-128 encrypt operation with a key in the device.

Parameters

in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.
in	*plaintext*	Input plaintext to be encrypted (16 bytes).
out	*ciphertext*	Output ciphertext is returned here (16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.17 atcab_aes_gcm_aad_update()

```
ATCA_STATUS atcab_aes_gcm_aad_update (
            atca_aes_gcm_ctx_t * ctx,
            const uint8_t * aad,
            uint32_t aad_size )
```

Process Additional Authenticated Data (AAD) using GCM mode and a key within the ATECC608A device.

This can be called multiple times. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function. When there is AAD to include, this should be called before atcab_aes_gcm_encrypt_update() or atcab_aes_gcm_decrypt_update().

Parameters

in	*ctx*	AES GCM context
in	*aad*	Additional authenticated data to be added
in	*aad_size*	Size of aad in bytes

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.18 atcab_aes_gcm_decrypt_finish()

```
ATCA_STATUS atcab_aes_gcm_decrypt_finish (
            atca_aes_gcm_ctx_t * ctx,
            const uint8_t * tag,
            size_t tag_size,
            bool * is_verified )
```

Complete a GCM decrypt operation verifying the authentication tag.

Parameters

in	*ctx*	AES GCM context structure.
in	*tag*	Expected authentication tag.
in	*tag_size*	Size of tag in bytes (12 to 16 bytes).
out	*is_verified*	Returns whether or not the tag verified.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.19 atcab_aes_gcm_decrypt_update()

```
ATCA_STATUS atcab_aes_gcm_decrypt_update (
            atca_aes_gcm_ctx_t * ctx,
            const uint8_t * ciphertext,
            uint32_t ciphertext_size,
            uint8_t * plaintext )
```

Decrypt data using GCM mode and a key within the ATECC608A device. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function.

Parameters

in	*ctx*	AES GCM context structure.
in	*ciphertext*	Ciphertext to be decrypted.
in	*ciphertext_size*	Size of ciphertext in bytes.
out	*plaintext*	Decrypted data is returned here.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.20 atcab_aes_gcm_encrypt_finish()

```
ATCA_STATUS atcab_aes_gcm_encrypt_finish (
            atca_aes_gcm_ctx_t * ctx,
```

```
            uint8_t * tag,
            size_t tag_size )
```

Complete a GCM encrypt operation returning the authentication tag.

Parameters

in	*ctx*	AES GCM context structure.
out	*tag*	Authentication tag is returned here.
in	*tag_size*	Tag size in bytes (12 to 16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.21 atcab_aes_gcm_encrypt_update()

```
ATCA_STATUS atcab_aes_gcm_encrypt_update (
            atca_aes_gcm_ctx_t * ctx,
            const uint8_t * plaintext,
            uint32_t plaintext_size,
            uint8_t * ciphertext )
```

Encrypt data using GCM mode and a key within the ATECC608A device. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function.

Parameters

in	*ctx*	AES GCM context structure.
in	*plaintext*	Plaintext to be encrypted (16 bytes).
in	*plaintext_size*	Size of plaintext in bytes.
out	*ciphertext*	Encrypted data is returned here.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.22 atcab_aes_gcm_init()

```
ATCA_STATUS atcab_aes_gcm_init (
            atca_aes_gcm_ctx_t * ctx,
            uint16_t key_id,
            uint8_t key_block,
            const uint8_t * iv,
            size_t iv_size )
```

Initialize context for AES GCM operation with an existing IV, which is common when starting a decrypt operation.

Parameters

in	*ctx*	AES GCM context to be initialized.
in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.
in	*iv*	Initialization vector.
in	*iv_size*	Size of IV in bytes. Standard is 12 bytes.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.23 atcab_aes_gcm_init_rand()

```
ATCA_STATUS atcab_aes_gcm_init_rand (
             atca_aes_gcm_ctx_t * ctx,
             uint16_t key_id,
             uint8_t key_block,
             size_t rand_size,
             const uint8_t * free_field,
             size_t free_field_size,
             uint8_t * iv )
```

Initialize context for AES GCM operation with a IV composed of a random and optional fixed(free) field, which is common when starting an encrypt operation.

Parameters

in	*ctx*	AES CTR context to be initialized.
in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.
in	*rand_size*	Size of the random field in bytes. Minimum and recommended size is 12 bytes. Max is 32 bytes.
in	*free_field*	Fixed data to include in the IV after the random field. Can be NULL if not used.
in	*free_field_size*	Size of the free field in bytes.
out	*iv*	Initialization vector is returned here. Its size will be rand_size and free_field_size combined.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.24 atcab_aes_gfm()

```
ATCA_STATUS atcab_aes_gfm (
             const uint8_t * h,
```

```
            const uint8_t * input,
            uint8_t * output )
```

Perform a Galois Field Multiply (GFM) operation.

Parameters

in	*h*	First input value (16 bytes).
in	*input*	Second input value (16 bytes).
out	*output*	GFM result is returned here (16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.25 atcab_cfg_discover()

```
ATCA_STATUS atcab_cfg_discover (
            ATCAIfaceCfg cfg_array[ ],
            int max_ifaces )
```

auto discovery of crypto auth devices

Calls interface discovery functions and fills in cfg_array up to the maximum number of configurations either found or the size of the array. The cfg_array can have a mixture of interface types (ie: some I2C, some SWI or UART) depending upon which interfaces you've enabled

Parameters

out	*cfg_array*	ptr to an array of interface configs
in	*max_ifaces*	maximum size of cfg_array

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.26 atcab_challenge()

```
ATCA_STATUS atcab_challenge (
            const uint8_t * num_in )
```

Execute a Nonce command in pass-through mode to initialize TempKey to a specified value.

Parameters

in	*num_in*	Data to be loaded into TempKey (32 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.27 atcab_challenge_seed_update()

```
ATCA_STATUS atcab_challenge_seed_update (
            const uint8_t * num_in,
            uint8_t * rand_out )
```

Execute a Nonce command to generate a random challenge combining a host nonce (num_in) and a device random number.

Parameters

in	*num_in*	Host nonce to be combined with the device random number (20 bytes).
out	*rand_out*	Internally generated 32-byte random number that was used in the nonce/challenge calculation is returned here. Can be NULL if not needed.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.28 atcab_checkmac()

```
ATCA_STATUS atcab_checkmac (
            uint8_t mode,
            uint16_t key_id,
            const uint8_t * challenge,
            const uint8_t * response,
            const uint8_t * other_data )
```

Compares a MAC response with input values.

Parameters

in	*mode*	Controls which fields within the device are used in the message
in	*key_id*	Key location in the CryptoAuth device to use for the MAC
in	*challenge*	Challenge data (32 bytes)
in	*response*	MAC response data (32 bytes)
in	*other_data*	OtherData parameter (13 bytes)

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.29 atcab_cmp_config_zone()

```
ATCA_STATUS atcab_cmp_config_zone (
            uint8_t * config_data,
            bool * same_config )
```

Compares a specified configuration zone with the configuration zone currently on the device.

This only compares the static portions of the configuration zone and skips those that are unique per device (first 16 bytes) and areas that can change after the configuration zone has been locked (e.g. LastKeyUse).

Parameters

in	*config_data*	Full configuration data to compare the device against.
out	*same_config*	Result is returned here. True if the static portions on the configuration zones are the same.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Max for all configs

13.6.4.30 atcab_counter()

```
ATCA_STATUS atcab_counter (
            uint8_t mode,
            uint16_t counter_id,
            uint32_t * counter_value )
```

Compute the Counter functions.

Parameters

in	*mode*	the mode used for the counter
in	*counter_id*	The counter to be used
out	*counter_value*	pointer to the counter value returned from device

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.31 atcab_counter_increment()

```
ATCA_STATUS atcab_counter_increment (
            uint16_t counter_id,
            uint32_t * counter_value )
```

Increments one of the device's monotonic counters.

Parameters

in	*counter_id*	Counter to be incremented
out	*counter_value*	New value of the counter is returned here. Can be NULL if not needed.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.32 atcab_counter_read()

```
ATCA_STATUS atcab_counter_read (
            uint16_t counter_id,
            uint32_t * counter_value )
```

Read one of the device's monotonic counters.

Parameters

in	*counter_id*	Counter to be read
out	*counter_value*	Counter value is returned here.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.33 atcab_derivekey()

```
ATCA_STATUS atcab_derivekey (
            uint8_t mode,
            uint16_t target_key,
            const uint8_t * mac )
```

Executes the DeviceKey command for deriving a new key from a nonce (TempKey) and an existing key.

Parameters

in	*mode*	Bit 2 must match the value in TempKey.SourceFlag
in	*target_key*	Key slot to be written
in	*mac*	Optional 32 byte MAC used to validate operation. NULL if not required.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.34 atcab_ecdh()

```
ATCA_STATUS atcab_ecdh (
            uint16_t key_id,
            const uint8_t * public_key,
            uint8_t * pms )
```

ECDH command with a private key in a slot and the premaster secret is returned in the clear.

Parameters

in	*key_id*	Slot of key for ECDH computation
in	*public_key*	Public key input to ECDH calculation. X and Y integers in big-endian format. 64 bytes for P256 key.
out	*pms*	Computed ECDH premaster secret is returned here. 32 bytes.

Returns

ATCA_SUCCESS on success

13.6.4.35 atcab_ecdh_base()

```
ATCA_STATUS atcab_ecdh_base (
            uint8_t mode,
            uint16_t key_id,
            const uint8_t * public_key,
            uint8_t * pms,
            uint8_t * out_nonce )
```

Base function for generating premaster secret key using ECDH.

Parameters

in	*mode*	Mode to be used for ECDH computation
in	*key_id*	Slot of key for ECDH computation
in	*public_key*	Public key input to ECDH calculation. X and Y integers in big-endian format. 64 bytes for P256 key.
out	*pms*	Computed ECDH pre-master secret is returned here (32 bytes) if returned directly. Otherwise NULL.
out	*out_nonce*	Nonce used to encrypt pre-master secret. NULL if output encryption not used.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.36 atcab_ecdh_enc()

```
ATCA_STATUS atcab_ecdh_enc (
            uint16_t key_id,
            const uint8_t * public_key,
            uint8_t * pms,
            const uint8_t * read_key,
            uint16_t read_key_id )
```

ECDH command with a private key in a slot and the premaster secret is read from the next slot.

This function only works for even numbered slots with the proper configuration.

Parameters

in	*key_id*	Slot of key for ECDH computation
in	*public_key*	Public key input to ECDH calculation. X and Y integers in big-endian format. 64 bytes for P256 key.
out	*pms*	Computed ECDH premaster secret is returned here (32 bytes).
in	*read_key*	Read key for the premaster secret slot (key_id|1).
in	*read_key↔_id*	Read key slot for read_key.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.37 atcab_ecdh_ioenc()

```
ATCA_STATUS atcab_ecdh_ioenc (
            uint16_t key_id,
            const uint8_t * public_key,
            uint8_t * pms,
            const uint8_t * io_key )
```

ECDH command with a private key in a slot and the premaster secret is returned encrypted using the IO protection key.

Parameters

in	*key_id*	Slot of key for ECDH computation
in	*public_key*	Public key input to ECDH calculation. X and Y integers in big-endian format. 64 bytes for P256 key.
out	*pms*	Computed ECDH premaster secret is returned here (32 bytes).
in	*io_key*	IO protection key.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.38 atcab_ecdh_tempkey()

```
ATCA_STATUS atcab_ecdh_tempkey (
            const uint8_t * public_key,
            uint8_t * pms )
```

ECDH command with a private key in TempKey and the premaster secret is returned in the clear.

Parameters

in	*public_key*	Public key input to ECDH calculation. X and Y integers in big-endian format. 64 bytes for P256 key.
out	*pms*	Computed ECDH premaster secret is returned here (32 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.39 atcab_ecdh_tempkey_ioenc()

```
ATCA_STATUS atcab_ecdh_tempkey_ioenc (
            const uint8_t * public_key,
            uint8_t * pms,
            const uint8_t * io_key )
```

ECDH command with a private key in TempKey and the premaster secret is returned encrypted using the IO protection key.

Parameters

in	*public_key*	Public key input to ECDH calculation. X and Y integers in big-endian format. 64 bytes for P256 key.
out	*pms*	Computed ECDH premaster secret is returned here (32 bytes).
in	*io_key*	IO protection key.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.40 atcab_gendig()

```
ATCA_STATUS atcab_gendig (
            uint8_t zone,
            uint16_t key_id,
            const uint8_t * other_data,
            uint8_t other_data_size )
```

Issues a GenDig command, which performs a SHA256 hash on the source data indicated by zone with the contents of TempKey. See the CryptoAuth datasheet for your chip to see what the values of zone correspond to.

Parameters

in	*zone*	Designates the source of the data to hash with TempKey.
in	*key_id*	Indicates the key, OTP block, or message order for shared nonce mode.
in	*other_data*	Four bytes of data for SHA calculation when using a NoMac key, 32 bytes for "Shared Nonce" mode, otherwise ignored (can be NULL).
in	*other_data_size*	Size of other_data in bytes.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.41 atcab_genkey()

```
ATCA_STATUS atcab_genkey (
            uint16_t key_id,
            uint8_t * public_key )
```

Issues GenKey command, which generates a new random private key in slot and returns the public key.

Parameters

in	*key_id*	Slot number where an ECC private key is configured. Can also be ATCA_TEMPKEY_KEYID to generate a private key in TempKey.
out	*public_key*	Public key will be returned here. Format will be the X and Y integers in big-endian format. 64 bytes for P256 curve. Set to NULL if public key isn't required.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.42 atcab_genkey_base()

```
ATCA_STATUS atcab_genkey_base (
            uint8_t mode,
```

```
                uint16_t key_id,
                const uint8_t * other_data,
                uint8_t * public_key )
```

Issues GenKey command, which can generate a private key, compute a public key, nd/or compute a digest of a public key.

Parameters

in	*mode*	Mode determines what operations the GenKey command performs.
in	*key_id*	Slot to perform the GenKey command on.
in	*other_data*	OtherData for PubKey digest calculation. Can be set to NULL otherwise.
out	*public_key*	If the mode indicates a public key will be calculated, it will be returned here. Format will be the X and Y integers in big-endian format. 64 bytes for P256 curve. Set to NULL if public key isn't required.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.43 atcab_get_addr()

```
ATCA_STATUS atcab_get_addr (
                uint8_t zone,
                uint16_t slot,
                uint8_t block,
                uint8_t offset,
                uint16_t * addr )
```

Compute the address given the zone, slot, block, and offset.

Parameters

in	*zone*	Zone to get address from. Config(0), OTP(1), or Data(2) which requires a slot.
in	*slot*	Slot Id number for data zone and zero for other zones.
in	*block*	Block number within the data or configuration or OTP zone .
in	*offset*	Offset Number within the block of data or configuration or OTP zone.
out	*addr*	Pointer to the address of data or configuration or OTP zone.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.44 atcab_get_device()

```
ATCADevice atcab_get_device (
                void )
```

Get the global device object.

Returns

instance of global ATCADevice

13.6.4.45 atcab_get_device_type()

```
ATCADeviceType atcab_get_device_type (
            void )
```

Get the current device type.

Returns

Device type if basic api is initialized or ATCA_DEV_UNKNOWN.

13.6.4.46 atcab_get_pubkey()

```
ATCA_STATUS atcab_get_pubkey (
            uint16_t key_id,
            uint8_t * public_key )
```

Uses GenKey command to calculate the public key from an existing private key in a slot.

Parameters

in	*key_id*	Slot number of the private key.
out	*public_key*	Public key will be returned here. Format will be the X and Y integers in big-endian format. 64 bytes for P256 curve. Set to NULL if public key isn't required.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.47 atcab_get_zone_size()

```
ATCA_STATUS atcab_get_zone_size (
            uint8_t zone,
            uint16_t slot,
            size_t * size )
```

Gets the size of the specified zone in bytes.

Parameters

in	*zone*	Zone to get size information from. Config(0), OTP(1), or Data(2) which requires a slot.
in	*slot*	If zone is Data(2), the slot to query for size.
out	*size*	Zone size is returned here.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.48 atcab_hmac()

```
ATCA_STATUS atcab_hmac (
            uint8_t mode,
            uint16_t key_id,
            uint8_t * digest )
```

Issues a HMAC command, which computes an HMAC/SHA-256 digest of a key stored in the device, a challenge, and other information on the device.

Parameters

in	*mode*	Controls which fields within the device are used in the message.
in	*key_id*	Which key is to be used to generate the response. Bits 0:3 only are used to select a slot but all 16 bits are used in the HMAC message.
out	*digest*	HMAC digest is returned in this buffer (32 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.49 atcab_hw_sha2_256()

```
ATCA_STATUS atcab_hw_sha2_256 (
            const uint8_t * data,
            size_t data_size,
            uint8_t * digest )
```

Use the SHA command to compute a SHA-256 digest.

Parameters

in	*data*	Message data to be hashed.
in	*data_size*	Size of data in bytes.
out	*digest*	Digest is returned here (32 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.50 atcab_hw_sha2_256_finish()

```
ATCA_STATUS atcab_hw_sha2_256_finish (
            atca_sha256_ctx_t * ctx,
            uint8_t * digest )
```

Finish SHA-256 digest for a SHA context for performing a hardware SHA-256 operation on a device.

Parameters

in	*ctx*	SHA256 context
out	*digest*	SHA256 digest is returned here (32 bytes)

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.51 atcab_hw_sha2_256_init()

```
ATCA_STATUS atcab_hw_sha2_256_init (
            atca_sha256_ctx_t * ctx )
```

Initialize a SHA context for performing a hardware SHA-256 operation on a device. Note that only one SHA operation can be run at a time.

Parameters

in	*ctx*	SHA256 context

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.52 atcab_hw_sha2_256_update()

```
ATCA_STATUS atcab_hw_sha2_256_update (
            atca_sha256_ctx_t * ctx,
            const uint8_t * data,
            size_t data_size )
```

Add message data to a SHA context for performing a hardware SHA-256 operation on a device.

Parameters

in	*ctx*	SHA256 context
in	*data*	Message data to be added to hash.
in	*data_size*	Size of data in bytes.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.53 atcab_idle()

```
ATCA_STATUS atcab_idle (
            void )
```

idle the CryptoAuth device

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.54 atcab_info()

```
ATCA_STATUS atcab_info (
            uint8_t * revision )
```

Use the Info command to get the device revision (DevRev).

Parameters

out	*revision*	Device revision is returned here (4 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.55 atcab_info_base()

```
ATCA_STATUS atcab_info_base (
            uint8_t mode,
            uint16_t param2,
            uint8_t * out_data )
```

Issues an Info command, which return internal device information and can control GPIO and the persistent latch.

Parameters

in	*mode*	Selects which mode to be used for info command.
in	*param2*	Selects the particular fields for the mode.
out	*out_data*	Response from info command (4 bytes). Can be set to NULL if not required.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.56 atcab_info_get_latch()

```
ATCA_STATUS atcab_info_get_latch (
            bool * state )
```

Use the Info command to get the persistent latch current state for an ATECC608A device.

Parameters

out	*state*	The state is returned here. Set (true) or Cler (false).

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.57 atcab_info_set_latch()

```
ATCA_STATUS atcab_info_set_latch (
            bool state )
```

Use the Info command to set the persistent latch state for an ATECC608A device.

Parameters

out	*state*	Persistent latch state. Set (true) or clear (false).

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.58 atcab_init()

```
ATCA_STATUS atcab_init (
            ATCAIfaceCfg * cfg )
```

Creates a global ATCADevice object used by Basic API.

Parameters

in	*cfg*	Logical interface configuration. Some predefined configurations can be found in atca_cfgs.h

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.59 atcab_init_device()

```
ATCA_STATUS atcab_init_device (
            ATCADevice ca_device )
```

Initialize the global ATCADevice object to point to one of your choosing for use with all the atcab_ basic API.

Parameters

in	*ca_device*	ATCADevice instance to use as the global Basic API crypto device instance

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.60 atcab_is_locked()

```
ATCA_STATUS atcab_is_locked (
            uint8_t zone,
            bool * is_locked )
```

Executes Read command, which reads the configuration zone to see if the specified zone is locked.

Parameters

in	*zone*	The zone to query for locked (use LOCK_ZONE_CONFIG or LOCK_ZONE_DATA).
out	*is_locked*	Lock state returned here. True if locked.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.61 atcab_is_slot_locked()

```
ATCA_STATUS atcab_is_slot_locked (
            uint16_t slot,
            bool * is_locked )
```

Executes Read command, which reads the configuration zone to see if the specified slot is locked.

Parameters

in	*slot*	Slot to query for locked (slot 0-15)
out	*is_locked*	Lock state returned here. True if locked.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.62 atcab_kdf()

```
ATCA_STATUS atcab_kdf (
            uint8_t mode,
            uint16_t key_id,
            const uint32_t details,
            const uint8_t * message,
            uint8_t * out_data,
            uint8_t * out_nonce )
```

Executes the KDF command, which derives a new key in PRF, AES, or HKDF modes.

Generally this function combines a source key with an input string and creates a result key/digest/array.

Parameters

in	*mode*	Mode determines KDF algorithm (PRF,AES,HKDF), source key location, and target key locations.
in	*key_id*	Source and target key slots if locations are in the EEPROM. Source key slot is the LSB and target key slot is the MSB.
in	*details*	Further information about the computation, depending on the algorithm (4 bytes).
in	*message*	Input value from system (up to 128 bytes). Actual size of message is 16 bytes for AES algorithm or is encoded in the MSB of the details parameter for other algorithms.
out	*out_data*	Output of the KDF function is returned here. If the result remains in the device, this can be NULL.
out	*out_nonce*	If the output is encrypted, a 32 byte random nonce generated by the device is returned here. If output encryption is not used, this can be NULL.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.63 atcab_lock()

```
ATCA_STATUS atcab_lock (
            uint8_t mode,
            uint16_t summary_crc )
```

The Lock command prevents future modifications of the Configuration and/or Data and OTP zones. If the device is so configured, then this command can be used to lock individual data slots. This command fails if the designated area is already locked.

Parameters

in	*mode*	Zone, and/or slot, and summary check (bit 7).
in	*summary_crc*	CRC of the config or data zones. Ignored for slot locks or when mode bit 7 is set.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.64 atcab_lock_config_zone()

```
ATCA_STATUS atcab_lock_config_zone (
            void )
```

Unconditionally (no CRC required) lock the config zone.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.65 atcab_lock_config_zone_crc()

```
ATCA_STATUS atcab_lock_config_zone_crc (
            uint16_t summary_crc )
```

Lock the config zone with summary CRC.

The CRC is calculated over the entire config zone contents. 88 bytes for ATSHA devices, 128 bytes for ATECC devices. Lock will fail if the provided CRC doesn't match the internally calculated one.

Parameters

in	*summary_crc*	Expected CRC over the config zone.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.66 atcab_lock_data_slot()

```
ATCA_STATUS atcab_lock_data_slot (
            uint16_t slot )
```

Lock an individual slot in the data zone on an ATECC device. Not available for ATSHA devices. Slot must be configured to be slot lockable (KeyConfig.Lockable=1).

Parameters

in	*slot*	Slot to be locked in data zone.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.67 atcab_lock_data_zone()

```
ATCA_STATUS atcab_lock_data_zone (
            void )
```

Unconditionally (no CRC required) lock the data zone (slots and OTP).

ConfigZone must be locked and DataZone must be unlocked for the zone to be successfully locked.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.68 atcab_lock_data_zone_crc()

```
ATCA_STATUS atcab_lock_data_zone_crc (
            uint16_t summary_crc )
```

Lock the data zone (slots and OTP) with summary CRC.

The CRC is calculated over the concatenated contents of all the slots and OTP at the end. Private keys (KeyConfig.Private=1) are skipped. Lock will fail if the provided CRC doesn't match the internally calculated one.

Parameters

in	*summary_crc*	Expected CRC over the data zone.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.69 atcab_mac()

```
ATCA_STATUS atcab_mac (
            uint8_t mode,
            uint16_t key_id,
            const uint8_t * challenge,
            uint8_t * digest )
```

Executes MAC command, which computes a SHA-256 digest of a key stored in the device, a challenge, and other information on the device.

Parameters

in	*mode*	Controls which fields within the device are used in the message
in	*key_id*	Key in the CryptoAuth device to use for the MAC
in	*challenge*	Challenge message (32 bytes). May be NULL if mode indicates a challenge isn't required.
out	*digest*	MAC response is returned here (32 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.70 atcab_nonce()

```
ATCA_STATUS atcab_nonce (
            const uint8_t * num_in )
```

Execute a Nonce command in pass-through mode to initialize TempKey to a specified value.

Parameters

in	*num_in*	Data to be loaded into TempKey (32 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.71 atcab_nonce_base()

```
ATCA_STATUS atcab_nonce_base (
            uint8_t mode,
            uint16_t zero,
            const uint8_t * num_in,
            uint8_t * rand_out )
```

Executes Nonce command, which loads a random or fixed nonce/data into the device for use by subsequent commands.

Parameters

in	*mode*	Controls the mechanism of the internal RNG or fixed write.
in	*zero*	Param2, normally 0, but can be used to indicate a nonce calculation mode (bit 15).
in	*num_in*	Input value to either be included in the nonce calculation in random modes (20 bytes) or to be written directly (32 bytes or 64 bytes(ATECC608A)) in pass-through mode.
out	*rand_out*	If using a random mode, the internally generated 32-byte random number that was used in the nonce calculation is returned here. Can be NULL if not needed.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.72 atcab_nonce_load()

```
ATCA_STATUS atcab_nonce_load (
            uint8_t target,
            const uint8_t * num_in,
            uint16_t num_in_size )
```

Execute a Nonce command in pass-through mode to load one of the device's internal buffers with a fixed value.

For the ATECC608A, available targets are TempKey (32 or 64 bytes), Message Digest Buffer (32 or 64 bytes), or the Alternate Key Buffer (32 bytes). For all other devices, only TempKey (32 bytes) is available.

Parameters

in	*target*	Target device buffer to load. Can be NONCE_MODE_TARGET_TEMPKEY, NONCE_MODE_TARGET_MSGDIGBUF, or NONCE_MODE_TARGET_ALTKEYBUF.
in	*num_in*	Data to load into the buffer.
in	*num_in_size*	Size of num_in in bytes. Can be 32 or 64 bytes depending on device and target.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.73 atcab_nonce_rand()

```
ATCA_STATUS atcab_nonce_rand (
            const uint8_t * num_in,
            uint8_t * rand_out )
```

Execute a Nonce command to generate a random nonce combining a host nonce (num_in) and a device random number.

Parameters

in	*num_in*	Host nonce to be combined with the device random number (20 bytes).
out	*rand_out*	Internally generated 32-byte random number that was used in the nonce/challenge calculation is returned here. Can be NULL if not needed.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.74 atcab_printbin()

```
ATCA_STATUS atcab_printbin (
            uint8_t * binary,
            size_t bin_len,
            bool add_space )
```

13.6.4.75 atcab_priv_write()

```
ATCA_STATUS atcab_priv_write (
            uint16_t key_id,
            const uint8_t priv_key[36],
            uint16_t write_key_id,
            const uint8_t write_key[32] )
```

Executes PrivWrite command, to write externally generated ECC private keys into the device.

Parameters

in	*key_id*	Slot to write the external private key into.
in	*priv_key*	External private key (36 bytes) to be written. The first 4 bytes should be zero for P256 curve.
in	*write_key←_id*	Write key slot. Ignored if write_key is NULL.
in	*write_key*	Write key (32 bytes). If NULL, perform an unencrypted PrivWrite, which is only available when the data zone is unlocked.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.76 atcab_random()

```
ATCA_STATUS atcab_random (
            uint8_t * rand_out )
```

Executes Random command, which generates a 32 byte random number from the CryptoAuth device.

Parameters

out	*rand_out*	32 bytes of random data is returned here.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.77 atcab_read_bytes_zone()

```
ATCA_STATUS atcab_read_bytes_zone (
            uint8_t zone,
            uint16_t slot,
            size_t offset,
            uint8_t * data,
            size_t length )
```

Used to read an arbitrary number of bytes from any zone configured for clear reads.

This function will issue the Read command as many times as is required to read the requested data.

Parameters

in	*zone*	Zone to read data from. Option are ATCA_ZONE_CONFIG(0), ATCA_ZONE_OTP(1), or ATCA_ZONE_DATA(2).
in	*slot*	Slot number to read from if zone is ATCA_ZONE_DATA(2). Ignored for all other zones.
in	*offset*	Byte offset within the zone to read from.
out	*data*	Read data is returned here.
in	*length*	Number of bytes to read starting from the offset.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.78 atcab_read_config_zone()

```
ATCA_STATUS atcab_read_config_zone (
            uint8_t * config_data )
```

Executes Read command to read the complete device configuration zone.

Parameters

out	*config_data*	Configuration zone data is returned here. 88 bytes for ATSHA devices, 128 bytes for ATECC devices.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.79 atcab_read_enc()

```
ATCA_STATUS atcab_read_enc (
            uint16_t key_id,
            uint8_t block,
            uint8_t * data,
            const uint8_t * enc_key,
            const uint16_t enc_key_id )
```

Executes Read command on a slot configured for encrypted reads and decrypts the data to return it as plaintext.

Data zone must be locked for this command to succeed. Can only read 32 byte blocks.

Parameters

in	*key_id*	The slot ID to read from.
in	*block*	Index of the 32 byte block within the slot to read.
out	*data*	Decrypted (plaintext) data from the read is returned here (32 bytes).
in	*enc_key*	32 byte ReadKey for the slot being read.
in	*enc_key_id*	KeyID of the ReadKey being used.

returns ATCA_SUCCESS on success, otherwise an error code.

13.6.4.80 atcab_read_pubkey()

```
ATCA_STATUS atcab_read_pubkey (
            uint16_t slot,
            uint8_t * public_key )
```

Executes Read command to read an ECC P256 public key from a slot configured for clear reads.

This function assumes the public key is stored using the ECC public key format specified in the datasheet.

Parameters

`in`	*slot*	Slot number to read from. Only slots 8 to 15 are large enough for a public key.
`out`	*public_key*	Public key is returned here (64 bytes). Format will be the 32 byte X and Y big-endian integers concatenated.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.81 atcab_read_serial_number()

```
ATCA_STATUS atcab_read_serial_number (
            uint8_t * serial_number )
```

Executes Read command, which reads the 9 byte serial number of the device from the config zone.

Parameters

`out`	*serial_number*	9 byte serial number is returned here.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.82 atcab_read_sig()

```
ATCA_STATUS atcab_read_sig (
            uint16_t slot,
            uint8_t * sig )
```

Executes Read command to read a 64 byte ECDSA P256 signature from a slot configured for clear reads.

Parameters

`in`	*slot*	Slot number to read from. Only slots 8 to 15 are large enough for a signature.
`out`	*sig*	Signature will be returned here (64 bytes). Format will be the 32 byte R and S big-endian integers concatenated.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.83 atcab_read_zone()

```
ATCA_STATUS atcab_read_zone (
            uint8_t zone,
            uint16_t slot,
            uint8_t block,
            uint8_t offset,
            uint8_t * data,
            uint8_t len )
```

Executes Read command, which reads either 4 or 32 bytes of data from a given slot, configuration zone, or the OTP zone.

When reading a slot or OTP, data zone must be locked and the slot configuration must not be secret for a slot to be successfully read.

Parameters

in	*zone*	Zone to be read from device. Options are ATCA_ZONE_CONFIG, ATCA_ZONE_OTP, or ATCA_ZONE_DATA.
in	*slot*	Slot number for data zone and ignored for other zones.
in	*block*	32 byte block index within the zone.
in	*offset*	4 byte work index within the block. Ignored for 32 byte reads.
out	*data*	Read data is returned here.
in	*len*	Length of the data to be read. Must be either 4 or 32.

returns ATCA_SUCCESS on success, otherwise an error code.

13.6.4.84 atcab_release()

```
ATCA_STATUS atcab_release (
            void )
```

release (free) the global ATCADevice instance. This must be called in order to release or free up the interface.

Returns

Returns ATCA_SUCCESS .

13.6.4.85 atcab_secureboot()

```
ATCA_STATUS atcab_secureboot (
            uint8_t mode,
            uint16_t param2,
            const uint8_t * digest,
            const uint8_t * signature,
            uint8_t * mac )
```

Executes Secure Boot command, which provides support for secure boot of an external MCU or MPU.

Parameters

in	*mode*	Mode determines what operations the SecureBoot command performs.
in	*param2*	Not used, must be 0.
in	*digest*	Digest of the code to be verified (32 bytes).
in	*signature*	Signature of the code to be verified (64 bytes). Can be NULL when using the FullStore mode.
out	*mac*	Validating MAC will be returned here (32 bytes). Can be NULL if not required.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.86 atcab_secureboot_mac()

```
ATCA_STATUS atcab_secureboot_mac (
            uint8_t mode,
            const uint8_t * digest,
            const uint8_t * signature,
            const uint8_t * num_in,
            const uint8_t * io_key,
            bool * is_verified )
```

Executes Secure Boot command with encrypted digest and validated MAC response using the IO protection key.

Parameters

in	*mode*	Mode determines what operations the SecureBoot command performs.
in	*digest*	Digest of the code to be verified (32 bytes). This is the plaintext digest (not encrypted).
in	*signature*	Signature of the code to be verified (64 bytes). Can be NULL when using the FullStore mode.
in	*num_in*	Host nonce (20 bytes).
in	*io_key*	IO protection key (32 bytes).
out	*is_verified*	Verify result is returned here.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.87 atcab_selftest()

```
ATCA_STATUS atcab_selftest (
            uint8_t mode,
            uint16_t param2,
            uint8_t * result )
```

Executes the SelfTest command, which performs a test of one or more of the cryptographic engines within the ATECC608A chip.

Parameters

in	*mode*	Functions to test. Can be a bit field combining any of the following: SELFTEST_MODE_RNG, SELFTEST_MODE_ECDSA_VERIFY, SELFTEST_MODE_ECDSA_SIGN, SELFTEST_MODE_ECDH, SELFTEST_MODE_AES, SELFTEST_MODE_SHA, SELFTEST_MODE_ALL.
in	*param2*	Currently unused, should be 0.
out	*result*	Results are returned here as a bit field.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.88 atcab_sha()

```
ATCA_STATUS atcab_sha (
            uint16_t length,
            const uint8_t * message,
            uint8_t * digest )
```

Use the SHA command to compute a SHA-256 digest.

Parameters

in	*length*	Size of message parameter in bytes.
in	*message*	Message data to be hashed.
out	*digest*	Digest is returned here (32 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.89 atcab_sha_base()

```
ATCA_STATUS atcab_sha_base (
            uint8_t mode,
            uint16_t length,
            const uint8_t * message,
            uint8_t * data_out,
            uint16_t * data_out_size )
```

Executes SHA command, which computes a SHA-256 or HMAC/SHA-256 digest for general purpose use by the host system.

Only the Start(0) and Compute(1) modes are available for ATSHA devices.

Parameters

in	*mode*	SHA command mode Start(0), Update/Compute(1), End(2), Public(3), HMACstart(4), HMACend(5), Read_Context(6), or Write_Context(7). Also message digest target location for the ATECC608A.
in	*length*	Number of bytes in the message parameter or KeySlot for the HMAC key if Mode is HMACstart(4) or Public(3).
in	*message*	Message bytes to be hashed or Write_Context if restoring a context on the ATECC608A. Can be NULL if not required by the mode.
out	*data_out*	Data returned by the command (digest or context).
in,out	*data_out_size*	As input, the size of the data_out buffer. As output, the number of bytes returned in data_out.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.90 atcab_sha_end()

```
ATCA_STATUS atcab_sha_end (
            uint8_t * digest,
            uint16_t length,
            const uint8_t * message )
```

Executes SHA command to complete SHA-256 or HMAC/SHA-256 operation.

Parameters

out	*digest*	Digest from SHA-256 or HMAC/SHA-256 will be returned here (32 bytes).
in	*length*	Length of any remaining data to include in hash. Max 64 bytes.
in	*message*	Remaining data to include in hash. NULL if length is 0.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.91 atcab_sha_hmac()

```
ATCA_STATUS atcab_sha_hmac (
            const uint8_t * data,
            size_t data_size,
            uint16_t key_slot,
            uint8_t * digest,
            uint8_t target )
```

Use the SHA command to compute an HMAC/SHA-256 operation.

Parameters

in	*data*	Message data to be hashed.
in	*data_size*	Size of data in bytes.
in	*key_slot*	Slot key id to use for the HMAC calculation
out	*digest*	Digest is returned here (32 bytes).
in	*target*	Where to save the digest internal to the device. For ATECC608A, can be SHA_MODE_TARGET_TEMPKEY, SHA_MODE_TARGET_MSGDIGBUF, or SHA_MODE_TARGET_OUT_ONLY. For all other devices, SHA_MODE_TARGET_TEMPKEY is the only option.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.92 atcab_sha_hmac_finish()

```
ATCA_STATUS atcab_sha_hmac_finish (
            atca_hmac_sha256_ctx_t * ctx,
            uint8_t * digest,
            uint8_t target )
```

Executes SHA command to complete a HMAC/SHA-256 operation.

Parameters

in	*ctx*	HMAC/SHA-256 context
out	*digest*	HMAC/SHA-256 result is returned here (32 bytes).
in	*target*	Where to save the digest internal to the device. For ATECC608A, can be SHA_MODE_TARGET_TEMPKEY, SHA_MODE_TARGET_MSGDIGBUF, or SHA_MODE_TARGET_OUT_ONLY. For all other devices, SHA_MODE_TARGET_TEMPKEY is the only option.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.93 atcab_sha_hmac_init()

```
ATCA_STATUS atcab_sha_hmac_init (
            atca_hmac_sha256_ctx_t * ctx,
            uint16_t key_slot )
```

Executes SHA command to start an HMAC/SHA-256 operation.

Parameters

in	*ctx*	HMAC/SHA-256 context
in	*key_slot*	Slot key id to use for the HMAC calculation

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.94 atcab_sha_hmac_update()

```
ATCA_STATUS atcab_sha_hmac_update (
            atca_hmac_sha256_ctx_t * ctx,
            const uint8_t * data,
            size_t data_size )
```

Executes SHA command to add an arbitrary amount of message data to a HMAC/SHA-256 operation.

Parameters

in	*ctx*	HMAC/SHA-256 context
in	*data*	Message data to add
in	*data_size*	Size of message data in bytes

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.95 atcab_sha_read_context()

```
ATCA_STATUS atcab_sha_read_context (
            uint8_t * context,
            uint16_t * context_size )
```

Executes SHA command to read the SHA-256 context back. Only for ATECC608A with SHA-256 contexts. HMAC not supported.

Parameters

out	*context*	Context data is returned here.
in,out	*context_size*	As input, the size of the context buffer in bytes. As output, the size of the returned context data.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.96 atcab_sha_start()

```
ATCA_STATUS atcab_sha_start (
            void )
```

Executes SHA command to initialize SHA-256 calculation engine.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.97 atcab_sha_update()

```
ATCA_STATUS atcab_sha_update (
            const uint8_t * message )
```

Executes SHA command to add 64 bytes of message data to the current context.

Parameters

in	*message*	64 bytes of message data to add to add to operation.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.98 atcab_sha_write_context()

```
ATCA_STATUS atcab_sha_write_context (
            const uint8_t * context,
            uint16_t context_size )
```

Executes SHA command to write (restore) a SHA-256 context into the the device. Only supported for ATECC608A with SHA-256 contexts.

Parameters

in	*context*	Context data to be restored.
in	*context_size*	Size of the context data in bytes.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.99 atcab_sign()

```
ATCA_STATUS atcab_sign (
            uint16_t key_id,
            const uint8_t * msg,
            uint8_t * signature )
```

Executes Sign command, to sign a 32-byte external message using the private key in the specified slot. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608A device or TempKey for other devices.

Parameters

in	*key_id*	Slot of the private key to be used to sign the message.
in	*msg*	32-byte message to be signed. Typically the SHA256 hash of the full message.
out	*signature*	Signature will be returned here. Format is R and S integers in big-endian format. 64 bytes for P256 curve.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.100 atcab_sign_base()

```
ATCA_STATUS atcab_sign_base (
            uint8_t mode,
            uint16_t key_id,
            uint8_t * signature )
```

Executes the Sign command, which generates a signature using the ECDSA algorithm.

Parameters

in	*mode*	Mode determines what the source of the message to be signed.
in	*key_id*	Private key slot used to sign the message.
out	*signature*	Signature is returned here. Format is R and S integers in big-endian format. 64 bytes for P256 curve.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.101 atcab_sign_internal()

```
ATCA_STATUS atcab_sign_internal (
            uint16_t key_id,
            bool is_invalidate,
            bool is_full_sn,
            uint8_t * signature )
```

Executes Sign command to sign an internally generated message.

Parameters

in	*key_id*	Slot of the private key to be used to sign the message.
in	*is_invalidate*	Set to true if the signature will be used with the Verify(Invalidate) command. false for all other cases.
in	*is_full_sn*	Set to true if the message should incorporate the device's full serial number.
out	*signature*	Signature is returned here. Format is R and S integers in big-endian format. 64 bytes for P256 curve.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.102 atcab_sleep()

```
ATCA_STATUS atcab_sleep (
            void )
```

invoke sleep on the CryptoAuth device

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.103 atcab_updateextra()

```
ATCA_STATUS atcab_updateextra (
            uint8_t mode,
            uint16_t new_value )
```

Executes UpdateExtra command to update the values of the two extra bytes within the Configuration zone (bytes 84 and 85).

Can also be used to decrement the limited use counter associated with the key in slot NewValue.

Parameters

in	*mode*	Mode determines what operations the UpdateExtra command performs.
in	*new_value*	Value to be written.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.104 atcab_verify()

```
ATCA_STATUS atcab_verify (
            uint8_t mode,
            uint16_t key_id,
            const uint8_t * signature,
            const uint8_t * public_key,
            const uint8_t * other_data,
            uint8_t * mac )
```

Executes the Verify command, which takes an ECDSA [R,S] signature and verifies that it is correctly generated from a given message and public key. In all cases, the signature is an input to the command.

For the Stored, External, and ValidateExternal Modes, the contents of TempKey (or Message Digest Buffer in some cases for the ATECC608A) should contain the 32 byte message.

Parameters

in	*mode*	Verify command mode and options
in	*key_id*	Stored mode, the slot containing the public key to be used for the verification. ValidateExternal mode, the slot containing the public key to be validated. External mode, KeyID contains the curve type to be used to Verify the signature. Validate or Invalidate mode, the slot containing the public key to be (in)validated.
in	*signature*	Signature to be verified. R and S integers in big-endian format. 64 bytes for P256 curve.
in	*public_key*	If mode is External, the public key to be used for verification. X and Y integers in big-endian format. 64 bytes for P256 curve. NULL for all other modes.
in	*other_data*	If mode is Validate, the bytes used to generate the message for the validation (19 bytes). NULL for all other modes.
out	*mac*	If mode indicates a validating MAC, then the MAC will will be returned here. Can be NULL otherwise.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.105 atcab_verify_extern()

```
ATCA_STATUS atcab_verify_extern (
            const uint8_t * message,
```

```
const uint8_t * signature,
const uint8_t * public_key,
bool * is_verified )
```

Executes the Verify command, which verifies a signature (ECDSA verify operation) with all components (message, signature, and public key) supplied. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608A device or TempKey for other devices.

Parameters

in	*message*	32 byte message to be verified. Typically the SHA256 hash of the full message.
in	*signature*	Signature to be verified. R and S integers in big-endian format. 64 bytes for P256 curve.
in	*public_key*	The public key to be used for verification. X and Y integers in big-endian format. 64 bytes for P256 curve.
out	*is_verified*	Boolean whether or not the message, signature, public key verified.

Returns

ATCA_SUCCESS on verification success or failure, because the command still completed successfully.

13.6.4.106 atcab_verify_extern_mac()

```
ATCA_STATUS atcab_verify_extern_mac (
            const uint8_t * message,
            const uint8_t * signature,
            const uint8_t * public_key,
            const uint8_t * num_in,
            const uint8_t * io_key,
            bool * is_verified )
```

Executes the Verify command with verification MAC, which verifies a signature (ECDSA verify operation) with all components (message, signature, and public key) supplied. This function is only available on the ATECC608A.

Parameters

in	*message*	32 byte message to be verified. Typically the SHA256 hash of the full message.
in	*signature*	Signature to be verified. R and S integers in big-endian format. 64 bytes for P256 curve.
in	*public_key*	The public key to be used for verification. X and Y integers in big-endian format. 64 bytes for P256 curve.
in	*num_in*	System nonce (32 byte) used for the verification MAC.
in	*io_key*	IO protection key for verifying the validation MAC.
out	*is_verified*	Boolean whether or not the message, signature, public key verified.

Returns

ATCA_SUCCESS on verification success or failure, because the command still completed successfully.

13.6.4.107 atcab_verify_invalidate()

```
ATCA_STATUS atcab_verify_invalidate (
             uint16_t key_id,
             const uint8_t * signature,
             const uint8_t * other_data,
             bool * is_verified )
```

Executes the Verify command in Invalidate mode which invalidates a previously validated public key stored in a slot.

This command can only be run after GenKey has been used to create a PubKey digest of the public key to be invalidated in TempKey (mode=0x10).

Parameters

in	*key_id*	Slot containing the public key to be invalidated.
in	*signature*	Signature to be verified. R and S integers in big-endian format. 64 bytes for P256 curve.
in	*other_data*	19 bytes of data used to build the verification message.
out	*is_verified*	Boolean whether or not the message, signature, validation public key verified.

Returns

ATCA_SUCCESS on verification success or failure, because the command still completed successfully.

13.6.4.108 atcab_verify_stored()

```
ATCA_STATUS atcab_verify_stored (
             const uint8_t * message,
             const uint8_t * signature,
             uint16_t key_id,
             bool * is_verified )
```

Executes the Verify command, which verifies a signature (ECDSA verify operation) with a public key stored in the device. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608A device or TempKey for other devices.

Parameters

in	*message*	32 byte message to be verified. Typically the SHA256 hash of the full message.
in	*signature*	Signature to be verified. R and S integers in big-endian format. 64 bytes for P256 curve.
in	*key_id*	Slot containing the public key to be used in the verification.
out	*is_verified*	Boolean whether or not the message, signature, public key verified.

Returns

ATCA_SUCCESS on verification success or failure, because the command still completed successfully.

13.6.4.109 atcab_verify_stored_mac()

```
ATCA_STATUS atcab_verify_stored_mac (
            const uint8_t * message,
            const uint8_t * signature,
            uint16_t key_id,
            const uint8_t * num_in,
            const uint8_t * io_key,
            bool * is_verified )
```

Executes the Verify command with verification MAC, which verifies a signature (ECDSA verify operation) with a public key stored in the device. This function is only available on the ATECC608A.

Parameters

in	*message*	32 byte message to be verified. Typically the SHA256 hash of the full message.
in	*signature*	Signature to be verified. R and S integers in big-endian format. 64 bytes for P256 curve.
in	*key_id*	Slot containing the public key to be used in the verification.
in	*num_in*	System nonce (32 byte) used for the verification MAC.
in	*io_key*	IO protection key for verifying the validation MAC.
out	*is_verified*	Boolean whether or not the message, signature, public key verified.

Returns

ATCA_SUCCESS on verification success or failure, because the command still completed successfully.

13.6.4.110 atcab_verify_validate()

```
ATCA_STATUS atcab_verify_validate (
            uint16_t key_id,
            const uint8_t * signature,
            const uint8_t * other_data,
            bool * is_verified )
```

Executes the Verify command in Validate mode to validate a public key stored in a slot.

This command can only be run after GenKey has been used to create a PubKey digest of the public key to be validated in TempKey (mode=0x10).

Parameters

in	*key_id*	Slot containing the public key to be validated.
in	*signature*	Signature to be verified. R and S integers in big-endian format. 64 bytes for P256 curve.
in	*other_data*	19 bytes of data used to build the verification message.
out	*is_verified*	Boolean whether or not the message, signature, validation public key verified.

Returns

ATCA_SUCCESS on verification success or failure, because the command still completed successfully.

13.6.4.111 atcab_version()

```
ATCA_STATUS atcab_version (
            char * ver_str )
```

basic API methods are all prefixed with atcab_ (CryptoAuthLib Basic) the fundamental premise of the basic API is it is based on a single interface instance and that instance is global, so all basic API commands assume that one global device is the one to operate on.

returns a version string for the CryptoAuthLib release. The format of the version string returned is "yyyymmdd"

Parameters

out	*ver_str*	ptr to space to receive version string

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.112 atcab_wakeup()

```
ATCA_STATUS atcab_wakeup (
            void )
```

wakeup the CryptoAuth device

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.113 atcab_write()

```
ATCA_STATUS atcab_write (
            uint8_t zone,
            uint16_t address,
            const uint8_t * value,
            const uint8_t * mac )
```

Executes the Write command, which writes either one four byte word or a 32-byte block to one of the EEPROM zones on the device. Depending upon the value of the WriteConfig byte for this slot, the data may be required to be encrypted by the system prior to being sent to the device. This command cannot be used to write slots configured as ECC private keys.

Parameters

in	*zone*	Zone/Param1 for the write command.
in	*address*	Address/Param2 for the write command.
in	*value*	Plain-text data to be written or cipher-text for encrypted writes. 32 or 4 bytes depending on bit 7 in the zone.
in	*mac*	MAC required for encrypted writes (32 bytes). Set to NULL if not required.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.114 atcab_write_bytes_zone()

```
ATCA_STATUS atcab_write_bytes_zone (
            uint8_t zone,
            uint16_t slot,
            size_t offset_bytes,
            const uint8_t * data,
            size_t length )
```

Executes the Write command, which writes data into the configuration, otp, or data zones with a given byte offset and length. Offset and length must be multiples of a word (4 bytes).

Config zone must be unlocked for writes to that zone. If data zone is unlocked, only 32-byte writes are allowed to slots and OTP and the offset and length must be multiples of 32 or the write will fail.

Parameters

in	*zone*	Zone to write data to: ATCA_ZONE_CONFIG(0), ATCA_ZONE_OTP(1), or ATCA_ZONE_DATA(2).
in	*slot*	If zone is ATCA_ZONE_DATA(2), the slot number to write to. Ignored for all other zones.
in	*offset_bytes*	Byte offset within the zone to write to. Must be a multiple of a word (4 bytes).
in	*data*	Data to be written.
in	*length*	Number of bytes to be written. Must be a multiple of a word (4 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.115 atcab_write_config_counter()

```
ATCA_STATUS atcab_write_config_counter (
            uint16_t counter_id,
            uint32_t counter_value )
```

Initialize one of the monotonic counters in device with a specific value.

The monotonic counters are stored in the configuration zone using a special format. This encodes a binary count value into the 8 byte encoded value required. Can only be set while the configuration zone is unlocked.

Parameters

in	*counter_id*	Counter to be written.
in	*counter_value*	Counter value to set.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.116 atcab_write_config_zone()

```
ATCA_STATUS atcab_write_config_zone (
            const uint8_t * config_data )
```

Executes the Write command, which writes the configuration zone.

First 16 bytes are skipped as they are not writable. LockValue and LockConfig are also skipped and can only be changed via the Lock command.

This command may fail if UserExtra and/or Selector bytes have already been set to non-zero values.

Parameters

in	*config_data*	Data to the config zone data. This should be 88 bytes for SHA devices and 128 bytes for ECC devices.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.117 atcab_write_enc()

```
ATCA_STATUS atcab_write_enc (
            uint16_t key_id,
            uint8_t block,
            const uint8_t * data,
            const uint8_t * enc_key,
            const uint16_t enc_key_id )
```

Executes the Write command, which performs an encrypted write of a 32 byte block into given slot.

The function takes clear text bytes and encrypts them for writing over the wire. Data zone must be locked and the slot configuration must be set to encrypted write for the block to be successfully written.

Parameters

in	*key_id*	Slot ID to write to.
in	*block*	Index of the 32 byte block to write in the slot.
in	*data*	32 bytes of clear text data to be written to the slot
in	*enc_key*	WriteKey to encrypt with for writing
in	*enc_key_id*	The KeyID of the WriteKey

returns ATCA_SUCCESS on success, otherwise an error code.

13.6.4.118 atcab_write_pubkey()

```
ATCA_STATUS atcab_write_pubkey (
            uint16_t slot,
            const uint8_t * public_key )
```

Uses the write command to write a public key to a slot in the proper format.

Parameters

in	*slot*	Slot number to write. Only slots 8 to 15 are large enough to store a public key.
in	*public_key*	Public key to write into the slot specified. X and Y integers in big-endian format. 64 bytes for P256 curve.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.4.119 atcab_write_zone()

```
ATCA_STATUS atcab_write_zone (
            uint8_t zone,
            uint16_t slot,
            uint8_t block,
            uint8_t offset,
            const uint8_t * data,
            uint8_t len )
```

Executes the Write command, which writes either 4 or 32 bytes of data into a device zone.

Parameters

in	*zone*	Device zone to write to (0=config, 1=OTP, 2=data).
in	*slot*	If writing to the data zone, it is the slot to write to, otherwise it should be 0.
in	*block*	32-byte block to write to.
in	*offset*	4-byte word within the specified block to write to. If performing a 32-byte write, this should be 0.
in	*data*	Data to be written.
in	*len*	Number of bytes to be written. Must be either 4 or 32.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.6.5 Variable Documentation

13.6.5.1 _gDevice

```
ATCADevice _gDevice
```

13.6.5.2 atca_basic_aes_gcm_version [1/2]

```
const char* atca_basic_aes_gcm_version
```

13.6.5.3 atca_basic_aes_gcm_version [2/2]

```
const char* atca_basic_aes_gcm_version = "1.0"
```

13.7 Software crypto methods (atcac_)

These methods provide a software implementation of various crypto algorithms.

Data Structures

- struct atcac_sha1_ctx
- struct atcac_sha2_256_ctx

Macros

- #define ATCA_ECC_P256_FIELD_SIZE (256 / 8)
- #define ATCA_ECC_P256_PRIVATE_KEY_SIZE (ATCA_ECC_P256_FIELD_SIZE)
- #define ATCA_ECC_P256_PUBLIC_KEY_SIZE (ATCA_ECC_P256_FIELD_SIZE * 2)
- #define ATCA_ECC_P256_SIGNATURE_SIZE (ATCA_ECC_P256_FIELD_SIZE * 2)
- #define ATCA_SHA1_DIGEST_SIZE (20)
- #define ATCA_SHA2_256_DIGEST_SIZE (32)

Functions

- int atcac_sw_ecdsa_verify_p256 (const uint8_t msg[ATCA_ECC_P256_FIELD_SIZE], const uint8_t signature[ATCA_ECC_P256_SIGNATURE_SIZE], const uint8_t public_key[ATCA_ECC_P256_PUBLIC_KEY_SIZE])

  return software generated ECDSA verification result and the function is currently not implemented
- int atcac_sw_random (uint8_t *data, size_t data_size)

  return software generated random number and the function is currently not implemented
- int atcac_sw_sha1_init (atcac_sha1_ctx *ctx)

  Initialize context for performing SHA1 hash in software.
- int atcac_sw_sha1_update (atcac_sha1_ctx *ctx, const uint8_t *data, size_t data_size)

  Add arbitrary data to a SHA1 hash.
- int atcac_sw_sha1_finish (atcac_sha1_ctx *ctx, uint8_t digest[ATCA_SHA1_DIGEST_SIZE])

  Complete the SHA1 hash in software and return the digest.
- int atcac_sw_sha1 (const uint8_t *data, size_t data_size, uint8_t digest[ATCA_SHA1_DIGEST_SIZE])

  Perform SHA1 hash of data in software.
- int atcac_sw_sha2_256_init (atcac_sha2_256_ctx *ctx)

  initializes the SHA256 software
- int atcac_sw_sha2_256_update (atcac_sha2_256_ctx *ctx, const uint8_t *data, size_t data_size)

  updates the running hash with the next block of data, called iteratively for the entire stream of data to be hashed using the SHA256 software
- int atcac_sw_sha2_256_finish (atcac_sha2_256_ctx *ctx, uint8_t digest[ATCA_SHA2_256_DIGEST_SIZE])

  completes the final SHA256 calculation and returns the final digest/hash
- int atcac_sw_sha2_256 (const uint8_t *data, size_t data_size, uint8_t digest[ATCA_SHA2_256_DIGEST_SIZE])

  single call convenience function which computes Hash of given data using SHA256 software

13.7.1 Detailed Description

These methods provide a software implementation of various crypto algorithms.

13.7.2 Macro Definition Documentation

13.7.2.1 ATCA_ECC_P256_FIELD_SIZE

```
#define ATCA_ECC_P256_FIELD_SIZE (256 / 8)
```

13.7.2.2 ATCA_ECC_P256_PRIVATE_KEY_SIZE

```
#define ATCA_ECC_P256_PRIVATE_KEY_SIZE (ATCA_ECC_P256_FIELD_SIZE)
```

13.7.2.3 ATCA_ECC_P256_PUBLIC_KEY_SIZE

```
#define ATCA_ECC_P256_PUBLIC_KEY_SIZE (ATCA_ECC_P256_FIELD_SIZE * 2)
```

13.7.2.4 ATCA_ECC_P256_SIGNATURE_SIZE

```
#define ATCA_ECC_P256_SIGNATURE_SIZE (ATCA_ECC_P256_FIELD_SIZE * 2)
```

13.7.2.5 ATCA_SHA1_DIGEST_SIZE

```
#define ATCA_SHA1_DIGEST_SIZE (20)
```

13.7.2.6 ATCA_SHA2_256_DIGEST_SIZE

```
#define ATCA_SHA2_256_DIGEST_SIZE (32)
```

13.7.3 Function Documentation

13.7.3.1 atcac_sw_ecdsa_verify_p256()

```
int atcac_sw_ecdsa_verify_p256 (
            const uint8_t msg[ATCA_ECC_P256_FIELD_SIZE],
            const uint8_t signature[ATCA_ECC_P256_SIGNATURE_SIZE],
            const uint8_t public_key[ATCA_ECC_P256_PUBLIC_KEY_SIZE] )
```

return software generated ECDSA verification result and the function is currently not implemented

Parameters

in	*msg*	ptr to message or challenge
in	*signature*	ptr to the signature to verify
in	*public_key*	ptr to public key of device which signed the challenge return ATCA_UNIMPLEMENTED , as the function is currently not implemented

13.7.3.2 atcac_sw_random()

```
int atcac_sw_random (
            uint8_t * data,
            size_t data_size )
```

return software generated random number and the function is currently not implemented

Parameters

out	*data*	ptr to space to receive the random number
in	*data_size*	size of data buffer return ATCA_UNIMPLEMENTED , as the function is not implemented

13.7.3.3 atcac_sw_sha1()

```
int atcac_sw_sha1 (
            const uint8_t * data,
            size_t data_size,
            uint8_t digest[ATCA_SHA1_DIGEST_SIZE] )
```

Perform SHA1 hash of data in software.

Parameters

in	*data*	Data to be hashed
in	*data_size*	Data size in bytes
out	*digest*	Digest is returned here (20 bytes)

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.7.3.4 atcac_sw_sha1_finish()

```
int atcac_sw_sha1_finish (
            atcac_sha1_ctx * ctx,
            uint8_t digest[ATCA_SHA1_DIGEST_SIZE] )
```

Complete the SHA1 hash in software and return the digest.

Parameters

in	*ctx*	Hash context
out	*digest*	Digest is returned here (20 bytes)

Returns

ATCA_SUCCESS

13.7.3.5 atcac_sw_sha1_init()

```
int atcac_sw_sha1_init (
            atcac_sha1_ctx * ctx )
```

Initialize context for performing SHA1 hash in software.

Parameters

in	*ctx*	Hash context

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.7.3.6 atcac_sw_sha1_update()

```
int atcac_sw_sha1_update (
            atcac_sha1_ctx * ctx,
            const uint8_t * data,
            size_t data_size )
```

Add arbitrary data to a SHA1 hash.

Parameters

in	*ctx*	Hash context
in	*data*	Data to be added to the hash
in	*data_size*	Data size in bytes

Returns

ATCA_SUCCESS

13.7.3.7 atcac_sw_sha2_256()

```
int atcac_sw_sha2_256 (
            const uint8_t * data,
            size_t data_size,
            uint8_t digest[ATCA_SHA2_256_DIGEST_SIZE] )
```

single call convenience function which computes Hash of given data using SHA256 software

Parameters

in	*data*	pointer to stream of data to hash
in	*data_size*	size of data stream to hash
out	*digest*	result

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.7.3.8 atcac_sw_sha2_256_finish()

```
int atcac_sw_sha2_256_finish (
            atcac_sha2_256_ctx * ctx,
            uint8_t digest[ATCA_SHA2_256_DIGEST_SIZE] )
```

completes the final SHA256 calculation and returns the final digest/hash

Parameters

in	*ctx*	ptr to context data structure
out	*digest*	receives the computed digest of the SHA 256

Returns

ATCA_SUCCESS

13.7.3.9 atcac_sw_sha2_256_init()

```
int atcac_sw_sha2_256_init (
            atcac_sha2_256_ctx * ctx )
```

initializes the SHA256 software

Parameters

in	*ctx*	ptr to context data structure

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.7.3.10 atcac_sw_sha2_256_update()

```
int atcac_sw_sha2_256_update (
            atcac_sha2_256_ctx * ctx,
            const uint8_t * data,
            size_t data_size )
```

updates the running hash with the next block of data, called iteratively for the entire stream of data to be hashed using the SHA256 software

Parameters

in	*ctx*	ptr to SHA context data structure
in	*data*	ptr to next block of data to hash
in	*data_size*	size amount of data to hash in the given block, in bytes

Returns

ATCA_SUCCESS

13.8 Hardware abstraction layer (hal_)

These methods define the hardware abstraction layer for communicating with a CryptoAuth device.

Data Structures

- struct ATCAHAL_t
  an intermediary data structure to allow the HAL layer to point the standard API functions used by the upper layers to the HAL implementation for the interface. This isolates the upper layers and loosely couples the ATCAIface object from the physical implementation.
- struct atcahid
- struct atcaI2Cmaster
  this is the hal_data for ATCA HAL created using ASF
- struct cdc_device
- struct atcacdc
- struct hid_device
- struct DRV_I2C_Object
- struct atcaSWImaster
  This is the hal_data for ATCA HAL.

Macros

- #define HID_DEVICES_MAX 10
- #define HID_PACKET_MAX 512
- #define MAX_I2C_BUSES 1
- #define MAX_I2C_BUSES 6
- #define MAX_I2C_BUSES 2
- #define max(a, b) (((a) > (b)) ? (a) : (b))
- #define min(a, b) (((a) < (b)) ? (a) : (b))
- #define CDC_DEVICES_MAX 10
- #define CDC_BUFFER_MAX 1024
- #define INVALID_HANDLE_VALUE ((int)(-1))
- #define HID_DEVICES_MAX 10
- #define HID_PACKET_MAX 512
- #define GetSystemClock() (80000000ul)
- #define GetPeripheralClock() (GetSystemClock() / (1 << OSCCONbits.PBDIV))
- #define GetInstructionClock() (GetSystemClock())
- #define MAX_I2C_BUSES 4
- #define CPU_CLOCK (80000000UL)
- #define us_SCALE ((CPU_CLOCK / 2) / 1000000)
- #define HARMONY_I2C_DRIVER 1
- #define MAX_I2C_BUSES 3
- #define GetSystemClock() (200000000UL)/∗ Fcy = 200MHz ∗/
- #define us_SCALE (GetSystemClock() / 2000000)
- #define MAX_I2C_BUSES 2
- #define MAX_I2C_BUSES 2
- #define MAX_I2C_BUSES 6
- #define MAX_I2C_BUSES 2
- #define MAX_I2C_BUSES 3
- #define SWI_WAKE_TOKEN ((uint8_t)0x00)
  flag preceding a command
- #define SWI_FLAG_CMD ((uint8_t)0x77)

flag preceding a command
- #define SWI_FLAG_TX ((uint8_t)0x88)

  flag requesting a response
- #define SWI_FLAG_IDLE ((uint8_t)0xBB)

  flag requesting to go into Idle mode
- #define SWI_FLAG_SLEEP ((uint8_t)0xCC)

  flag requesting to go into Sleep mode
- #define MAX_I2C_BUSES 3
- #define HID_GUID { 0x4d1e55b2, 0xf16f, 0x11cf, 0x88, 0xcb, 0x00, 0x11, 0x11, 0x00, 0x00, 0x30 }
- #define HID_DEVICES_MAX 10
- #define HID_PACKET_MAX 512
- #define MAX_I2C_BUSES 4
- #define KIT_MAX_SCAN_COUNT 4
- #define KIT_MAX_TX_BUF 32
- #define KIT_TX_WRAP_SIZE (7)
- #define KIT_MSG_SIZE (32)
- #define KIT_RX_WRAP_SIZE (KIT_MSG_SIZE + 6)
- #define MAX_SWI_BUSES 1
- #define RECEIVE_MODE 0
- #define TRANSMIT_MODE 1
- #define RX_DELAY 10
- #define TX_DELAY 90
- #define MAX_SWI_BUSES 6
- #define RECEIVE_MODE 0
- #define TRANSMIT_MODE 1
- #define RX_DELAY 10
- #define TX_DELAY 90
- #define DEBUG_PIN_1 EXT2_PIN_5
- #define DEBUG_PIN_2 EXT2_PIN_6
- #define MAX_SWI_BUSES 6
- #define RECEIVE_MODE 0
- #define TRANSMIT_MODE 1
- #define RX_DELAY 10
- #define TX_DELAY 93
- #define MAX_SWI_BUSES 6
- #define RECEIVE_MODE 0
- #define TRANSMIT_MODE 1
- #define RX_DELAY 10
- #define TX_DELAY 90

Typedefs

- typedef struct atcahid atcahid_t
- typedef struct atcaI2Cmaster ATCAI2CMaster_t

  this is the hal_data for ATCA HAL created using ASF
- typedef struct atcaI2Cmaster ATCAI2CMaster_t

  This is the hal_data for ATCA HAL.
- typedef struct atcaI2Cmaster ATCAI2CMaster_t

  this is the hal_data for ATCA HAL for Atmel START SERCOM
- typedef struct atcaI2Cmaster ATCAI2CMaster_t
- typedef int HANDLE
- typedef struct cdc_device cdc_device_t
- typedef struct atcacdc atcacdc_t

- typedef struct hid_device hid_device_t
- typedef struct atcahid atcahid_t
- typedef struct atcaI2Cmaster ATCAI2CMaster_t

  this is the hal_data for ATCA HAL
- typedef struct atcaI2Cmaster ATCAI2CMaster_t

  this is the hal_data for ATCA HAL
- typedef struct atcaI2Cmaster ATCAI2CMaster_t

  this is the hal_data for ATCA HAL
- typedef struct atcaI2Cmaster ATCAI2CMaster_t

  this is the hal_data for ATCA HAL for ASF
- typedef struct atcaI2Cmaster ATCAI2CMaster_t

  this is the hal_data for ATCA HAL for ASF SERCOM
- typedef struct atcaI2Cmaster ATCAI2CMaster_t

  this is the hal_data for ATCA HAL
- typedef struct atcaI2Cmaster ATCAI2CMaster_t

  this is the hal_data for ATCA HAL for ASF SERCOM
- typedef struct atcaSWImaster ATCASWIMaster_t

  This is the hal_data for ATCA HAL.
- typedef struct atcaI2Cmaster ATCAI2CMaster_t

  this is the hal_data for ATCA HAL for ASF SERCOM
- typedef struct hid_device hid_device_t
- typedef struct atcahid atcahid_t
- typedef struct atcaI2Cmaster ATCAI2CMaster_t

  this is the hal_data for ATCA HAL created using ASF
- typedef struct atcaSWImaster ATCASWIMaster_t

  this is the hal_data for ATCA HAL for SWI UART
- typedef struct atcaSWImaster ATCASWIMaster_t

  this is the hal_data for ATCA HAL for ASF SERCOM
- typedef struct atcaSWImaster ATCASWIMaster_t

  this is the hal_data for ATCA HAL for ASF SERCOM
- typedef struct atcaSWImaster ATCASWIMaster_t

  this is the hal_data for ATCA HAL for SWI UART

Enumerations

- enum i2c_read_write_flag { I2C_WRITE = (uint8_t)0x00, I2C_READ = (uint8_t)0x01 }

  This enumeration lists flags for I2C read or write addressing.
- enum swi_flag { SWI_FLAG_CMD = (uint8_t)0x77, SWI_FLAG_TX = (uint8_t)0x88, SWI_FLAG_IDLE = (uint8_t)0xBB, SWI_FLAG_SLEEP = (uint8_t)0xCC }

  This enumeration lists flags for SWI.

Functions

- ATCA_STATUS hal_iface_init (ATCAIfaceCfg ∗, ATCAHAL_t ∗hal)

  Standard HAL API for ATCA to initialize a physical interface.
- ATCA_STATUS hal_iface_release (ATCAIfaceType, void ∗hal_data)

  releases a physical interface, HAL knows how to interpret hal_data
- ATCA_STATUS hal_check_wake (const uint8_t ∗response, int response_size)

  Utility function for hal_wake to check the reply.
- void atca_delay_us (uint32_t delay)

Timer API implemented at the HAL level.
- void atca_delay_10us (uint32_t delay)

    This function delays for a number of tens of microseconds.
- void atca_delay_ms (uint32_t delay)

    This function delays for a number of milliseconds.
- ATCA_STATUS hal_create_mutex (void ∗∗ppMutex, char ∗pName)

    Optional hal interfaces.
- ATCA_STATUS hal_destroy_mutex (void ∗pMutex)
- ATCA_STATUS hal_lock_mutex (void ∗pMutex)
- ATCA_STATUS hal_unlock_mutex (void ∗pMutex)
- ATCA_STATUS hal_kit_hid_discover_buses (int i2c_buses[ ], int max_buses)

    discover cdc buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge
- ATCA_STATUS hal_kit_hid_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

    discover any CryptoAuth devices on a given logical bus number
- ATCA_STATUS hal_kit_hid_init (void ∗hal, ATCAIfaceCfg ∗cfg)

    HAL implementation of Kit USB HID init.
- ATCA_STATUS hal_kit_hid_post_init (ATCAIface iface)

    HAL implementation of Kit HID post init.
- ATCA_STATUS kit_phy_send (ATCAIface iface, uint8_t ∗txdata, int txlength)

    HAL implementation of send over USB HID.
- ATCA_STATUS kit_phy_receive (ATCAIface iface, uint8_t ∗rxdata, int ∗rxsize)

    HAL implementation of kit protocol send over USB HID.
- ATCA_STATUS kit_phy_num_found (int8_t ∗num_found)

    Number of USB HID devices found.
- ATCA_STATUS hal_kit_hid_send (ATCAIface iface, uint8_t ∗txdata, int txlength)

    HAL implementation of kit protocol send over USB HID.
- ATCA_STATUS hal_kit_hid_receive (ATCAIface iface, uint8_t ∗rxdata, uint16_t ∗rxsize)

    HAL implementation of send over USB HID.
- ATCA_STATUS hal_kit_hid_wake (ATCAIface iface)

    Call the wake for kit protocol.
- ATCA_STATUS hal_kit_hid_idle (ATCAIface iface)

    Call the idle for kit protocol.
- ATCA_STATUS hal_kit_hid_sleep (ATCAIface iface)

    Call the sleep for kit protocol.
- ATCA_STATUS hal_kit_hid_release (void ∗hal_data)

    Close the physical port for HID.
- ATCA_STATUS hal_i2c_discover_buses (int i2c_buses[ ], int max_buses)

    discover i2c buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the prior knowledge
- ATCA_STATUS hal_i2c_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

    discover any CryptoAuth devices on a given logical bus number
- ATCA_STATUS hal_i2c_init (void ∗hal, ATCAIfaceCfg ∗cfg)

    initialize an I2C interface using given config
- ATCA_STATUS hal_i2c_post_init (ATCAIface iface)

    HAL implementation of I2C post init.
- ATCA_STATUS hal_i2c_send (ATCAIface iface, uint8_t ∗txdata, int txlength)

    HAL implementation of I2C send over ASF.
- ATCA_STATUS hal_i2c_receive (ATCAIface iface, uint8_t ∗rxdata, uint16_t ∗rxlength)

    HAL implementation of I2C receive function for ASF I2C.
- void change_i2c_speed (ATCAIface iface, uint32_t speed)

method to change the bus speed of I2C
- ATCA_STATUS hal_i2c_wake (ATCAIface iface)

  wake up CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_idle (ATCAIface iface)

  idle CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_sleep (ATCAIface iface)

  sleep CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_release (void ∗hal_data)

  manages reference count on given bus and releases resource if no more refences exist
- ATCA_STATUS hal_cdc_discover_buses (int cdc_buses[ ], int max_buses)

  discover cdc buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge.This function is currently not implemented.
- ATCA_STATUS hal_cdc_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

  discover any CryptoAuth devices on a given logical bus number
- ATCA_STATUS hal_kit_cdc_init (void ∗hal, ATCAIfaceCfg ∗cfg)

  HAL implementation of Kit USB CDC init.
- ATCA_STATUS hal_kit_cdc_post_init (ATCAIface iface)

  HAL implementation of Kit USB CDC post init.
- ATCA_STATUS kit_phy_send (ATCAIface iface, const char ∗txdata, int txlength)

  HAL implementation of kit protocol send .It is called by the top layer.
- ATCA_STATUS kit_phy_receive (ATCAIface iface, char ∗rxdata, int ∗rxsize)

  HAL implementation of kit protocol receive data.It is called by the top layer.
- ATCA_STATUS hal_kit_phy_num_found (int8_t ∗num_found)

  Number of USB CDC devices found.
- ATCA_STATUS hal_kit_cdc_send (ATCAIface iface, uint8_t ∗txdata, int txlength)

  HAL implementation of kit protocol send over USB CDC.
- ATCA_STATUS hal_kit_cdc_receive (ATCAIface iface, uint8_t ∗rxdata, uint16_t ∗rxsize)

  HAL implementation of kit protocol receive over USB CDC.
- ATCA_STATUS hal_kit_cdc_wake (ATCAIface iface)

  Call the wake for kit protocol over USB CDC.
- ATCA_STATUS hal_kit_cdc_idle (ATCAIface iface)

  Call the idle for kit protocol over USB CDC.
- ATCA_STATUS hal_kit_cdc_sleep (ATCAIface iface)

  Call the sleep for kit protocol over USB CDC.
- ATCA_STATUS hal_kit_cdc_release (void ∗hal_data)

  Close the physical port for CDC over USB CDC.
- ATCA_STATUS hal_kit_cdc_discover_buses (int cdc_buses[ ], int max_buses)

  discover cdc buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge.This function is currently not implemented.
- ATCA_STATUS hal_kit_cdc_discover_devices (int bus_num, ATCAIfaceCfg ∗cfg, int ∗found)

  discover any CryptoAuth devices on a given logical bus number
- void i2c_write (I2C_MODULE i2c_id, uint8_t address, uint8_t ∗data, int len)
- ATCA_STATUS i2c_read (I2C_MODULE i2c_id, uint8_t address, uint8_t ∗data, uint16_t len)
- void delay_us (UINT32 delay)
- void delay_us (uint32_t delay)
- ATCA_STATUS hal_swi_discover_buses (int swi_buses[ ], int max_buses)

  discover swi buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application.This function is currently not supported. of the a-priori knowledge
- ATCA_STATUS hal_swi_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

  discover any CryptoAuth devices on a given logical bus number.This function is curently not supported.
- ATCA_STATUS hal_swi_init (void ∗hal, ATCAIfaceCfg ∗cfg)

hal_swi_init manages requests to initialize a physical interface. It manages use counts so when an interface has released the physical layer, it will disable the interface for some other use. You can have multiple ATCAIFace instances using the same bus, and you can have multiple ATCAIFace instances on multiple swi buses, so hal_swi_init manages these things and ATCAIFace is abstracted from the physical details.

- ATCA_STATUS hal_swi_post_init (ATCAIface iface)

    HAL implementation of SWI post init.
- ATCA_STATUS hal_swi_send (ATCAIface iface, uint8_t ∗txdata, int txlength)

    Send byte(s) via SWI.
- ATCA_STATUS hal_swi_receive (ATCAIface iface, uint8_t ∗rxdata, uint16_t ∗rxlength)

    Receive byte(s) via SWI.
- ATCA_STATUS hal_swi_wake (ATCAIface iface)

    Send Wake flag via SWI.
- ATCA_STATUS hal_swi_idle (ATCAIface iface)

    Send Idle flag via SWI.
- ATCA_STATUS hal_swi_sleep (ATCAIface iface)

    Send Sleep flag via SWI.
- ATCA_STATUS hal_swi_release (void ∗hal_data)

    Manages reference count on given bus and releases resource if no more reference(s) exist.
- ATCA_STATUS hal_swi_send_flag (ATCAIface iface, uint8_t data)

    HAL implementation of SWI send one byte over UART.
- char ∗ strnchr (const char ∗s, size_t count, int c)
- const char ∗ kit_id_from_devtype (ATCADeviceType devtype)
- const char ∗ kit_interface_from_kittype (ATCAKitType kittype)
- ATCA_STATUS kit_init (ATCAIface iface)

    HAL implementation of kit protocol init. This function calls back to the physical protocol to send the bytes.
- ATCA_STATUS kit_send (ATCAIface iface, const uint8_t ∗txdata, int txlength)

    HAL implementation of kit protocol send. This function calls back to the physical protocol to send the bytes.
- ATCA_STATUS kit_receive (ATCAIface iface, uint8_t ∗rxdata, uint16_t ∗rxsize)

    HAL implementation to receive bytes and unwrap from kit protocol. This function calls back to the physical protocol to receive the bytes.
- ATCA_STATUS kit_wake (ATCAIface iface)

    Call the wake for kit protocol.
- ATCA_STATUS kit_idle (ATCAIface iface)

    Call the idle for kit protocol.
- ATCA_STATUS kit_sleep (ATCAIface iface)

    Call the sleep for kit protocol.
- ATCA_STATUS kit_wrap_cmd (const uint8_t ∗txdata, int txlen, char ∗pkitcmd, int ∗nkitcmd, char target)

    Wrap binary bytes in ascii kit protocol.
- ATCA_STATUS kit_parse_rsp (const char ∗pkitbuf, int nkitbuf, uint8_t ∗kitstatus, uint8_t ∗rxdata, int ∗datasize)

    Parse the response ascii from the kit.
- ATCA_STATUS swi_uart_init (ATCASWIMaster_t ∗instance)

    Implementation of SWI UART init.
- ATCA_STATUS swi_uart_deinit (ATCASWIMaster_t ∗instance)

    Implementation of SWI UART deinit.
- void swi_uart_setbaud (ATCASWIMaster_t ∗instance, uint32_t baudrate)

    implementation of SWI UART change baudrate.
- void swi_uart_mode (ATCASWIMaster_t ∗instance, uint8_t mode)

    implementation of SWI UART change mode.
- void swi_uart_discover_buses (int swi_uart_buses[ ], int max_buses)

    discover UART buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge

- ATCA_STATUS swi_uart_send_byte (ATCASWIMaster_t ∗instance, uint8_t data)

  HAL implementation of SWI UART send byte over ASF. This function send one byte over UART.
- ATCA_STATUS swi_uart_receive_byte (ATCASWIMaster_t ∗instance, uint8_t ∗data)

  HAL implementation of SWI UART receive bytes over ASF. This function receive one byte over UART.

Variables

- atcahid_t _gHid
- atcacdc_t _gCdc
- char ∗ dev = "/dev/ttyACM0"
- int speed = B115200
- atcahid_t _gHid
- atcahid_t _gHid
- struct port_config pin_conf

13.8.1 Detailed Description

These methods define the hardware abstraction layer for communicating with a CryptoAuth device.

These methods define the hardware abstraction layer for communicating with a CryptoAuth device using SWI interface.

These methods define the hardware abstraction layer for communicating with a CryptoAuth device using SWI Interface.

These methods define the hardware abstraction layer for communicating with a CryptoAuth device using SWI bit banging.

< Uncomment when debugging

These methods define the hardware abstraction layer for communicating with a CryptoAuth device using I2C driver of Harmony.

These methods define the hardware abstraction layer for communicating with a CryptoAuth device using I2C bit banging.

These methods define the hardware abstraction layer for communicating with a CryptoAuth device using I2C driver of ASF.

13.8.2 Macro Definition Documentation

13.8.2.1 CDC_BUFFER_MAX

```
#define CDC_BUFFER_MAX 1024
```


13.8.2.2 CDC_DEVICES_MAX

```
#define CDC_DEVICES_MAX 10
```

13.8.2.3 CPU_CLOCK

```
#define CPU_CLOCK (80000000UL)
```

13.8.2.4 DEBUG_PIN_1

```
#define DEBUG_PIN_1 EXT2_PIN_5
```

13.8.2.5 DEBUG_PIN_2

```
#define DEBUG_PIN_2 EXT2_PIN_6
```

13.8.2.6 GetInstructionClock

```
#define GetInstructionClock( ) (GetSystemClock())
```

13.8.2.7 GetPeripheralClock

```
#define GetPeripheralClock( ) (GetSystemClock() / (1 << OSCCONbits.PBDIV))
```

13.8.2.8 GetSystemClock [1/2]

```
#define GetSystemClock( ) (80000000ul)
```

13.8.2.9 GetSystemClock [2/2]

```
#define GetSystemClock( ) (200000000UL)/* Fcy = 200MHz */
```

13.8.2.10 HARMONY_I2C_DRIVER

```
#define HARMONY_I2C_DRIVER 1
```

13.8.2.11 HID_DEVICES_MAX [1/3]

```
#define HID_DEVICES_MAX 10
```

13.8.2.12 HID_DEVICES_MAX [2/3]

```
#define HID_DEVICES_MAX 10
```

13.8.2.13 HID_DEVICES_MAX [3/3]

```
#define HID_DEVICES_MAX 10
```

13.8.2.14 HID_GUID

```
#define HID_GUID { 0x4d1e55b2, 0xf16f, 0x11cf, 0x88, 0xcb, 0x00, 0x11, 0x11, 0x00, 0x00, 0x30
}
```

13.8.2.15 HID_PACKET_MAX [1/3]

```
#define HID_PACKET_MAX 512
```

13.8.2.16 HID_PACKET_MAX [2/3]

```
#define HID_PACKET_MAX 512
```

13.8.2.17 HID_PACKET_MAX [3/3]

```
#define HID_PACKET_MAX 512
```

13.8.2.18 INVALID_HANDLE_VALUE

```
#define INVALID_HANDLE_VALUE ((int)(-1))
```

13.8.2.19 KIT_MAX_SCAN_COUNT

```
#define KIT_MAX_SCAN_COUNT 4
```

13.8.2.20 KIT_MAX_TX_BUF

```
#define KIT_MAX_TX_BUF 32
```

13.8.2.21 KIT_MSG_SIZE

```
#define KIT_MSG_SIZE (32)
```

13.8.2.22 KIT_RX_WRAP_SIZE

```
#define KIT_RX_WRAP_SIZE (KIT_MSG_SIZE + 6)
```

13.8.2.23 KIT_TX_WRAP_SIZE

```
#define KIT_TX_WRAP_SIZE (7)
```

13.8.2.24 max

```
#define max(
              a,
              b ) (((a) > (b)) ?  (a) :  (b))
```

13.8.2.25 MAX_I2C_BUSES [1/12]

```
#define MAX_I2C_BUSES 2
```

13.8.2.26 MAX_I2C_BUSES [2/12]

```
#define MAX_I2C_BUSES 2
```

13.8.2.27 MAX_I2C_BUSES [3/12]

```
#define MAX_I2C_BUSES 6
```

13.8.2.28 MAX_I2C_BUSES [4/12]

```
#define MAX_I2C_BUSES 6
```

13.8.2.29 MAX_I2C_BUSES [5/12]

```
#define MAX_I2C_BUSES 3
```

13.8.2.30 MAX_I2C_BUSES [6/12]

```
#define MAX_I2C_BUSES 3
```

13.8.2.31 MAX_I2C_BUSES [7/12]

```
#define MAX_I2C_BUSES 4
```

13.8.2.32 MAX_I2C_BUSES [8/12]

```
#define MAX_I2C_BUSES 3
```

13.8.2.33 MAX_I2C_BUSES [9/12]

```
#define MAX_I2C_BUSES 2
```

13.8.2.34 MAX_I2C_BUSES [10/12]

```
#define MAX_I2C_BUSES 1
```

13.8.2.35 MAX_I2C_BUSES [11/12]

```
#define MAX_I2C_BUSES 2
```

13.8.2.36 MAX_I2C_BUSES [12/12]

```
#define MAX_I2C_BUSES 4
```

13.8.2.37 MAX_SWI_BUSES [1/4]

```
#define MAX_SWI_BUSES 6
```

- this HAL implementation assumes you've included the ASF SERCOM UART libraries in your project, otherwise, the HAL layer will not compile because the ASF UART drivers are a dependency *

13.8.2.38 MAX_SWI_BUSES [2/4]

```
#define MAX_SWI_BUSES 1
```

- this HAL implementation assumes you've included the ASF UART libraries in your project, otherwise, the HAL layer will not compile because the ASF UART drivers are a dependency *

13.8.2.39 MAX_SWI_BUSES [3/4]

```
#define MAX_SWI_BUSES 6
```

- this HAL implementation assumes you've included the ASF UART libraries in your project, otherwise, the HAL layer will not compile because the ASF UART drivers are a dependency *

13.8.2.40 MAX_SWI_BUSES [4/4]

```
#define MAX_SWI_BUSES 6
```

- this HAL implementation assumes you've included the ASF SERCOM UART libraries in your project, otherwise, the HAL layer will not compile because the ASF UART drivers are a dependency *

13.8.2.41 min

```
#define min(
              a,
              b ) (((a) < (b)) ? (a) : (b))
```

13.8.2.42 RECEIVE_MODE [1/4]

```
#define RECEIVE_MODE 0
```

13.8.2.43 RECEIVE_MODE [2/4]

```
#define RECEIVE_MODE 0
```

13.8.2.44 RECEIVE_MODE [3/4]

```
#define RECEIVE_MODE 0
```

13.8.2.45 RECEIVE_MODE [4/4]

```
#define RECEIVE_MODE 0
```

13.8.2.46 RX_DELAY [1/4]

```
#define RX_DELAY 10
```

13.8.2.47 RX_DELAY [2/4]

```
#define RX_DELAY 10
```

13.8.2.48 RX_DELAY [3/4]

```
#define RX_DELAY 10
```

13.8.2.49 RX_DELAY [4/4]

```
#define RX_DELAY 10
```

13.8.2.50 SWI_FLAG_CMD

```
#define SWI_FLAG_CMD ((uint8_t)0x77)
```

flag preceding a command

13.8.2.51 SWI_FLAG_IDLE

```
#define SWI_FLAG_IDLE ((uint8_t)0xBB)
```

flag requesting to go into Idle mode

13.8.2.52 SWI_FLAG_SLEEP

```
#define SWI_FLAG_SLEEP ((uint8_t)0xCC)
```

flag requesting to go into Sleep mode

13.8.2.53 SWI_FLAG_TX

```
#define SWI_FLAG_TX ((uint8_t)0x88)
```

flag requesting a response

13.8.2.54 SWI_WAKE_TOKEN

```
#define SWI_WAKE_TOKEN ((uint8_t)0x00)
```

flag preceding a command

13.8.2.55 TRANSMIT_MODE [1/4]

```
#define TRANSMIT_MODE 1
```

13.8.2.56 TRANSMIT_MODE [2/4]

```
#define TRANSMIT_MODE 1
```

13.8.2.57 TRANSMIT_MODE [3/4]

```
#define TRANSMIT_MODE 1
```

13.8.2.58 TRANSMIT_MODE [4/4]

```
#define TRANSMIT_MODE 1
```

13.8.2.59 TX_DELAY [1/4]

```
#define TX_DELAY 90
```

13.8.2.60 TX_DELAY [2/4]

```
#define TX_DELAY 90
```

13.8.2.61 TX_DELAY [3/4]

```
#define TX_DELAY 90
```

13.8.2.62 TX_DELAY [4/4]

```
#define TX_DELAY 93
```

13.8.2.63 us_SCALE [1/2]

```
#define us_SCALE ((CPU_CLOCK / 2) / 1000000)
```

13.8.2.64 us_SCALE [2/2]

```
#define us_SCALE (GetSystemClock() / 2000000)
```

13.8.3 Typedef Documentation

13.8.3.1 atcacdc_t

```
typedef struct atcacdc atcacdc_t
```

13.8.3.2 atcahid_t [1/3]

```
typedef struct atcahid atcahid_t
```

13.8.3.3 atcahid_t [2/3]

```
typedef struct atcahid atcahid_t
```

13.8.3.4 atcahid_t [3/3]

```
typedef struct atcahid atcahid_t
```

13.8.3.5 ATCAI2CMaster_t [1/13]

```
typedef struct atcaI2Cmaster ATCAI2CMaster_t
```

13.8.3.6 ATCAI2CMaster_t [2/13]

```
typedef struct atcaI2Cmaster ATCAI2CMaster_t
```

this is the hal_data for ATCA HAL created using ASF

13.8.3.7 ATCAI2CMaster_t [3/13]

```
typedef struct atcaI2Cmaster ATCAI2CMaster_t
```

this is the hal_data for ATCA HAL for ASF

13.8.3.8 ATCAI2CMaster_t [4/13]

```
typedef struct atcaI2Cmaster ATCAI2CMaster_t
```

this is the hal_data for ATCA HAL for Atmel START SERCOM

13.8.3.9 ATCAI2CMaster_t [5/13]

```
typedef struct atcaI2Cmaster ATCAI2CMaster_t
```

this is the hal_data for ATCA HAL created using ASF

13.8.3.10 ATCAI2CMaster_t [6/13]

```
typedef struct atcaI2Cmaster ATCAI2CMaster_t
```

this is the hal_data for ATCA HAL

13.8.3.11 ATCAI2CMaster_t [7/13]

```
typedef struct atcaI2Cmaster ATCAI2CMaster_t
```

this is the hal_data for ATCA HAL for ASF SERCOM

13.8.3.12 ATCAI2CMaster_t [8/13]

```
typedef struct atcaI2Cmaster ATCAI2CMaster_t
```

this is the hal_data for ATCA HAL for ASF SERCOM

13.8.3.13 ATCAI2CMaster_t [9/13]

```
typedef struct atcaI2Cmaster ATCAI2CMaster_t
```

this is the hal_data for ATCA HAL for ASF SERCOM

13.8.3.14 ATCAI2CMaster_t [10/13]

```
typedef struct atcaI2Cmaster ATCAI2CMaster_t
```

this is the hal_data for ATCA HAL

13.8.3.15 ATCAI2CMaster_t [11/13]

```
typedef struct atcaI2Cmaster ATCAI2CMaster_t
```

this is the hal_data for ATCA HAL

13.8.3.16 ATCAI2CMaster_t [12/13]

```
typedef struct atcaI2Cmaster ATCAI2CMaster_t
```

this is the hal_data for ATCA HAL

13.8.3.17 ATCAI2CMaster_t [13/13]

```
typedef struct atcaI2Cmaster ATCAI2CMaster_t
```

This is the hal_data for ATCA HAL.

13.8.3.18 ATCASWIMaster_t [1/5]

```
typedef struct atcaSWImaster ATCASWIMaster_t
```

this is the hal_data for ATCA HAL for SWI UART

13.8.3.19 ATCASWIMaster_t [2/5]

```
typedef struct atcaSWImaster ATCASWIMaster_t
```

this is the hal_data for ATCA HAL for SWI UART

13.8.3.20 ATCASWIMaster_t [3/5]

`typedef struct atcaSWImaster ATCASWIMaster_t`

this is the hal_data for ATCA HAL for ASF SERCOM

13.8.3.21 ATCASWIMaster_t [4/5]

`typedef struct atcaSWImaster ATCASWIMaster_t`

this is the hal_data for ATCA HAL for ASF SERCOM

13.8.3.22 ATCASWIMaster_t [5/5]

`typedef struct atcaSWImaster ATCASWIMaster_t`

This is the hal_data for ATCA HAL.

13.8.3.23 cdc_device_t

`typedef struct cdc_device cdc_device_t`

13.8.3.24 HANDLE

`typedef int HANDLE`

13.8.3.25 hid_device_t [1/2]

`typedef struct hid_device hid_device_t`

13.8.3.26 hid_device_t [2/2]

`typedef struct hid_device hid_device_t`

13.8.4 Enumeration Type Documentation

13.8.4.1 i2c_read_write_flag

`enum i2c_read_write_flag`

This enumeration lists flags for I2C read or write addressing.

Enumerator

I2C_WRITE	write command flag
I2C_READ	read command flag

13.8.4.2 swi_flag

```
enum swi_flag
```

This enumeration lists flags for SWI.

Enumerator

SWI_FLAG_CMD	flag preceding a command
SWI_FLAG_TX	flag requesting a response
SWI_FLAG_IDLE	flag requesting to go into Idle mode
SWI_FLAG_SLEEP	flag requesting to go into Sleep mode

13.8.5 Function Documentation

13.8.5.1 atca_delay_10us()

```
void atca_delay_10us (
            uint32_t delay )
```

This function delays for a number of tens of microseconds.

Parameters

in	*delay*	number of 0.01 milliseconds to delay

13.8.5.2 atca_delay_ms()

```
void atca_delay_ms (
            uint32_t delay )
```

This function delays for a number of milliseconds.

```
You can override this function if you like to do
something else in your system while delaying.
```

Parameters

in	*delay*	number of milliseconds to delay

13.8.5.3 atca_delay_us()

```
void atca_delay_us (
            uint32_t delay )
```

Timer API implemented at the HAL level.

This function delays for a number of microseconds.

Parameters

in	*delay*	number of microseconds to delay
in	*delay*	number of 0.001 milliseconds to delay

13.8.5.4 change_i2c_speed()

```
void change_i2c_speed (
            ATCAIface iface,
            uint32_t speed )
```

method to change the bus speed of I2C

method to change the bus speed of I2C.This function is not used in Linux.

method to change the bus speec of I2C

Parameters

in	*iface*	interface on which to change bus speed
in	*speed*	baud rate (typically 100000 or 400000)
in	*iface*	interface on which to change bus speed
in	*speed*	baud rate (typically 100000 or 400000)

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.5 delay_us() [1/2]

```
void delay_us (
            UINT32 delay )
```

13.8.5.6 delay_us() [2/2]

```
void delay_us (
            uint32_t delay )
```

13.8.5.7 hal_cdc_discover_buses()

```
ATCA_STATUS hal_cdc_discover_buses (
            int cdc_buses[ ],
            int max_buses )
```

discover cdc buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge.This function is currently not implemented.

Parameters

in	*cdc_buses*	- an array of logical bus numbers
in	*max_buses*	- maximum number of buses the app wants to attempt to discover

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.8 hal_cdc_discover_devices()

```
ATCA_STATUS hal_cdc_discover_devices (
            int bus_num,
            ATCAIfaceCfg cfg[ ],
            int * found )
```

discover any CryptoAuth devices on a given logical bus number

Parameters

in	*bus_num*	- logical bus number on which to look for CryptoAuth devices
out	*cfg[ ]*	- pointer to head of an array of interface config structures which get filled in by this method
out	**found*	- number of devices found on this bus

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.9 hal_check_wake()

```
ATCA_STATUS hal_check_wake (
            const uint8_t * response,
            int response_size )
```

Utility function for hal_wake to check the reply.

Parameters

in	*response*	Wake response to be checked.
in	*response_size*	Size of the response to check.

Returns

ATCA_SUCCESS for expected wake, ATCA_STATUS_SELFTEST_ERROR if the power on self test failed, ATCA_WAKE_FAILED for other failures.

13.8.5.10 hal_create_mutex()

```
ATCA_STATUS hal_create_mutex (
            void ** ppMutex,
            char * pName )
```

Optional hal interfaces.

13.8.5.11 hal_destroy_mutex()

```
ATCA_STATUS hal_destroy_mutex (
            void * pMutex )
```

13.8.5.12 hal_i2c_discover_buses()

```
ATCA_STATUS hal_i2c_discover_buses (
            int i2c_buses[ ],
            int max_buses )
```

discover i2c buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the prior knowledge

This HAL implementation assumes you've included the ASF TWI libraries in your project, otherwise, the HAL layer will not compile because the ASF TWI drivers are a dependency.

This HAL implementation assumes you've included the Plib libraries in your project, otherwise, the HAL layer will not compile because the Plib drivers are a dependency.

discover i2c buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge.This function is not implemented.

discover i2c buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge

Parameters

in	*i2c_buses*	an array of logical bus numbers
in	*max_buses*	maximum number of buses the app wants to attempt to discover

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*i2c_buses*	- an array of logical bus numbers
in	*max_buses*	- maximum number of buses the app wants to attempt to discover

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*i2c_buses*	- an array of logical bus numbers
in	*max_buses*	- maximum number of buses the app wants to attempt to discover

Returns

ATCA_SUCCESS

Parameters

in	*i2c_buses*	- an array of logical bus numbers
in	*max_buses*	- maximum number of buses the app wants to attempt to discover

Returns

ATCA_UNIMPLEMENTED

discover i2c buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge.This function is currently not implemented.

Parameters

in	*i2c_buses*	- an array of logical bus numbers
in	*max_buses*	- maximum number of buses the app wants to attempt to discover

Returns

ATCA_UNIMPLEMENTED

discover i2c buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge

Parameters

in	*i2c_buses*	- an array of logical bus numbers
in	*max_buses*	- maximum number of buses the app wants to attempt to discover

Returns

ATCA_SUCCESS

discover i2c buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge

Parameters

in	*i2c_buses*	- an array of logical bus numbers
in	*max_buses*	- maximum number of buses the app wants to attempt to discover return ATCA_SUCCESS

13.8.5.13 hal_i2c_discover_devices()

```
ATCA_STATUS hal_i2c_discover_devices (
            int bus_num,
            ATCAIfaceCfg cfg[ ],
            int * found )
```

discover any CryptoAuth devices on a given logical bus number

discover any CryptoAuth devices on a given logical bus number.This function is currently not implemented.

Parameters

in	*bus_num*	logical bus number on which to look for CryptoAuth devices
out	*cfg*	pointer to head of an array of interface config structures which get filled in by this method
out	*found*	number of devices found on this bus

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*bus_num*	- logical bus number on which to look for CryptoAuth devices
out	*cfg[ ]*	- pointer to head of an array of interface config structures which get filled in by this method
out	*∗found*	- number of devices found on this bus

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*bus_num*	logical bus number on which to look for CryptoAuth devices
out	*cfg*	pointer to head of an array of interface config structures which get filled in by this method
out	*found*	number of devices found on this bus

Returns

ATCA_SUCCESS

Parameters

in	*bus_num*	logical bus number on which to look for CryptoAuth devices
out	*cfg*	pointer to head of an array of interface config structures which get filled in by this method
out	*found*	number of devices found on this bus

Returns

ATCA_UNIMPLEMENTED

Parameters

in	*bus_num*	- logical bus number on which to look for CryptoAuth devices
out	*cfg[ ]*	- pointer to head of an array of interface config structures which get filled in by this method
out	*∗found*	- number of devices found on this bus

Returns

ATCA_UNIMPLEMENTED

Parameters

in	*bus_num*	- logical bus number on which to look for CryptoAuth devices
out	*cfg[ ]*	- pointer to head of an array of interface config structures which get filled in by this method
out	*∗found*	- number of devices found on this bus

Returns

ATCA_SUCCESS

Parameters

in	*bus_num*	Logical bus number on which to look for CryptoAuth devices
out	*cfg*	Pointer to head of an array of interface config structures which get filled in by this method
out	*found*	Number of devices found on this bus

Returns

ATCA_SUCCESS on success, otherwise an error code.

default configuration, to be reused during discovery process

default configuration, to be reused during discovery process

default configuration, to be reused during discovery process

default configuration, to be reused during discovery process

default configuration, to be reused during discovery process

default configuration, to be reused during discovery process

default configuration, to be reused during discovery process

default configuration, to be reused during discovery process

13.8.5.14 hal_i2c_idle()

```
ATCA_STATUS hal_i2c_idle (
            ATCAIface iface )
```

idle CryptoAuth device using I2C bus

Parameters

in	*iface*	interface to logical device to idle

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*iface*	interface to logical device to idle

Returns

ATCA_SUCCESS

< Word Address Value = Idle

13.8.5.15 hal_i2c_init()

```
ATCA_STATUS hal_i2c_init (
            void * hal,
            ATCAIfaceCfg * cfg )
```

initialize an I2C interface using given config

HAL implementation of I2C init.

hal_i2c_init manages requests to initialize a physical interface. it manages use counts so when an interface has released the physical layer, it will disable the interface for some other use. You can have multiple ATCAIFace instances using the same bus, and you can have multiple ATCAIFace instances on multiple i2c buses, so hal_i2c_init manages these things and ATCAIFace is abstracted from the physical details.

hal_i2c_init manages requests to initialize a physical interface. It manages use counts so when an interface has released the physical layer, it will disable the interface for some other use. You can have multiple ATCAIFace instances using the same bus, and you can have multiple ATCAIFace instances on multiple i2c buses, so hal_i2c_init manages these things and ATCAIFace is abstracted from the physical details.

Parameters

in	*hal*	opaque ptr to HAL data
in	*cfg*	pointer to interface configuration

Returns

ATCA_SUCCESS on success, otherwise an error code.

Initialize an I2C interface using given config.

Parameters

in	*hal*	opaque pointer to HAL data
in	*cfg*	interface configuration

Returns

ATCA_SUCCESS on success, otherwise an error code.

- this HAL implementation assumes you've included the START Twi libraries in your project, otherwise, the HAL layer will not compile because the START TWI drivers are a dependency *

initialize an I2C interface using given config

Parameters

in	*hal*	- opaque ptr to HAL data
in	*cfg*	- interface configuration

Returns

ATCA_SUCCESS on success, otherwise an error code.

this implementation assumes I2C peripheral has been enabled by user. It only initialize an I2C interface using given config.

Parameters

in	*hal*	pointer to HAL specific data that is maintained by this HAL
in	*cfg*	pointer to HAL specific configuration data that is used to initialize this HAL

Returns

ATCA_SUCCESS on success, otherwise an error code.

initialize an I2C interface using given config

Parameters

in	*hal*	- opaque ptr to HAL data
in	*cfg*	- interface configuration

Returns

ATCA_SUCCESS on success, otherwise an error code.

- this HAL implementation assumes you've included the ASF I2C libraries in your project, otherwise, the HAL layer will not compile because the ASF I2C drivers are a dependency *

initialize an I2C interface using given config

Parameters

in	*hal*	- opaque ptr to HAL data
in	*cfg*	- interface configuration

Returns

ATCA_SUCCESS on success, otherwise an error code.

- this HAL implementation assumes you've included the ASF SERCOM I2C libraries in your project, otherwise, the HAL layer will not compile because the ASF I2C drivers are a dependency *

initialize an I2C interface using given config

Parameters

in	*hal*	- opaque ptr to HAL data
in	*cfg*	- interface configuration

Returns

ATCA_SUCCESS on success, otherwise an error code.

- this HAL implementation assumes you've included the ASF Twi libraries in your project, otherwise, the HAL layer will not compile because the ASF TWI drivers are a dependency *

initialize an I2C interface using given config

Parameters

in	*hal*	- opaque ptr to HAL data
in	*cfg*	- interface configuration

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.16 hal_i2c_post_init()

```
ATCA_STATUS hal_i2c_post_init (
            ATCAIface iface )
```

HAL implementation of I2C post init.

Parameters

in	*iface*	instance

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*iface*	instance

Returns

ATCA_STATUS

Parameters

in	*iface*	instance

Returns

ATCA_SUCCESS

13.8.5.17 hal_i2c_receive()

```
ATCA_STATUS hal_i2c_receive (
            ATCAIface iface,
            uint8_t * rxdata,
            uint16_t * rxlength )
```

HAL implementation of I2C receive function for ASF I2C.

HAL implementation of I2C receive function.

HAL implementation of I2C receive function for START I2C.

HAL implementation of receive bytes via I2C bit-banged.

Parameters

in	*iface*	Device to interact with.
out	*rxdata*	Data received will be returned here.
in,out	*rxlength*	As input, the size of the rxdata buffer. As output, the number of bytes received.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.18 hal_i2c_release()

```
ATCA_STATUS hal_i2c_release (
            void * hal_data )
```

manages reference count on given bus and releases resource if no more refences exist

manages reference count on given bus and releases resource if no more refernces exist

manages reference count on given bus and releases resource if no more references exist

Parameters

in	*hal_data*	- opaque pointer to hal data structure - known only to the HAL implementation

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*hal_data*	opaque pointer to hal data structure - known only to the HAL implementation

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*hal_data*	- opaque pointer to hal data structure - known only to the HAL implementation

Returns

ATCA_SUCESS

Parameters

in	*hal_data*	- opaque pointer to hal data structure - known only to the HAL implementation

Returns

ATCA_SUCCESS

Parameters

in	*hal_data*	- opaque pointer to hal data structure - known only to the HAL implementation return ATCA_SUCCESS

13.8.5.19 hal_i2c_send()

```
ATCA_STATUS hal_i2c_send (
            ATCAIface iface,
```

```
uint8_t * txdata,
int txlength )
```

HAL implementation of I2C send over ASF.

HAL implementation of I2C send.

HAL implementation of I2C send over START.

HAL implementation of Send byte(s) via I2C.

Parameters

in	*iface*	instance
in	*txdata*	pointer to space to bytes to send
in	*txlength*	number of bytes to send

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*iface*	interface of the logical device to send data to
in	*txdata*	pointer to bytes to send
in	*txlength*	number of bytes to send

Returns

ATCA_SUCCESS on success, otherwise an error code.

< Word Address Value = Command

< count Word Address byte towards txlength

Set I2C pins

Address the device and indicate that bytes are to be written

Send the remaining bytes

Send STOP regardless of i2c_status

13.8.5.20 hal_i2c_sleep()

```
ATCA_STATUS hal_i2c_sleep (
            ATCAIface iface )
```

sleep CryptoAuth device using I2C bus

Parameters

in	*iface*	interface to logical device to sleep

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*iface*	interface to logical device to sleep

Returns

ATCA_SUCESS

< Word Address Value = Sleep

13.8.5.21 hal_i2c_wake()

```
ATCA_STATUS hal_i2c_wake (
            ATCAIface iface )
```

wake up CryptoAuth device using I2C bus

Parameters

in	*iface*	interface to logical device to wakeup

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.22 hal_iface_init()

```
ATCA_STATUS hal_iface_init (
            ATCAIfaceCfg * cfg,
            ATCAHAL_t * hal )
```

Standard HAL API for ATCA to initialize a physical interface.

Parameters

in	*cfg*	pointer to ATCAIfaceCfg object
in	*hal*	pointer to ATCAHAL_t intermediate data structure

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.23 hal_iface_release()

```
ATCA_STATUS hal_iface_release (
            ATCAIfaceType iface_type,
            void * hal_data )
```

releases a physical interface, HAL knows how to interpret hal_data

Parameters

in	*iface_type*	- the type of physical interface to release
in	*hal_data*	- pointer to opaque hal data maintained by HAL implementation for this interface type

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.24 hal_kit_cdc_discover_buses()

```
ATCA_STATUS hal_kit_cdc_discover_buses (
            int cdc_buses[ ],
            int max_buses )
```

discover cdc buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge.This function is currently not implemented.

Parameters

in	*cdc_buses*	- an array of logical bus numbers
in	*max_buses*	- maximum number of buses the app wants to attempt to discover

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.25 hal_kit_cdc_discover_devices()

```
ATCA_STATUS hal_kit_cdc_discover_devices (
            int bus_num,
```

```
ATCAIfaceCfg * cfg,
int * found )
```

discover any CryptoAuth devices on a given logical bus number

Parameters

in	*bus_num*	- logical bus number on which to look for CryptoAuth devices
out	*cfg[ ]*	- pointer to head of an array of interface config structures which get filled in by this method
out	*∗found*	- number of devices found on this bus

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.26 hal_kit_cdc_idle()

```
ATCA_STATUS hal_kit_cdc_idle (
            ATCAIface iface )
```

Call the idle for kit protocol over USB CDC.

Parameters

in	*iface*	ATCAIface instance that is the interface object to send the bytes over

Returns

ATCA_SUCCESS on success, otherwise an error code.S

13.8.5.27 hal_kit_cdc_init()

```
ATCA_STATUS hal_kit_cdc_init (
            void * hal,
            ATCAIfaceCfg * cfg )
```

HAL implementation of Kit USB CDC init.

this discovery assumes a udev rule is active which renames the ATCK101 CDC device as a ttyATCAn the udev rule is:

SUBSYSTEMS=="usb", ATTRS{idVendor}=="03eb", ATTRS{idProduct}=="2122", MODE:="0777", SYMLIN↩ K+="ttyATCA%n"

Parameters

in	*hal*	pointer to HAL specific data that is maintained by this HAL
in	*cfg*	pointer to HAL specific configuration data that is used to initialize this HAL

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.28 hal_kit_cdc_post_init()

```
ATCA_STATUS hal_kit_cdc_post_init (
            ATCAIface iface )
```

HAL implementation of Kit USB CDC post init.

Parameters

in	*iface*	instance

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.29 hal_kit_cdc_receive()

```
ATCA_STATUS hal_kit_cdc_receive (
            ATCAIface iface,
            uint8_t * rxdata,
            uint16_t * rxsize )
```

HAL implementation of kit protocol receive over USB CDC.

Parameters

in	*iface*	Device to interact with.
out	*rxdata*	Data received will be returned here.
in,out	*rxsize*	As input, the size of the rxdata buffer. As output, the number of bytes received.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.30 hal_kit_cdc_release()

```
ATCA_STATUS hal_kit_cdc_release (
            void * hal_data )
```

Close the physical port for CDC over USB CDC.

Parameters

in	*hal_data*	The hardware abstraction data specific to this HAL

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.31 hal_kit_cdc_send()

```
ATCA_STATUS hal_kit_cdc_send (
            ATCAIface iface,
            uint8_t * txdata,
            int txlength )
```

HAL implementation of kit protocol send over USB CDC.

Parameters

in	*iface*	instance
in	*txdata*	pointer to bytes to send
in	*txlength*	number of bytes to send

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.32 hal_kit_cdc_sleep()

```
ATCA_STATUS hal_kit_cdc_sleep (
            ATCAIface iface )
```

Call the sleep for kit protocol over USB CDC.

Parameters

in	*iface*	ATCAIface instance that is the interface object to send the bytes over

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.33 hal_kit_cdc_wake()

```
ATCA_STATUS hal_kit_cdc_wake (
             ATCAIface iface )
```

Call the wake for kit protocol over USB CDC.

Parameters

in	*iface*	ATCAIface instance that is the interface object to send the bytes over

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.34 hal_kit_hid_discover_buses()

```
ATCA_STATUS hal_kit_hid_discover_buses (
             int cdc_buses[ ],
             int max_buses )
```

discover cdc buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge

discover all HID kits available.This function is currently not implemented. this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge

discover hid buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge.This function is currently not implemented.

Parameters

in	*cdc_buses*	- an array of logical bus numbers
in	*max_buses*	- maximum number of buses the app wants to attempt to discover
in	*cdc_buses*	- an array of logical bus numbers
in	*max_buses*	- maximum number of buses the app wants to attempt to discover

Returns

ATCA_UNIMPLEMENTED

Parameters

in	*cdc_buses*	an array of logical bus numbers
in	*max_buses*	maximum number of buses the app wants to attempt to discover

Returns

ATCA_UNIMPLEMENTED

13.8.5.35 hal_kit_hid_discover_devices()

```
ATCA_STATUS hal_kit_hid_discover_devices (
            int bus_num,
            ATCAIfaceCfg cfg[ ],
            int * found )
```

discover any CryptoAuth devices on a given logical bus number

discover any CryptoAuth devices on a given logical bus number.This function is currently not implemented.

Parameters

in	*bus_num*	- logical bus number on which to look for CryptoAuth devices
out	*cfg[ ]*	- pointer to head of an array of interface config structures which get filled in by this method
out	*∗found*	- number of devices found on this bus
in	*bus_num*	- logical bus number on which to look for CryptoAuth devices
out	*cfg[ ]*	- pointer to head of an array of interface config structures which get filled in by this method
out	*∗found*	- number of devices found on this bus

Returns

ATCA_UNIMPLEMENTED

13.8.5.36 hal_kit_hid_idle()

```
ATCA_STATUS hal_kit_hid_idle (
            ATCAIface iface )
```

Call the idle for kit protocol.

Call the idle for kit protocol over USB HID.

Parameters

in	*iface*	ATCAIface instance that is the interface object to send the bytes over

Returns

ATCA_STATUS

Parameters

in	*iface*	ATCAIface instance that is the interface object to send the bytes over

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.37 hal_kit_hid_init()

```
ATCA_STATUS hal_kit_hid_init (
            void * hal,
            ATCAIfaceCfg * cfg )
```

HAL implementation of Kit USB HID init.

Parameters

in	*hal*	pointer to HAL specific data that is maintained by this HAL
in	*cfg*	pointer to HAL specific configuration data that is used to initialize this HAL

Returns

ATCA_STATUS

Parameters

in	*hal*	pointer to HAL specific data that is maintained by this HAL
in	*cfg*	pointer to HAL specific configuration data that is used to initialize this HAL

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.38 hal_kit_hid_post_init()

```
ATCA_STATUS hal_kit_hid_post_init (
            ATCAIface iface )
```

HAL implementation of Kit HID post init.

Parameters

in	*iface*	instance

Returns

ATCA_STATUS

Parameters

in	*iface*	instance

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.39 hal_kit_hid_receive()

```
ATCA_STATUS hal_kit_hid_receive (
            ATCAIface iface,
            uint8_t * rxdata,
            uint16_t * rxsize )
```

HAL implementation of send over USB HID.

Parameters

in	*iface*	instance
in	*rxdata*	pointer to space to receive the data
in,out	*rxsize*	ptr to expected number of receive bytes to request

Returns

ATCA_STATUS

Parameters

in	*iface*	instance
in	*rxdata*	pointer to space to receive the data
in,out	*rxsize*	ptr to expected number of receive bytes to request

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.40 hal_kit_hid_release()

```
ATCA_STATUS hal_kit_hid_release (
            void * hal_data )
```

Close the physical port for HID.

Parameters

in	*hal_data*	The hardware abstraction data specific to this HAL

Returns

ATCA_STATUS

Parameters

in	*hal_data*	The hardware abstraction data specific to this HAL

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.41 hal_kit_hid_send()

```
ATCA_STATUS hal_kit_hid_send (
            ATCAIface iface,
            uint8_t * txdata,
            int txlength )
```

HAL implementation of kit protocol send over USB HID.

Parameters

in	*iface*	instance
in	*txdata*	pointer to bytes to send
in	*txlength*	number of bytes to send

Returns

ATCA_STATUS

Parameters

in	*iface*	instance
in	*txdata*	pointer to bytes to send
in	*txlength*	number of bytes to send

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.42 hal_kit_hid_sleep()

```
ATCA_STATUS hal_kit_hid_sleep (
            ATCAIface iface )
```

Call the sleep for kit protocol.

Call the sleep for kit protocol over USB HID.

Parameters

in	*iface*	ATCAIface instance that is the interface object to send the bytes over

Returns

ATCA_STATUS

Parameters

in	*iface*	ATCAIface instance that is the interface object to send the bytes over

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.43 hal_kit_hid_wake()

```
ATCA_STATUS hal_kit_hid_wake (
            ATCAIface iface )
```

Call the wake for kit protocol.

Call the wake for kit protocol over USB HID.

Parameters

in	*iface*	ATCAIface instance that is the interface object to send the bytes over

Returns

ATCA_STATUS

Parameters

in	*iface*	ATCAIface instance that is the interface object to send the bytes over

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.44 hal_kit_phy_num_found()

```
ATCA_STATUS hal_kit_phy_num_found (
              int8_t * num_found )
```

Number of USB CDC devices found.

Parameters

out	*num_found*	

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.45 hal_lock_mutex()

```
ATCA_STATUS hal_lock_mutex (
              void * pMutex )
```

13.8.5.46 hal_swi_discover_buses()

```
ATCA_STATUS hal_swi_discover_buses (
              int swi_buses[ ],
              int max_buses )
```

discover swi buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application.This function is currently not supported. of the a-priori knowledge

discover swi buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge

Parameters

in	*swi_buses*	- an array of logical bus numbers
in	*max_buses*	- maximum number of buses the app wants to attempt to discover

Returns

ATCA_UNIMPLEMENTED

Parameters

in	*swi_buses*	- an array of logical bus numbers
in	*max_buses*	- maximum number of buses the app wants to attempt to discover

Returns

ATCA_SUCCESS

13.8.5.47 hal_swi_discover_devices()

```
ATCA_STATUS hal_swi_discover_devices (
            int bus_num,
            ATCAIfaceCfg cfg[ ],
            int * found )
```

discover any CryptoAuth devices on a given logical bus number.This function is curently not supported.

discover any CryptoAuth devices on a given logical bus number

Parameters

in	*bus_num*	- logical bus number on which to look for CryptoAuth devices
out	*cfg[ ]*	- pointer to head of an array of interface config structures which get filled in by this method
out	*∗found*	- number of devices found on this bus

Returns

ATCA_UNIMPLEMENTED

Parameters

in	*bus_num*	- logical bus number on which to look for CryptoAuth devices
out	*cfg[ ]*	- pointer to head of an array of interface config structures which get filled in by this method
out	*∗found*	- number of devices found on this bus

Returns

ATCA_SUCCESS

default configuration, to be reused during discovery process

13.8.5.48 hal_swi_idle()

```
ATCA_STATUS hal_swi_idle (
            ATCAIface iface )
```

Send Idle flag via SWI.

idle CryptoAuth device using SWI interface

Parameters

in	*iface*	interface of the logical device to idle

Returns

ATCA_SUCCES

Parameters

in	*iface*	interface to logical device to idle

Returns

ATCA_SUCCESS on success, otherwise an error code.

Set SWI pin

13.8.5.49 hal_swi_init()

```
ATCA_STATUS hal_swi_init (
            void * hal,
            ATCAIfaceCfg * cfg )
```

hal_swi_init manages requests to initialize a physical interface. It manages use counts so when an interface has released the physical layer, it will disable the interface for some other use. You can have multiple ATCAIFace instances using the same bus, and you can have multiple ATCAIFace instances on multiple swi buses, so hal_swi_init manages these things and ATCAIFace is abstracted from the physical details.

hal_swi_init manages requests to initialize a physical interface. it manages use counts so when an interface has released the physical layer, it will disable the interface for some other use. You can have multiple ATCAIFace instances using the same bus, and you can have multiple ATCAIFace instances on multiple swi buses, so hal_swi_init manages these things and ATCAIFace is abstracted from the physical details.

Initialize an SWI interface using given config.

Parameters

in	*hal*	opaque pointer to HAL data
in	*cfg*	interface configuration

Returns

ATCA_SUCCESS on success, otherwise an error code.

initialize an SWI interface using given config

Parameters

in	*hal*	- opaque ptr to HAL data
in	*cfg*	- interface configuration

Returns

ATCA_SUCCESS on success, otherwise an error code.

assign GPIO pin

13.8.5.50 hal_swi_post_init()

```
ATCA_STATUS hal_swi_post_init (
            ATCAIface iface )
```

HAL implementation of SWI post init.

Parameters

in	*iface*	ATCAIface instance

Returns

ATCA_SUCCESS

Parameters

in	*iface*	instance

Returns

ATCA_SUCCESS

13.8.5.51 hal_swi_receive()

```
ATCA_STATUS hal_swi_receive (
            ATCAIface iface,
            uint8_t * rxdata,
            uint16_t * rxlength )
```

Receive byte(s) via SWI.

HAL implementation of SWI receive function over UART.

Parameters

in	*iface*	Device to interact with.
out	*rxdata*	Data received will be returned here.
in,out	*rxlength*	As input, the size of the rxdata buffer. As output, the number of bytes received.

Returns

ATCA_SUCCESS on success, otherwise an error code.

Set SWI pin

13.8.5.52 hal_swi_release()

```
ATCA_STATUS hal_swi_release (
            void * hal_data )
```

Manages reference count on given bus and releases resource if no more reference(s) exist.

manages reference count on given bus and releases resource if no more refences exist

Parameters

in	*hal_data*	opaque pointer to hal data structure - known only to the HAL implementation

Returns

ATCA_SUCCESS

Parameters

in	*hal_data*	- opaque pointer to hal data structure - known only to the HAL implementation

Returns

ATCA_SUCCESS on success, otherwise an error code.

if the use count for this bus has gone to 0 references, disable it. protect against an unbracketed release

13.8.5.53 hal_swi_send()

```
ATCA_STATUS hal_swi_send (
            ATCAIface iface,
            uint8_t * txdata,
            int txlength )
```

Send byte(s) via SWI.

HAL implementation of SWI send command over UART.

Parameters

in	*iface*	interface of the logical device to send data to
in	*txdata*	pointer to bytes to send
in	*txlength*	number of bytes to send

Returns

ATCA_SUCCESS

Parameters

in	*iface*	instance
in	*txdata*	pointer to space to bytes to send
in	*txlength*	number of bytes to send

Returns

ATCA_SUCCESS on success, otherwise an error code.

Skip the Word Address data as SWI doesn't use it

Set SWI pin

Send Command Flag

Send the remaining bytes

13.8.5.54 hal_swi_send_flag()

```
ATCA_STATUS hal_swi_send_flag (
            ATCAIface iface,
            uint8_t data )
```

HAL implementation of SWI send one byte over UART.

Parameters

in	*iface*	instance
in	*data*	bytes to send

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.55 hal_swi_sleep()

```
ATCA_STATUS hal_swi_sleep (
            ATCAIface iface )
```

Send Sleep flag via SWI.

sleep CryptoAuth device using SWI interface

Parameters

in	*iface*	interface of the logical device to sleep

Returns

ATCA_SUCCESS

Parameters

in	*iface*	interface to logical device to sleep

Returns

ATCA_SUCCESS on success, otherwise an error code.

Set SWI pin

13.8.5.56 hal_swi_wake()

```
ATCA_STATUS hal_swi_wake (
            ATCAIface iface )
```

Send Wake flag via SWI.

wake up CryptoAuth device using SWI interface

Parameters

in	*iface*	interface of the logical device to wake up

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*iface*	interface to logical device to wakeup

Returns

ATCA_SUCCESS on success, otherwise an error code.

Set SWI pin

Generate Wake Token

Wait tWHI + tWLO

13.8.5.57 hal_unlock_mutex()

```
ATCA_STATUS hal_unlock_mutex (
            void * pMutex )
```

13.8.5.58 i2c_read()

```
ATCA_STATUS i2c_read (
            I2C_MODULE i2c_id,
            uint8_t address,
            uint8_t * data,
            uint16_t len )
```

13.8.5.59 i2c_write()

```
void i2c_write (
            I2C_MODULE i2c_id,
            uint8_t address,
            uint8_t * data,
            int len )
```

13.8.5.60 kit_id_from_devtype()

```
const char* kit_id_from_devtype (
            ATCADeviceType devtype )
```

Kit Protocol is key

13.8.5.61 kit_idle()

```
ATCA_STATUS kit_idle (
            ATCAIface iface )
```

Call the idle for kit protocol.

Parameters

in	*iface*	the interface object to send the bytes over

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.62 kit_init()

```
ATCA_STATUS kit_init (
            ATCAIface iface )
```

HAL implementation of kit protocol init. This function calls back to the physical protocol to send the bytes.

Parameters

in	*iface*	instance

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.63 kit_interface_from_kittype()

```
const char* kit_interface_from_kittype (
            ATCAKitType kittype )
```

Kit interface from device

13.8.5.64 kit_parse_rsp()

```
ATCA_STATUS kit_parse_rsp (
            const char * pkitbuf,
            int nkitbuf,
            uint8_t * kitstatus,
            uint8_t * rxdata,
            int * datasize )
```

Parse the response ascii from the kit.

Parameters

out	*pkitbuf*	pointer to ascii kit protocol data to parse
in	*nkitbuf*	length of the ascii kit protocol data
in	*kitstatus*	status of the ascii device
in	*rxdata*	pointer to the binary data buffer
in	*datasize*	size of the pointer to the binary data buffer

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.65 kit_phy_num_found()

```
ATCA_STATUS kit_phy_num_found (
            int8_t * num_found )
```

Number of USB HID devices found.

Parameters

out	*num_found*	

Returns

ATCA_STATUS

Parameters

out	*num_found*	

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

out	*num_found*	

Returns

SUCCESS

13.8.5.66 kit_phy_receive() [1/2]

```
ATCA_STATUS kit_phy_receive (
            ATCAIface iface,
            char * rxdata,
            int * rxsize )
```

HAL implementation of kit protocol receive data.It is called by the top layer.

Parameters

in	*iface*	instance
out	*rxdata*	pointer to space to receive the data
in,out	*rxsize*	ptr to expected number of receive bytes to request

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.67 kit_phy_receive() [2/2]

```
ATCA_STATUS kit_phy_receive (
            ATCAIface iface,
            uint8_t * rxdata,
            int * rxsize )
```

HAL implementation of kit protocol send over USB HID.

HAL implementation of kit protocol receive.This function is called by the top layer.

Parameters

in	*iface*	instance
out	*rxdata*	pointer to space to receive the data
in,out	*rxsize*	ptr to expected number of receive bytes to request

Returns

ATCA_STATUS

Parameters

in	*iface*	instance
out	*rxdata*	pointer to space to receive the data
in,out	*rxsize*	ptr to expected number of receive bytes to request

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.68 kit_phy_send() [1/2]

```
ATCA_STATUS kit_phy_send (
            ATCAIface iface,
```

```
const char * txdata,
int txlength )
```

HAL implementation of kit protocol send .It is called by the top layer.

Parameters

in	*iface*	instance
in	*txdata*	pointer to bytes to send
in	*txlength*	number of bytes to send

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.69 kit_phy_send() [2/2]

```
ATCA_STATUS kit_phy_send (
            ATCAIface iface,
            uint8_t * txdata,
            int txlength )
```

HAL implementation of send over USB HID.

HAL implementation of send over Kit protocol.This function is called by the top layer.

Parameters

in	*iface*	instance
in	*txdata*	pointer to bytes to send
in	*txlength*	number of bytes to send

Returns

ATCA_STATUS

Parameters

in	*iface*	instance
in	*txdata*	pointer to bytes to send
in	*txlength*	number of bytes to send

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.70 kit_receive()

```
ATCA_STATUS kit_receive (
            ATCAIface iface,
            uint8_t * rxdata,
            uint16_t * rxsize )
```

HAL implementation to receive bytes and unwrap from kit protocol. This function calls back to the physical protocol to receive the bytes.

Parameters

in	*iface*	instance
in	*rxdata*	pointer to space to receive the data
in,out	*rxsize*	ptr to expected number of receive bytes to request

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.71 kit_send()

```
ATCA_STATUS kit_send (
            ATCAIface iface,
            const uint8_t * txdata,
            int txlength )
```

HAL implementation of kit protocol send. This function calls back to the physical protocol to send the bytes.

Parameters

in	*iface*	instance
in	*txdata*	pointer to bytes to send
in	*txlength*	number of bytes to send

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.72 kit_sleep()

```
ATCA_STATUS kit_sleep (
            ATCAIface iface )
```

Call the sleep for kit protocol.

Parameters

in	*iface*	the interface object to send the bytes over

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.73 kit_wake()

```
ATCA_STATUS kit_wake (
            ATCAIface iface )
```

Call the wake for kit protocol.

Parameters

in	*iface*	the interface object to send the bytes over

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.74 kit_wrap_cmd()

```
ATCA_STATUS kit_wrap_cmd (
            const uint8_t * txdata,
            int txlen,
            char * pkitcmd,
            int * nkitcmd,
            char target )
```

Wrap binary bytes in ascii kit protocol.

Parameters

in	*txdata*	Binary data to wrap.
in	*txlen*	Length of binary data in bytes.
out	*pkitcmd*	ASCII kit protocol wrapped data is return here.
in,out	*nkitcmd*	As input, the size of the pkitcmd buffer. As output, the number of bytes returned in the pkitcmd buffer.
in	*target*	Target char to use 's' for SHA devices, 'e' for ECC devices.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.75 strnchr()

```
char* strnchr (
            const char * s,
            size_t count,
            int c )
```

13.8.5.76 swi_uart_deinit()

```
ATCA_STATUS swi_uart_deinit (
            ATCASWIMaster_t * instance )
```

Implementation of SWI UART deinit.

HAL implementation of SWI UART deinit.

Parameters

in	*instance*	instance

Returns

ATCA_SUCCESS

Parameters

in	*instance*	instance

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.77 swi_uart_discover_buses()

```
void swi_uart_discover_buses (
            int swi_uart_buses[ ],
            int max_buses )
```

discover UART buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge

Parameters

in	*swi_uart_buses*	- an array of logical bus numbers
in	*max_buses*	- maximum number of buses the app wants to attempt to discover

13.8.5.78 swi_uart_init()

```
ATCA_STATUS swi_uart_init (
            ATCASWIMaster_t * instance )
```

Implementation of SWI UART init.

HAL implementation of SWI UART init.

- this HAL implementation assumes you've included the ASF SERCOM UART libraries in your project, otherwise, the HAL layer will not compile because the ASF UART drivers are a dependency *

Parameters

in	*instance*	instance

Returns

ATCA_SUCCESS on success, otherwise an error code.

- this HAL implementation assumes you've included the START SERCOM UART libraries in your project, otherwise, the HAL layer will not compile because the START UART drivers are a dependency *

Parameters

in	*instance*	instance

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.79 swi_uart_mode()

```
void swi_uart_mode (
            ATCASWIMaster_t * instance,
            uint8_t mode )
```

implementation of SWI UART change mode.

HAL implementation of SWI UART change mode.

Parameters

in	*instance*	instance
in	*mode*	(TRANSMIT_MODE or RECEIVE_MODE)

13.8.5.80 swi_uart_receive_byte()

```
ATCA_STATUS swi_uart_receive_byte (
            ATCASWIMaster_t * instance,
            uint8_t * data )
```

HAL implementation of SWI UART receive bytes over ASF. This function receive one byte over UART.

Parameters

in	*instance*	instance
in,out	*data*	pointer to space to receive the data

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*instance*	instance
out	*data*	pointer to space to receive the data

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.81 swi_uart_send_byte()

```
ATCA_STATUS swi_uart_send_byte (
            ATCASWIMaster_t * instance,
            uint8_t data )
```

HAL implementation of SWI UART send byte over ASF. This function send one byte over UART.

Parameters

in	*instance*	instance
in	*data*	byte to send

Returns

ATCA_SUCCESS on success, otherwise an error code.

Parameters

in	*instance*	instance
in	*data*	number of byte to send

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.8.5.82 swi_uart_setbaud()

```
void swi_uart_setbaud (
            ATCASWIMaster_t * instance,
            uint32_t baudrate )
```

implementation of SWI UART change baudrate.

HAL implementation of SWI UART change baudrate.

Parameters

in	*instance*	instance
in	*baudrate*	(typically 230400 , 160000 or 115200)
in	*instance*	instance
in	*baudrate*	(typically 230400 or 115200)

13.8.6 Variable Documentation

13.8.6.1 _gCdc

```
atcacdc_t _gCdc
```

13.8.6.2 _gHid [1/3]

```
atcahid_t _gHid
```

13.8.6.3 _gHid [2/3]

```
atcahid_t _gHid
```

13.8.6.4 _gHid [3/3]

```
atcahid_t _gHid
```

13.8.6.5 dev

```
char* dev = "/dev/ttyACM0"
```

13.8.6.6 pin_conf

```
struct port_config pin_conf
```

13.8.6.7 speed

```
int speed = B115200
```

13.9 Host side crypto methods (atcah_)

Use these functions if your system does not use an ATCADevice as a host but implements the host in firmware. The functions provide host-side cryptographic functionality for an ATECC client device. They are intended to accompany the CryptoAuthLib functions. They can be called directly from an application, or integrated into an API.

Data Structures

- struct atca_temp_key

  Structure to hold TempKey fields.
- struct atca_include_data_in_out

  Input / output parameters for function atca_include_data().
- struct atca_nonce_in_out

  Input/output parameters for function atca_nonce().
- struct atca_io_decrypt_in_out
- struct atca_verify_mac
- struct atca_secureboot_enc_in_out
- struct atca_secureboot_mac_in_out
- struct atca_mac_in_out

  Input/output parameters for function atca_mac().
- struct atca_hmac_in_out

  Input/output parameters for function atca_hmac().
- struct atca_gen_dig_in_out

  Input/output parameters for function atcah_gen_dig().
- struct atca_write_mac_in_out

  Input/output parameters for function atcah_write_auth_mac() and atcah_privwrite_auth_mac().
- struct atca_derive_key_in_out

  Input/output parameters for function atcah_derive_key().
- struct atca_derive_key_mac_in_out

  Input/output parameters for function atcah_derive_key_mac().
- struct atca_decrypt_in_out

  Input/output parameters for function atca_decrypt().
- struct atca_check_mac_in_out

  Input/output parameters for function atcah_check_mac().
- struct atca_verify_in_out

  Input/output parameters for function atcah_verify().
- struct atca_gen_key_in_out

  Input/output parameters for calculating the PubKey digest put into TempKey by the GenKey command with the atcah_gen_key_msg() function.
- struct atca_sign_internal_in_out

  Input/output parameters for calculating the message and digest used by the Sign(internal) command. Used with the atcah_sign_internal_msg() function.

Typedefs

- typedef struct atca_temp_key atca_temp_key_t

  Structure to hold TempKey fields.
- typedef struct atca_nonce_in_out atca_nonce_in_out_t
- typedef struct atca_io_decrypt_in_out atca_io_decrypt_in_out_t
- typedef struct atca_verify_mac atca_verify_mac_in_out_t
- typedef struct atca_secureboot_enc_in_out atca_secureboot_enc_in_out_t
- typedef struct atca_secureboot_mac_in_out atca_secureboot_mac_in_out_t
- typedef struct atca_mac_in_out atca_mac_in_out_t
- typedef struct atca_gen_dig_in_out atca_gen_dig_in_out_t

  Input/output parameters for function atcah_gen_dig().
- typedef struct atca_write_mac_in_out atca_write_mac_in_out_t

  Input/output parameters for function atcah_write_auth_mac() and atcah_privwrite_auth_mac().
- typedef struct atca_check_mac_in_out atca_check_mac_in_out_t

  Input/output parameters for function atcah_check_mac().
- typedef struct atca_verify_in_out atca_verify_in_out_t
- typedef struct atca_gen_key_in_out atca_gen_key_in_out_t

  Input/output parameters for calculating the PubKey digest put into TempKey by the GenKey command with the atcah_gen_key_msg() function.
- typedef struct atca_sign_internal_in_out atca_sign_internal_in_out_t

  Input/output parameters for calculating the message and digest used by the Sign(internal) command. Used with the atcah_sign_internal_msg() function.

Functions

- ATCA_STATUS atcah_nonce (struct atca_nonce_in_out ∗param)

  This function calculates host side nonce with the parameters passed.
- ATCA_STATUS atcah_mac (struct atca_mac_in_out ∗param)

  This function generates an SHA-256 digest (MAC) of a key, challenge, and other information.
- ATCA_STATUS atcah_check_mac (struct atca_check_mac_in_out ∗param)

  This function performs the checkmac operation to generate client response on the host side .
- ATCA_STATUS atcah_hmac (struct atca_hmac_in_out ∗param)

  This function generates an HMAC / SHA-256 hash of a key and other information.
- ATCA_STATUS atcah_gen_dig (struct atca_gen_dig_in_out ∗param)

  This function combines the current TempKey with a stored value.
- ATCA_STATUS atcah_gen_mac (struct atca_gen_dig_in_out ∗param)

  This function generates mac with session key with a plain text.
- ATCA_STATUS atcah_write_auth_mac (struct atca_write_mac_in_out ∗param)

  This function calculates the input MAC for the Write command.
- ATCA_STATUS atcah_privwrite_auth_mac (struct atca_write_mac_in_out ∗param)

  This function calculates the input MAC for the PrivWrite command.
- ATCA_STATUS atcah_derive_key (struct atca_derive_key_in_out ∗param)

  This function derives a key with a key and TempKey.
- ATCA_STATUS atcah_derive_key_mac (struct atca_derive_key_mac_in_out ∗param)

  This function calculates the input MAC for a DeriveKey command.
- ATCA_STATUS atcah_decrypt (struct atca_decrypt_in_out ∗param)

  This function decrypts 32-byte encrypted data received with the Read command.
- ATCA_STATUS atcah_sha256 (int32_t len, const uint8_t ∗message, uint8_t ∗digest)

  This function creates a SHA256 digest on a little-endian system.
- uint8_t ∗ atcah_include_data (struct atca_include_data_in_out ∗param)

This function copies otp and sn data into a command buffer.

- ATCA_STATUS atcah_gen_key_msg (struct atca_gen_key_in_out ∗param)

  Calculate the PubKey digest created by GenKey and saved to TempKey.
- ATCA_STATUS atcah_config_to_sign_internal (ATCADeviceType device_type, struct atca_sign_internal_in_out ∗param, const uint8_t ∗config)

  Populate the slot_config, key_config, and is_slot_locked fields in the atca_sign_internal_in_out structure from the provided config zone.
- ATCA_STATUS atcah_sign_internal_msg (ATCADeviceType device_type, struct atca_sign_internal_in_out ∗param)

  Builds the full message that would be signed by the Sign(Internal) command.
- ATCA_STATUS atcah_verify_mac (atca_verify_mac_in_out_t ∗param)

  Calculate the expected MAC on the host side for the Verify command.
- ATCA_STATUS atcah_secureboot_enc (atca_secureboot_enc_in_out_t ∗param)

  Encrypts the digest for the SecureBoot command when using the encrypted digest / validating mac option.
- ATCA_STATUS atcah_secureboot_mac (atca_secureboot_mac_in_out_t ∗param)

  Calculates the expected MAC returned from the SecureBoot command when verification is a success.
- ATCA_STATUS atcah_encode_counter_match (uint32_t counter, uint8_t ∗counter_match)

  Builds the counter match value that needs to be stored in a slot.
- ATCA_STATUS atcah_io_decrypt (struct atca_io_decrypt_in_out ∗param)

  Decrypt data that's been encrypted by the IO protection key. The ECDH and KDF commands on the ATECC608A are the only ones that support this operation.

Variables

- uint8_t ∗ p_temp

  [out] pointer to output buffer
- const uint8_t ∗ otp

  [in] pointer to one-time-programming data
- const uint8_t ∗ sn

  [in] pointer to serial number data
- uint8_t mode

  [in] Mode parameter used in Nonce command (Param1).
- uint16_t zero

  [in] Zero parameter used in Nonce command (Param2).
- const uint8_t ∗ num_in

  [in] Pointer to 20-byte NumIn data used in Nonce command.
- const uint8_t ∗ rand_out

  [in] Pointer to 32-byte RandOut data from Nonce command.
- struct atca_temp_key ∗ temp_key

  [in,out] Pointer to TempKey structure.
- uint8_t mode

  [in] Mode parameter used in MAC command (Param1).
- uint16_t key_id

  [in] KeyID parameter used in MAC command (Param2).
- const uint8_t ∗ challenge

  [in] Pointer to 32-byte Challenge data used in MAC command, depending on mode.
- const uint8_t ∗ key

  [in] Pointer to 32-byte key used to generate MAC digest.
- const uint8_t ∗ otp

  [in] Pointer to 11-byte OTP, optionally included in MAC digest, depending on mode.
- const uint8_t ∗ sn

[in] Pointer to 9-byte SN, optionally included in MAC digest, depending on mode.

- uint8_t ∗ response

  [out] Pointer to 32-byte SHA-256 digest (MAC).

- struct atca_temp_key ∗ temp_key

  [in,out] Pointer to TempKey structure.

- uint8_t mode

  [in] Mode parameter used in HMAC command (Param1).

- uint16_t key_id

  [in] KeyID parameter used in HMAC command (Param2).

- const uint8_t ∗ key

  [in] Pointer to 32-byte key used to generate HMAC digest.

- const uint8_t ∗ otp

  [in] Pointer to 11-byte OTP, optionally included in HMAC digest, depending on mode.

- const uint8_t ∗ sn

  [in] Pointer to 9-byte SN, optionally included in HMAC digest, depending on mode.

- uint8_t ∗ response

  [out] Pointer to 32-byte SHA-256 HMAC digest.

- struct atca_temp_key ∗ temp_key

  [in,out] Pointer to TempKey structure.

- uint8_t ∗ crypto_data

  [in,out] Pointer to 32-byte data. Input encrypted data from Read command (Contents field), output decrypted.

- struct atca_temp_key ∗ temp_key

  [in,out] Pointer to TempKey structure.

- uint16_t curve_type

  [in] Curve type used in Verify command (Param2).

- const uint8_t ∗ signature

  [in] Pointer to ECDSA signature to be verified

- const uint8_t ∗ public_key

  [in] Pointer to the public key to be used for verification

- struct atca_temp_key ∗ temp_key

  [in,out] Pointer to TempKey structure.

Definitions for ATECC Message Sizes to Calculate a SHA256 Hash

"||" is the concatenation operator. The number in braces is the length of the hash input value in bytes.

- #define ATCA_MSG_SIZE_NONCE (55)

  RandOut{32} || NumIn{20} || OpCode{1} || Mode{1} || LSB of Param2{1}.

- #define ATCA_MSG_SIZE_MAC (88)

  (Key or TempKey){32} || (Challenge or TempKey){32} || OpCode{1} || Mode{1} || Param2{2} || (OTP0_7 or 0){8} || (OTP8_10 or 0){3} || SN8{1} || (SN4_7 or 0){4} || SN0_1{2} || (SN2_3 or 0){2}

- #define ATCA_MSG_SIZE_HMAC (88)
- #define ATCA_MSG_SIZE_GEN_DIG (96)

  KeyId{32} || OpCode{1} || Param1{1} || Param2{2} || SN8{1} || SN0_1{2} || 0{25} || TempKey{32}.

- #define ATCA_MSG_SIZE_DERIVE_KEY (96)

  KeyId{32} || OpCode{1} || Param1{1} || Param2{2} || SN8{1} || SN0_1{2} || 0{25} || TempKey{32}.

- #define ATCA_MSG_SIZE_DERIVE_KEY_MAC (39)

  KeyId{32} || OpCode{1} || Param1{1} || Param2{2} || SN8{1} || SN0_1{2}.

- #define ATCA_MSG_SIZE_ENCRYPT_MAC (96)

KeyId{32} || OpCode{1} || Param1{1} || Param2{2}|| SN8{1} || SN0_1{2} || 0{25} || TempKey{32}.

- #define ATCA_MSG_SIZE_PRIVWRITE_MAC (96)

  KeyId{32} || OpCode{1} || Param1{1} || Param2{2}|| SN8{1} || SN0_1{2} || 0{21} || PlainText{36}.
- #define ATCA_COMMAND_HEADER_SIZE (4)
- #define ATCA_GENDIG_ZEROS_SIZE (25)
- #define ATCA_WRITE_MAC_ZEROS_SIZE (25)
- #define ATCA_PRIVWRITE_MAC_ZEROS_SIZE (21)
- #define ATCA_PRIVWRITE_PLAIN_TEXT_SIZE (36)
- #define ATCA_DERIVE_KEY_ZEROS_SIZE (25)
- #define HMAC_BLOCK_SIZE (64)
- #define ENCRYPTION_KEY_SIZE (64)

Default Fixed Byte Values of Serial Number (SN[0:1] and SN[8])

- #define ATCA_SN_0_DEF (0x01)
- #define ATCA_SN_1_DEF (0x23)
- #define ATCA_SN_8_DEF (0xEE)

Definition for TempKey Mode

- #define MAC_MODE_USE_TEMPKEY_MASK ((uint8_t)0x03)

  mode mask for MAC command when using TempKey

13.9.1 Detailed Description

Use these functions if your system does not use an ATCADevice as a host but implements the host in firmware. The functions provide host-side cryptographic functionality for an ATECC client device. They are intended to accompany the CryptoAuthLib functions. They can be called directly from an application, or integrated into an API.

Modern compilers can garbage-collect unused functions. If your compiler does not support this feature, you can just discard this module from your project if you do use an ATECC as a host. Or, if you don't, delete the functions you do not use.

13.9.2 Macro Definition Documentation

13.9.2.1 ATCA_COMMAND_HEADER_SIZE

```
#define ATCA_COMMAND_HEADER_SIZE ( 4)
```

13.9.2.2 ATCA_DERIVE_KEY_ZEROS_SIZE

```
#define ATCA_DERIVE_KEY_ZEROS_SIZE (25)
```

13.9.2.3 ATCA_GENDIG_ZEROS_SIZE

```
#define ATCA_GENDIG_ZEROS_SIZE (25)
```

13.9.2.4 ATCA_MSG_SIZE_DERIVE_KEY

```
#define ATCA_MSG_SIZE_DERIVE_KEY (96)
```

KeyId{32} || OpCode{1} || Param1{1} || Param2{2} || SN8{1} || SN0_1{2} || 0{25} || TempKey{32}.

13.9.2.5 ATCA_MSG_SIZE_DERIVE_KEY_MAC

```
#define ATCA_MSG_SIZE_DERIVE_KEY_MAC (39)
```

KeyId{32} || OpCode{1} || Param1{1} || Param2{2} || SN8{1} || SN0_1{2}.

13.9.2.6 ATCA_MSG_SIZE_ENCRYPT_MAC

```
#define ATCA_MSG_SIZE_ENCRYPT_MAC (96)
```

KeyId{32} || OpCode{1} || Param1{1} || Param2{2}|| SN8{1} || SN0_1{2} || 0{25} || TempKey{32}.

13.9.2.7 ATCA_MSG_SIZE_GEN_DIG

```
#define ATCA_MSG_SIZE_GEN_DIG (96)
```

KeyId{32} || OpCode{1} || Param1{1} || Param2{2} || SN8{1} || SN0_1{2} || 0{25} || TempKey{32}.

13.9.2.8 ATCA_MSG_SIZE_HMAC

```
#define ATCA_MSG_SIZE_HMAC (88)
```

13.9.2.9 ATCA_MSG_SIZE_MAC

```
#define ATCA_MSG_SIZE_MAC (88)
```

(Key or TempKey){32} || (Challenge or TempKey){32} || OpCode{1} || Mode{1} || Param2{2} || (OTP0_7 or 0){8} || (OTP8_10 or 0){3} || SN8{1} || (SN4_7 or 0){4} || SN0_1{2} || (SN2_3 or 0){2}

13.9.2.10 ATCA_MSG_SIZE_NONCE

```
#define ATCA_MSG_SIZE_NONCE (55)
```

RandOut{32} || NumIn{20} || OpCode{1} || Mode{1} || LSB of Param2{1}.

13.9.2.11 ATCA_MSG_SIZE_PRIVWRITE_MAC

```
#define ATCA_MSG_SIZE_PRIVWRITE_MAC (96)
```

KeyId{32} || OpCode{1} || Param1{1} || Param2{2}|| SN8{1} || SN0_1{2} || 0{21} || PlainText{36}.

13.9.2.12 ATCA_PRIVWRITE_MAC_ZEROS_SIZE

```
#define ATCA_PRIVWRITE_MAC_ZEROS_SIZE (21)
```

13.9.2.13 ATCA_PRIVWRITE_PLAIN_TEXT_SIZE

```
#define ATCA_PRIVWRITE_PLAIN_TEXT_SIZE (36)
```

13.9.2.14 ATCA_SN_0_DEF

```
#define ATCA_SN_0_DEF (0x01)
```

13.9.2.15 ATCA_SN_1_DEF

```
#define ATCA_SN_1_DEF (0x23)
```

13.9.2.16 ATCA_SN_8_DEF

```
#define ATCA_SN_8_DEF (0xEE)
```

13.9.2.17 ATCA_WRITE_MAC_ZEROS_SIZE

```
#define ATCA_WRITE_MAC_ZEROS_SIZE (25)
```

13.9.2.18 ENCRYPTION_KEY_SIZE

```
#define ENCRYPTION_KEY_SIZE (64)
```

13.9.2.19 HMAC_BLOCK_SIZE

```
#define HMAC_BLOCK_SIZE (64)
```

13.9.2.20 MAC_MODE_USE_TEMPKEY_MASK

```
#define MAC_MODE_USE_TEMPKEY_MASK ((uint8_t)0x03)
```

mode mask for MAC command when using TempKey

13.9.3 Typedef Documentation

13.9.3.1 atca_check_mac_in_out_t

```
typedef struct atca_check_mac_in_out atca_check_mac_in_out_t
```

Input/output parameters for function atcah_check_mac().

13.9.3.2 atca_gen_dig_in_out_t

```
typedef struct atca_gen_dig_in_out atca_gen_dig_in_out_t
```

Input/output parameters for function atcah_gen_dig().

13.9.3.3 atca_gen_key_in_out_t

```
typedef struct atca_gen_key_in_out atca_gen_key_in_out_t
```

Input/output parameters for calculating the PubKey digest put into TempKey by the GenKey command with the atcah_gen_key_msg() function.

13.9.3.4 atca_io_decrypt_in_out_t

```
typedef struct atca_io_decrypt_in_out atca_io_decrypt_in_out_t
```

13.9.3.5 atca_mac_in_out_t

```
typedef struct atca_mac_in_out atca_mac_in_out_t
```

13.9.3.6 atca_nonce_in_out_t

```
typedef struct atca_nonce_in_out atca_nonce_in_out_t
```

13.9.3.7 atca_secureboot_enc_in_out_t

```
typedef struct atca_secureboot_enc_in_out atca_secureboot_enc_in_out_t
```

13.9.3.8 atca_secureboot_mac_in_out_t

```
typedef struct atca_secureboot_mac_in_out atca_secureboot_mac_in_out_t
```

13.9.3.9 atca_sign_internal_in_out_t

```
typedef struct atca_sign_internal_in_out atca_sign_internal_in_out_t
```

Input/output parameters for calculating the message and digest used by the Sign(internal) command. Used with the atcah_sign_internal_msg() function.

13.9.3.10 atca_temp_key_t

```
typedef struct atca_temp_key atca_temp_key_t
```

Structure to hold TempKey fields.

13.9.3.11 atca_verify_in_out_t

```
typedef struct atca_verify_in_out atca_verify_in_out_t
```

13.9.3.12 atca_verify_mac_in_out_t

```
typedef struct atca_verify_mac atca_verify_mac_in_out_t
```

13.9.3.13 atca_write_mac_in_out_t

```
typedef struct atca_write_mac_in_out atca_write_mac_in_out_t
```

Input/output parameters for function atcah_write_auth_mac() and atcah_privwrite_auth_mac().

13.9.4 Function Documentation

13.9.4.1 atcah_check_mac()

```
ATCA_STATUS atcah_check_mac (
            struct atca_check_mac_in_out * param )
```

This function performs the checkmac operation to generate client response on the host side .

Parameters

in,out	*param*	Input and output parameters

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.9.4.2 atcah_config_to_sign_internal()

```
ATCA_STATUS atcah_config_to_sign_internal (
            ATCADeviceType device_type,
            struct atca_sign_internal_in_out * param,
            const uint8_t * config )
```

Populate the slot_config, key_config, and is_slot_locked fields in the atca_sign_internal_in_out structure from the provided config zone.

The atca_sign_internal_in_out structure has a number of fields (slot_config, key_config, is_slot_locked) that can be determined automatically from the current state of TempKey and the full config zone.

Parameters

in,out	*param*	Sign(Internal) parameters to be filled out. Only slot_config, key_config, and is_slot_locked will be set.
in	*device_type*	The type of the device.
in	*config*	Full 128 byte config zone for the device.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.9.4.3 atcah_decrypt()

```
ATCA_STATUS atcah_decrypt (
            struct atca_decrypt_in_out * param )
```

This function decrypts 32-byte encrypted data received with the Read command.

To use this function, first the nonce must be valid and synchronized between device and application. The application sends a GenDig command to the Device, using a key specified by SlotConfig.ReadKey. The device updates its TempKey. The application then updates its own TempKey using the GenDig calculation function, using the same key. The application sends a Read command to the device for a user zone configured with EncryptRead. The device encrypts 32-byte zone content, and outputs it to the host. The application passes these encrypted data to this decryption function. The function decrypts the data and returns them. TempKey must be updated by GenDig using a ParentKey as specified by SlotConfig.ReadKey before executing this function. The decryption function does not check whether the TempKey has been generated by a correct ParentKey for the corresponding zone. Therefore to get a correct result, the application has to make sure that prior GenDig calculation was done using correct ParentKey.

Parameters

in,out	*param*	pointer to parameter structure

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.9.4.4 atcah_derive_key()

```
ATCA_STATUS atcah_derive_key (
            struct atca_derive_key_in_out * param )
```

This function derives a key with a key and TempKey.

Used in conjunction with DeriveKey command, the key derived by this function will match the key in the device. Two kinds of operation are supported:

- Roll Key operation: target_key and parent_key parameters should be set to point to the same location (TargetKey).
- Create Key operation: target_key should be set to point to TargetKey, parent_key should be set to point to ParentKey.

After executing this function, the initial value of target_key will be overwritten with the derived key. The TempKey should be valid (temp_key.valid = 1) before executing this function.

Parameters

in,out	*param*	pointer to parameter structure

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.9.4.5 atcah_derive_key_mac()

```
ATCA_STATUS atcah_derive_key_mac (
            struct atca_derive_key_mac_in_out * param )
```

This function calculates the input MAC for a DeriveKey command.

The DeriveKey command will need an input MAC if SlotConfig[TargetKey].Bit15 is set.

Parameters

in,out	*param*	pointer to parameter structure

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.9.4.6 atcah_encode_counter_match()

```
ATCA_STATUS atcah_encode_counter_match (
            uint32_t counter_value,
            uint8_t * counter_match_value )
```

Builds the counter match value that needs to be stored in a slot.

Parameters

in	*counter_value*	Counter value to be used for the counter match. This must be a multiple of 32.
out	*counter_match_value*	Data to be stored in the beginning of a counter match slot will be returned here (8 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.9.4.7 atcah_gen_dig()

```
ATCA_STATUS atcah_gen_dig (
            struct atca_gen_dig_in_out * param )
```

This function combines the current TempKey with a stored value.

The stored value can be a data slot, OTP page, configuration zone, or hardware transport key. The TempKey generated by this function will match with the TempKey in the device generated when executing a GenDig command. The TempKey should be valid (temp_key.valid = 1) before executing this function. To use this function, an application first sends a GenDig command with a chosen stored value to the device. This stored value must be known by the application and is passed to this GenDig calculation function. The function calculates a new TempKey and returns it.

Parameters

in,out	*param*	pointer to parameter structure

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.9.4.8 atcah_gen_key_msg()

```
ATCA_STATUS atcah_gen_key_msg (
            struct atca_gen_key_in_out * param )
```

Calculate the PubKey digest created by GenKey and saved to TempKey.

Parameters

in,out	*param*	GenKey parameters required to calculate the PubKey digest. Digest is return in the temp_key parameter.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.9.4.9 atcah_gen_mac()

```
ATCA_STATUS atcah_gen_mac (
            struct atca_gen_dig_in_out * param )
```

This function generates mac with session key with a plain text.

Parameters

in,out	*param*	pointer to parameter structure

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.9.4.10 atcah_hmac()

```
ATCA_STATUS atcah_hmac (
            struct atca_hmac_in_out * param )
```

This function generates an HMAC / SHA-256 hash of a key and other information.

The resulting hash will match with the one generated in the device by an HMAC command. The TempKey has to be valid (temp_key.valid = 1) before executing this function.

Parameters

in,out	*param*	pointer to parameter structure

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.9.4.11 atcah_include_data()

```
uint8_t* atcah_include_data (
            struct atca_include_data_in_out * param )
```

This function copies otp and sn data into a command buffer.

Parameters

in,out	*param*	pointer to parameter structure

Returns

pointer to command buffer byte that was copied last

13.9.4.12 atcah_io_decrypt()

```
ATCA_STATUS atcah_io_decrypt (
            struct atca_io_decrypt_in_out * param )
```

Decrypt data that's been encrypted by the IO protection key. The ECDH and KDF commands on the ATECC608A are the only ones that support this operation.

Parameters

in,out	*param*	Parameters required to perform the operation.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.9.4.13 atcah_mac()

```
ATCA_STATUS atcah_mac (
            struct atca_mac_in_out * param )
```

This function generates an SHA-256 digest (MAC) of a key, challenge, and other information.

The resulting digest will match with the one generated by the device when executing a MAC command. The Temp↩ Key (if used) should be valid (temp_key.valid = 1) before executing this function.

Parameters

in,out	*param*	pointer to parameter structure

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.9.4.14 atcah_nonce()

```
ATCA_STATUS atcah_nonce (
            struct atca_nonce_in_out * param )
```

This function calculates host side nonce with the parameters passed.

Parameters

in,out	*param*	pointer to parameter structure

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.9.4.15 atcah_privwrite_auth_mac()

```
ATCA_STATUS atcah_privwrite_auth_mac (
            struct atca_write_mac_in_out * param )
```

This function calculates the input MAC for the PrivWrite command.

The PrivWrite command will need an input MAC if SlotConfig.WriteConfig.Encrypt is set.

Parameters

in,out	*param*	pointer to parameter structure

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.9.4.16 atcah_secureboot_enc()

```
ATCA_STATUS atcah_secureboot_enc (
            atca_secureboot_enc_in_out_t * param )
```

Encrypts the digest for the SecureBoot command when using the encrypted digest / validating mac option.

Parameters

in,out	*param*	Data required to perform the operation.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.9.4.17 atcah_secureboot_mac()

```
ATCA_STATUS atcah_secureboot_mac (
            atca_secureboot_mac_in_out_t * param )
```

Calculates the expected MAC returned from the SecureBoot command when verification is a success.

The result of this function (param->mac) should be compared with the actual MAC returned to validate the response.

Parameters

in,out	*param*	Data required to perform the operation.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.9.4.18 atcah_sha256()

```
ATCA_STATUS atcah_sha256 (
            int32_t len,
            const uint8_t * message,
            uint8_t * digest )
```

This function creates a SHA256 digest on a little-endian system.

Parameters

in	*len*	byte length of message
in	*message*	pointer to message
out	*digest*	SHA256 of message

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.9.4.19 atcah_sign_internal_msg()

```
ATCA_STATUS atcah_sign_internal_msg (
            ATCADeviceType device_type,
            struct atca_sign_internal_in_out * param )
```

Builds the full message that would be signed by the Sign(Internal) command.

Additionally, the function will optionally output the OtherData data required by the Verify(In/Validate) command as well as the SHA256 digest of the full message.

Parameters

out	*device_type*	Device type to perform the calculation for.
out	*param*	Input data and output buffers required.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.9.4.20 atcah_verify_mac()

```
ATCA_STATUS atcah_verify_mac (
            atca_verify_mac_in_out_t * param )
```

Calculate the expected MAC on the host side for the Verify command.

Parameters

in,out	*param*	Data required to perform the operation.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.9.4.21 atcah_write_auth_mac()

```
ATCA_STATUS atcah_write_auth_mac (
            struct atca_write_mac_in_out * param )
```

This function calculates the input MAC for the Write command.

The Write command will need an input MAC if SlotConfig.WriteConfig.Encrypt is set.

Parameters

in,out	*param*	pointer to parameter structure

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.9.5 Variable Documentation

13.9.5.1 challenge

`challenge`

[in] Pointer to 32-byte Challenge data used in MAC command, depending on mode.

13.9.5.2 crypto_data

`crypto_data`

[in,out] Pointer to 32-byte data. Input encrypted data from Read command (Contents field), output decrypted.

13.9.5.3 curve_type

`curve_type`

[in] Curve type used in Verify command (Param2).

13.9.5.4 key `[1/2]`

`key`

[in] Pointer to 32-byte key used to generate MAC digest.

13.9.5.5 key [2/2]

`key`

[in] Pointer to 32-byte key used to generate HMAC digest.

13.9.5.6 key_id [1/2]

`key_id`

[in] KeyID parameter used in MAC command (Param2).

13.9.5.7 key_id [2/2]

`key_id`

[in] KeyID parameter used in HMAC command (Param2).

13.9.5.8 mode [1/3]

`mode`

[in] Mode parameter used in Nonce command (Param1).

13.9.5.9 mode [2/3]

`mode`

[in] Mode parameter used in MAC command (Param1).

13.9.5.10 mode [3/3]

`mode`

[in] Mode parameter used in HMAC command (Param1).

13.9.5.11 num_in

`num_in`

[in] Pointer to 20-byte NumIn data used in Nonce command.

13.9.5.12 otp [1/3]

`otp`

[in] pointer to one-time-programming data

13.9.5.13 otp [2/3]

`otp`

[in] Pointer to 11-byte OTP, optionally included in MAC digest, depending on mode.

13.9.5.14 otp [3/3]

`otp`

[in] Pointer to 11-byte OTP, optionally included in HMAC digest, depending on mode.

13.9.5.15 p_temp

`p_temp`

[out] pointer to output buffer

13.9.5.16 public_key

`public_key`

[in] Pointer to the public key to be used for verification

13.9.5.17 rand_out

`rand_out`

[in] Pointer to 32-byte RandOut data from Nonce command.

13.9.5.18 response [1/2]

`response`

[out] Pointer to 32-byte SHA-256 digest (MAC).

13.9.5.19 response [2/2]

`response`

[out] Pointer to 32-byte SHA-256 HMAC digest.

13.9.5.20 signature

`signature`

[in] Pointer to ECDSA signature to be verified

13.9.5.21 sn [1/3]

`sn`

[in] pointer to serial number data

13.9.5.22 sn [2/3]

`sn`

[in] Pointer to 9-byte SN, optionally included in MAC digest, depending on mode.

13.9.5.23 sn [3/3]

sn

[in] Pointer to 9-byte SN, optionally included in HMAC digest, depending on mode.

13.9.5.24 temp_key [1/5]

temp_key

[in,out] Pointer to TempKey structure.

13.9.5.25 temp_key [2/5]

temp_key

[in,out] Pointer to TempKey structure.

13.9.5.26 temp_key [3/5]

temp_key

[in,out] Pointer to TempKey structure.

13.9.5.27 temp_key [4/5]

temp_key

[in,out] Pointer to TempKey structure.

13.9.5.28 temp_key [5/5]

temp_key

[in,out] Pointer to TempKey structure.

13.9.5.29 zero

zero

[in] Zero parameter used in Nonce command (Param2).

13.10 JSON Web Token (JWT) methods (atca_jwt_)

Methods for signing and verifying JSON Web Token (JWT) tokens.

Data Structures

- struct atca_jwt_t
    Structure to hold metadata information about the jwt being built.

Functions

- ATCA_STATUS atca_jwt_init (atca_jwt_t ∗jwt, char ∗buf, uint16_t buflen)
    Initialize a JWT structure.
- ATCA_STATUS atca_jwt_add_claim_string (atca_jwt_t ∗jwt, const char ∗claim, const char ∗value)
    Add a string claim to a token.
- ATCA_STATUS atca_jwt_add_claim_numeric (atca_jwt_t ∗jwt, const char ∗claim, int32_t value)
    Add a numeric claim to a token.
- ATCA_STATUS atca_jwt_finalize (atca_jwt_t ∗jwt, uint16_t key_id)
    Close the claims of a token, encode them, then sign the result.
- void atca_jwt_check_payload_start (atca_jwt_t ∗jwt)
    Check the provided context to see what character needs to be added in order to append a claim.
- ATCA_STATUS atca_jwt_verify (const char ∗buf, uint16_t buflen, const uint8_t ∗pubkey)
    Verifies the signature of a jwt using the provided public key.

13.10.1 Detailed Description

Methods for signing and verifying JSON Web Token (JWT) tokens.

13.10.2 Function Documentation

13.10.2.1 atca_jwt_add_claim_numeric()

```
ATCA_STATUS atca_jwt_add_claim_numeric (
            atca_jwt_t * jwt,
            const char * claim,
            int32_t value )
```

Add a numeric claim to a token.

Note

This function does not escape strings so the user has to ensure the claim is valid first

Parameters

in	*jwt*	JWT Context to use
in	*claim*	Name of the claim to be inserted
in	*value*	integer value to be inserted

13.10.2.2 atca_jwt_add_claim_string()

```
ATCA_STATUS atca_jwt_add_claim_string (
            atca_jwt_t * jwt,
            const char * claim,
            const char * value )
```

Add a string claim to a token.

Note

This function does not escape strings so the user has to ensure they are valid for use in a JSON string first

Parameters

in	*jwt*	JWT Context to use
in	*claim*	Name of the claim to be inserted
in	*value*	Null terminated string to be insterted

13.10.2.3 atca_jwt_check_payload_start()

```
void atca_jwt_check_payload_start (
            atca_jwt_t * jwt )
```

Check the provided context to see what character needs to be added in order to append a claim.

Parameters

in	*jwt*	JWT Context to use

13.10.2.4 atca_jwt_finalize()

```
ATCA_STATUS atca_jwt_finalize (
            atca_jwt_t * jwt,
            uint16_t key_id )
```

Close the claims of a token, encode them, then sign the result.

Parameters

in	*jwt*	JWT Context to use
in	*key_id*	Key Id (Slot number) used to sign

13.10.2.5 atca_jwt_init()

```
ATCA_STATUS atca_jwt_init (
            atca_jwt_t * jwt,
            char * buf,
            uint16_t buflen )
```

Initialize a JWT structure.

Parameters

in	*jwt*	JWT Context to initialize
in,out	*buf*	Pointer to a buffer to store the token
in	*buflen*	Length of the buffer

13.10.2.6 atca_jwt_verify()

```
ATCA_STATUS atca_jwt_verify (
            const char * buf,
            uint16_t buflen,
            const uint8_t * pubkey )
```

Verifies the signature of a jwt using the provided public key.

Parameters

in	*buf*	Buffer holding an encoded jwt
in	*buflen*	Length of the buffer/jwt
in	*pubkey*	Public key (raw byte format)

13.11 mbedTLS Wrapper methods (atca_mbedtls_)

These methods are for interfacing cryptoauthlib to mbedtls.

Functions

- int atca_mbedtls_pk_init (struct mbedtls_pk_context *pkey, const uint16_t slotid)

  Initializes an mbedtls pk context for use with EC operations.
- int atca_mbedtls_cert_add (struct mbedtls_x509_crt *cert, const struct atcacert_def_s *cert_def)
- int atca_mbedtls_ecdh_slot_cb (void)

  ECDH Callback to obtain the "slot" used in ECDH operations from the application.
- int atca_mbedtls_ecdh_ioprot_cb (uint8_t secret[32])

  ECDH Callback to obtain the IO Protection secret from the application.

13.11.1 Detailed Description

These methods are for interfacing cryptoauthlib to mbedtls.

13.11.2 Function Documentation

13.11.2.1 atca_mbedtls_cert_add()

```
int atca_mbedtls_cert_add (
            struct mbedtls_x509_crt * cert,
            const struct atcacert_def_s * cert_def )
```

13.11.2.2 atca_mbedtls_ecdh_ioprot_cb()

```
int atca_mbedtls_ecdh_ioprot_cb (
            uint8_t secret[32] )
```

ECDH Callback to obtain the IO Protection secret from the application.

Parameters

out	*secret*	32 byte array used to store the secret

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.11.2.3 atca_mbedtls_ecdh_slot_cb()

```
int atca_mbedtls_ecdh_slot_cb (
            void  )
```

ECDH Callback to obtain the "slot" used in ECDH operations from the application.

Returns

Slot Number

13.11.2.4 atca_mbedtls_pk_init()

```
int atca_mbedtls_pk_init (
            mbedtls_pk_context * pkey,
            const uint16_t slotid )
```

Initializes an mbedtls pk context for use with EC operations.

Parameters

in,out	*pkey*	ptr to space to receive version string
in	*slotid*	Associated with this key

Returns

0 on success, otherwise an error code.

13.12 TNG API (tng_)

These methods provide some convenience functions (mostly around certificates) for TNG devices, which currently include ATECC608A-MAHTN-T.

Macros

- #define TNG22_PRIMARY_KEY_SLOT 0
- #define TNGTN_PRIMARY_KEY_SLOT 1

Enumerations

- enum tng_type_t { TNGTYPE_UNKNOWN, TNGTYPE_22, TNGTYPE_TN }

Functions

- ATCA_STATUS tng_get_type (tng_type_t ∗type)

  Get the type of TNG device.
- ATCA_STATUS tng_get_device_pubkey (uint8_t ∗public_key)

  Uses GenKey command to calculate the public key from the primary device public key.

- const atcacert_def_t g_tng22_cert_def_1_signer
- #define TNG22_CERT_TEMPLATE_1_SIGNER_SIZE 520

- const atcacert_def_t g_tng22_cert_def_2_device
- #define TNG22_CERT_TEMPLATE_2_DEVICE_SIZE 505
- #define TNG22_CERT_ELEMENTS_2_DEVICE_COUNT 2

- int tng_atcacert_max_device_cert_size (size_t ∗max_cert_size)

  Return the maximum possible certificate size in bytes for a TNG device certificate. Certificate can be variable size, so this gives an appropriate buffer size when reading the certificate.
- int tng_atcacert_read_device_cert (uint8_t ∗cert, size_t ∗cert_size, const uint8_t ∗signer_cert)

  Reads the device certificate for a TNG device.
- int tng_atcacert_device_public_key (uint8_t ∗public_key, uint8_t ∗cert)

  Reads the device public key.
- int tng_atcacert_max_signer_cert_size (size_t ∗max_cert_size)

  Return the maximum possible certificate size in bytes for a TNG signer certificate. Certificate can be variable size, so this gives an appropriate buffer size when reading the certificate.
- int tng_atcacert_read_signer_cert (uint8_t ∗cert, size_t ∗cert_size)

  Reads the signer certificate for a TNG device.
- int tng_atcacert_signer_public_key (uint8_t ∗public_key, uint8_t ∗cert)

  Reads the signer public key.
- int tng_atcacert_root_cert_size (size_t ∗cert_size)

  Get the size of the TNG root cert.
- int tng_atcacert_root_cert (uint8_t ∗cert, size_t ∗cert_size)

  Get the TNG root cert.
- int tng_atcacert_root_public_key (uint8_t ∗public_key)

  Gets the root public key.

- const uint8_t g_cryptoauth_root_ca_002_cert [ ]
- const size_t g_cryptoauth_root_ca_002_cert_size
- #define CRYPTOAUTH_ROOT_CA_002_PUBLIC_KEY_OFFSET 266

- const atcacert_def_t g_tngtn_cert_def_1_signer

- const atcacert_def_t g_tngtn_cert_def_2_device

13.12.1 Detailed Description

These methods provide some convenience functions (mostly around certificates) for TNG devices, which currently include ATECC608A-MAHTN-T.

13.12.2 Macro Definition Documentation

13.12.2.1 CRYPTOAUTH_ROOT_CA_002_PUBLIC_KEY_OFFSET

```
#define CRYPTOAUTH_ROOT_CA_002_PUBLIC_KEY_OFFSET 266
```

13.12.2.2 TNG22_CERT_ELEMENTS_2_DEVICE_COUNT

```
#define TNG22_CERT_ELEMENTS_2_DEVICE_COUNT 2
```

13.12.2.3 TNG22_CERT_TEMPLATE_1_SIGNER_SIZE

```
#define TNG22_CERT_TEMPLATE_1_SIGNER_SIZE 520
```

13.12.2.4 TNG22_CERT_TEMPLATE_2_DEVICE_SIZE

```
#define TNG22_CERT_TEMPLATE_2_DEVICE_SIZE 505
```

13.12.2.5 TNG22_PRIMARY_KEY_SLOT

```
#define TNG22_PRIMARY_KEY_SLOT 0
```

13.12.2.6 TNGTN_PRIMARY_KEY_SLOT

```
#define TNGTN_PRIMARY_KEY_SLOT 1
```

13.12.3 Enumeration Type Documentation

13.12.3.1 tng_type_t

```
enum tng_type_t
```

Enumerator

TNGTYPE_UNKNOWN	
TNGTYPE_22	
TNGTYPE_TN	

13.12.4 Function Documentation

13.12.4.1 tng_atcacert_device_public_key()

```
int tng_atcacert_device_public_key (
            uint8_t * public_key,
            uint8_t * cert )
```

Reads the device public key.

Parameters

out	*public_key*	Public key will be returned here. Format will be the X and Y integers in big-endian format. 64 bytes for P256 curve.
in	*cert*	If supplied, the device public key is used from this certificate. If set to NULL, the device public key is read from the device.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.12.4.2 tng_atcacert_max_device_cert_size()

```
int tng_atcacert_max_device_cert_size (
            size_t * max_cert_size )
```

Return the maximum possible certificate size in bytes for a TNG device certificate. Certificate can be variable size, so this gives an appropriate buffer size when reading the certificate.

Parameters

out	*max_cert_size*	Maximum certificate size will be returned here in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.12.4.3 tng_atcacert_max_signer_cert_size()

```
int tng_atcacert_max_signer_cert_size (
            size_t * max_cert_size )
```

Return the maximum possible certificate size in bytes for a TNG signer certificate. Certificate can be variable size, so this gives an appropriate buffer size when reading the certificate.

Parameters

out	*max_cert_size*	Maximum certificate size will be returned here in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.12.4.4 tng_atcacert_read_device_cert()

```
int tng_atcacert_read_device_cert (
            uint8_t * cert,
            size_t * cert_size,
            const uint8_t * signer_cert )
```

Reads the device certificate for a TNG device.

Parameters

out	*cert*	Buffer to received the certificate (DER format).
in,out	*cert_size*	As input, the size of the cert buffer in bytes. As output, the size of the certificate returned in cert in bytes.
in	*signer_cert*	If supplied, the signer public key is used from this certificate. If set to NULL, the signer public key is read from the device.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.12.4.5 tng_atcacert_read_signer_cert()

```
int tng_atcacert_read_signer_cert (
            uint8_t * cert,
            size_t * cert_size )
```

Reads the signer certificate for a TNG device.

Parameters

out	*cert*	Buffer to received the certificate (DER format).
in,out	*cert_size*	As input, the size of the cert buffer in bytes. As output, the size of the certificate returned in cert in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.12.4.6 tng_atcacert_root_cert()

```
int tng_atcacert_root_cert (
            uint8_t * cert,
            size_t * cert_size )
```

Get the TNG root cert.

Parameters

out	*cert*	Buffer to received the certificate (DER format).
in,out	*cert_size*	As input, the size of the cert buffer in bytes. As output, the size of the certificate returned in cert in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.12.4.7 tng_atcacert_root_cert_size()

```
int tng_atcacert_root_cert_size (
            size_t * cert_size )
```

Get the size of the TNG root cert.

Parameters

out	*cert_size*	Certificate size will be returned here in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.12.4.8 tng_atcacert_root_public_key()

```
int tng_atcacert_root_public_key (
            uint8_t * public_key )
```

Gets the root public key.

Parameters

out	*public_key*	Public key will be returned here. Format will be the X and Y integers in big-endian format. 64 bytes for P256 curve.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.12.4.9 tng_atcacert_signer_public_key()

```
int tng_atcacert_signer_public_key (
            uint8_t * public_key,
            uint8_t * cert )
```

Reads the signer public key.

Parameters

out	*public_key*	Public key will be returned here. Format will be the X and Y integers in big-endian format. 64 bytes for P256 curve.
in	*cert*	If supplied, the signer public key is used from this certificate. If set to NULL, the signer public key is read from the device.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

13.12.4.10 tng_get_device_pubkey()

```
ATCA_STATUS tng_get_device_pubkey (
            uint8_t * public_key )
```

Uses GenKey command to calculate the public key from the primary device public key.

Parameters

out	*public_key*	Public key will be returned here. Format will be the X and Y integers in big-endian format. 64 bytes for P256 curve.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.12.4.11 tng_get_type()

```
ATCA_STATUS tng_get_type (
            tng_type_t * type )
```

Get the type of TNG device.

Parameters

out	*type*	TNG device type is returned here.

Returns

ATCA_SUCCESS on success, otherwise an error code.

13.12.5 Variable Documentation

13.12.5.1 g_cryptoauth_root_ca_002_cert

```
const uint8_t g_cryptoauth_root_ca_002_cert[]
```

13.12.5.2 g_cryptoauth_root_ca_002_cert_size

```
const size_t g_cryptoauth_root_ca_002_cert_size
```

13.12.5.3 g_tng22_cert_def_1_signer

```
const atcacert_def_t g_tng22_cert_def_1_signer
```

13.12.5.4 g_tng22_cert_def_2_device

```
const atcacert_def_t g_tng22_cert_def_2_device
```

13.12.5.5 g_tngtn_cert_def_1_signer

```
const atcacert_def_t g_tngtn_cert_def_1_signer
```

13.12.5.6 g_tngtn_cert_def_2_device

```
const atcacert_def_t g_tngtn_cert_def_2_device
```

Chapter 14

Data Structure Documentation

14.1 atca_aes_cbc_ctx Struct Reference

```
#include <atca_basic.h>
```

Data Fields

- uint16_t key_id
  Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
- uint8_t key_block
  Index of the 16-byte block to use within the key location for the actual key.
- uint8_t ciphertext [AES_DATA_SIZE]
  Ciphertext from last operation.

14.1.1 Field Documentation

14.1.1.1 ciphertext

```
uint8_t ciphertext[AES_DATA_SIZE]
```

Ciphertext from last operation.

14.1.1.2 key_block

```
uint8_t key_block
```

Index of the 16-byte block to use within the key location for the actual key.

14.1.1.3 key_id

```
uint16_t key_id
```

Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.

14.2 atca_aes_cmac_ctx Struct Reference

```
#include <atca_basic.h>
```

Data Fields

- atca_aes_cbc_ctx_t cbc_ctx
  CBC context.
- uint32_t block_size
  Number of bytes in current block.
- uint8_t block [AES_DATA_SIZE]
  Unprocessed message storage.

14.2.1 Field Documentation

14.2.1.1 block

```
uint8_t block[AES_DATA_SIZE]
```

Unprocessed message storage.

14.2.1.2 block_size

```
uint32_t block_size
```

Number of bytes in current block.

14.2.1.3 cbc_ctx

```
atca_aes_cbc_ctx_t cbc_ctx
```

CBC context.

14.3 atca_aes_ctr_ctx Struct Reference

```
#include <atca_basic.h>
```

Data Fields

- uint16_t key_id

  Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
- uint8_t key_block

  Index of the 16-byte block to use within the key location for the actual key.
- uint8_t cb [AES_DATA_SIZE]

  Counter block, comprises of nonce + count value (16 bytes).
- uint8_t counter_size

  Size of counter in the initialization vector.

14.3.1 Field Documentation

14.3.1.1 cb

```
uint8_t cb[AES_DATA_SIZE]
```

Counter block, comprises of nonce + count value (16 bytes).

14.3.1.2 counter_size

```
uint8_t counter_size
```

Size of counter in the initialization vector.

14.3.1.3 key_block

```
uint8_t key_block
```

Index of the 16-byte block to use within the key location for the actual key.

14.3.1.4 key_id

```
uint16_t key_id
```

Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.

14.4 atca_aes_gcm_ctx Struct Reference

```
#include <atca_basic_aes_gcm.h>
```

Data Fields

- uint16_t key_id

    Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
- uint8_t key_block

    Index of the 16-byte block to use within the key location for the actual key.
- uint8_t cb [AES_DATA_SIZE]

    Counter block, comprises of nonce + count value (16 bytes).
- uint32_t data_size

    Size of the data being encrypted/decrypted in bytes.
- uint32_t aad_size

    Size of the additional authenticated data in bytes.
- uint8_t h [AES_DATA_SIZE]

    Subkey for ghash functions in GCM.
- uint8_t j0 [AES_DATA_SIZE]

    Precounter block generated from IV.
- uint8_t y [AES_DATA_SIZE]

    Current GHASH output.
- uint8_t partial_aad [AES_DATA_SIZE]

    Partial blocks of data waiting to be processed.
- uint32_t partial_aad_size

    Amount of data in the partial block buffer.
- uint8_t enc_cb [AES_DATA_SIZE]

    Last encrypted counter block.
- uint8_t ciphertext_block [AES_DATA_SIZE]

    Last ciphertext block.

14.4.1 Detailed Description

Context structure for AES GCM operations.

14.4.2 Field Documentation

14.4.2.1 aad_size

```
uint32_t aad_size
```

Size of the additional authenticated data in bytes.

14.4.2.2 cb

```
uint8_t cb[AES_DATA_SIZE]
```

Counter block, comprises of nonce + count value (16 bytes).

14.4.2.3 ciphertext_block

```
uint8_t ciphertext_block[AES_DATA_SIZE]
```

Last ciphertext block.

14.4.2.4 data_size

```
uint32_t data_size
```

Size of the data being encrypted/decrypted in bytes.

14.4.2.5 enc_cb

```
uint8_t enc_cb[AES_DATA_SIZE]
```

Last encrypted counter block.

14.4.2.6 h

```
uint8_t h[AES_DATA_SIZE]
```

Subkey for ghash functions in GCM.

14.4.2.7 j0

```
uint8_t j0[AES_DATA_SIZE]
```

Precounter block generated from IV.

14.4.2.8 key_block

```
uint8_t key_block
```

Index of the 16-byte block to use within the key location for the actual key.

14.4.2.9 key_id

```
uint16_t key_id
```

Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.

14.4.2.10 partial_aad

```
uint8_t partial_aad[AES_DATA_SIZE]
```

Partial blocks of data waiting to be processed.

14.4.2.11 partial_aad_size

```
uint32_t partial_aad_size
```

Amount of data in the partial block buffer.

14.4.2.12 y

```
uint8_t y[AES_DATA_SIZE]
```

Current GHASH output.

14.5 atca_check_mac_in_out Struct Reference

Input/output parameters for function atcah_check_mac().

```
#include <atca_host.h>
```

Data Fields

- uint8_t mode

  [in] CheckMac command Mode
- uint16_t key_id

  [in] CheckMac command KeyID
- const uint8_t ∗ sn

  [in] Device serial number SN[0:8]. Only SN[0:1] and SN[8] are required though.
- const uint8_t ∗ client_chal

  [in] ClientChal data, 32 bytes. Can be NULL if mode[0] is 1.
- uint8_t ∗ client_resp

  [out] Calculated ClientResp will be returned here.
- const uint8_t ∗ other_data

  [in] OtherData, 13 bytes
- const uint8_t ∗ otp

  [in] First 8 bytes of the OTP zone data. Can be NULL is mode[5] is 0.
- const uint8_t ∗ slot_key
- const uint8_t ∗ target_key
- struct atca_temp_key ∗ temp_key

  [in,out] Current state of TempKey. Required if mode[0] or mode[1] are 1.

14.5.1 Detailed Description

Input/output parameters for function atcah_check_mac().

14.5.2 Field Documentation

14.5.2.1 client_chal

```
const uint8_t* client_chal
```

[in] ClientChal data, 32 bytes. Can be NULL if mode[0] is 1.

14.5.2.2 client_resp

```
uint8_t* client_resp
```

[out] Calculated ClientResp will be returned here.

14.5.2.3 key_id

```
uint16_t key_id
```

[in] CheckMac command KeyID

14.5.2.4 mode

```
uint8_t mode
```

[in] CheckMac command Mode

14.5.2.5 other_data

```
const uint8_t* other_data
```

[in] OtherData, 13 bytes

14.5.2.6 otp

```
const uint8_t* otp
```

[in] First 8 bytes of the OTP zone data. Can be NULL is mode[5] is 0.

14.5.2.7 slot_key

```
const uint8_t* slot_key
```

[in] 32 byte key value in the slot specified by slot_id. Can be NULL if mode[1] is 1.

14.5.2.8 sn

```
const uint8_t* sn
```

[in] Device serial number SN[0:8]. Only SN[0:1] and SN[8] are required though.

14.5.2.9 target_key

```
const uint8_t* target_key
```

[in] If this is not NULL, it assumes CheckMac copy is enabled for the specified key_id (ReadKey=0). If key_id is even, this should be the 32-byte key value for the slot key_id+1, otherwise this should be set to slot_key.

14.5.2.10 temp_key

```
struct atca_temp_key* temp_key
```

[in,out] Current state of TempKey. Required if mode[0] or mode[1] are 1.

14.6 atca_command Struct Reference

atca_command is the C object backing ATCACommand.

```
#include <atca_command.h>
```

Data Fields

- ATCADeviceType dt
- uint8_t clock_divider
- uint16_t execution_time_msec

14.6.1 Detailed Description

atca_command is the C object backing ATCACommand.

14.6.2 Field Documentation

14.6.2.1 clock_divider

```
uint8_t clock_divider
```

14.6.2.2 dt

```
ATCADeviceType dt
```

14.6.2.3 execution_time_msec

```
uint16_t execution_time_msec
```

14.7 atca_decrypt_in_out Struct Reference

Input/output parameters for function atca_decrypt().

```
#include <atca_host.h>
```

Data Fields

- uint8_t ∗ crypto_data
  [in,out] Pointer to 32-byte data. Input encrypted data from Read command (Contents field), output decrypted.
- struct atca_temp_key ∗ temp_key
  [in,out] Pointer to TempKey structure.

14.7.1 Detailed Description

Input/output parameters for function atca_decrypt().

14.8 atca_derive_key_in_out Struct Reference

Input/output parameters for function atcah_derive_key().

```
#include <atca_host.h>
```

Data Fields

- uint8_t mode
  Mode (param 1) of the derive key command.
- uint16_t target_key_id
  Key ID (param 2) of the target slot to run the command on.
- const uint8_t ∗ sn
  Device serial number SN[0:8]. Only SN[0:1] and SN[8] are required though.
- const uint8_t ∗ parent_key
  Parent key to be used in the derive key calculation (32 bytes).
- uint8_t ∗ target_key
  Derived key will be returned here (32 bytes).
- struct atca_temp_key ∗ temp_key
  Current state of TempKey.

14.8.1 Detailed Description

Input/output parameters for function atcah_derive_key().

14.8.2 Field Documentation

14.8.2.1 mode

```
uint8_t mode
```

Mode (param 1) of the derive key command.

14.8.2.2 parent_key

```
const uint8_t* parent_key
```

Parent key to be used in the derive key calculation (32 bytes).

14.8.2.3 sn

```
const uint8_t* sn
```

Device serial number SN[0:8]. Only SN[0:1] and SN[8] are required though.

14.8.2.4 target_key

```
uint8_t* target_key
```

Derived key will be returned here (32 bytes).

14.8.2.5 target_key_id

```
uint16_t target_key_id
```

Key ID (param 2) of the target slot to run the command on.

14.8.2.6 temp_key

```
struct atca_temp_key* temp_key
```

Current state of TempKey.

14.9 atca_derive_key_mac_in_out Struct Reference

Input/output parameters for function atcah_derive_key_mac().

```
#include <atca_host.h>
```

Data Fields

- uint8_t mode

    Mode (param 1) of the derive key command.
- uint16_t target_key_id

    Key ID (param 2) of the target slot to run the command on.
- const uint8_t * sn

    Device serial number SN[0:8]. Only SN[0:1] and SN[8] are required though.
- const uint8_t * parent_key

    Parent key to be used in the derive key calculation (32 bytes).
- uint8_t * mac

    DeriveKey MAC will be returned here.

14.9.1 Detailed Description

Input/output parameters for function atcah_derive_key_mac().

14.9.2 Field Documentation

14.9.2.1 mac

```
uint8_t* mac
```

DeriveKey MAC will be returned here.

14.9.2.2 mode

```
uint8_t mode
```

Mode (param 1) of the derive key command.

14.9.2.3 parent_key

```
const uint8_t* parent_key
```

Parent key to be used in the derive key calculation (32 bytes).

14.9.2.4 sn

```
const uint8_t* sn
```

Device serial number SN[0:8]. Only SN[0:1] and SN[8] are required though.

14.9.2.5 target_key_id

```
uint16_t target_key_id
```

Key ID (param 2) of the target slot to run the command on.

14.10 atca_device Struct Reference

atca_device is the C object backing ATCADevice. See the atca_device.h file for details on the ATCADevice methods.

```
#include <atca_device.h>
```

Data Fields

- ATCACommand mCommands

  Command set for a given CryptoAuth device.
- ATCAIface mIface

  Physical interface.

14.10.1 Detailed Description

atca_device is the C object backing ATCADevice. See the atca_device.h file for details on the ATCADevice methods.

14.10.2 Field Documentation

14.10.2.1 mCommands

`ATCACommand mCommands`

Command set for a given CryptoAuth device.

14.10.2.2 mIface

`ATCAIface mIface`

Physical interface.

14.11 atca_gen_dig_in_out Struct Reference

Input/output parameters for function atcah_gen_dig().

```
#include <atca_host.h>
```

Data Fields

- uint8_t zone
  [in] Zone/Param1 for the GenDig command
- uint16_t key_id
  [in] KeyId/Param2 for the GenDig command
- uint16_t slot_conf
  [in] Slot config for the GenDig command
- uint16_t key_conf
  [in] Key config for the GenDig command
- uint8_t slot_locked
  [in] slot locked for the GenDig command
- uint32_t counter
  [in] counter for the GenDig command
- bool is_key_nomac
  [in] Set to true if the slot pointed to be key_id has the SotConfig.NoMac bit set
- const uint8_t ∗ sn
  [in] Device serial number SN[0:8]. Only SN[0:1] and SN[8] are required though.
- const uint8_t ∗ stored_value
  [in] 32-byte slot value, config block, OTP block as specified by the Zone/KeyId parameters
- const uint8_t ∗ other_data
  [in] 32-byte value for shared nonce zone, 4-byte value if is_key_nomac is true, ignored and/or NULL otherwise
- struct atca_temp_key ∗ temp_key
  [inout] Current state of TempKey

14.11.1 Detailed Description

Input/output parameters for function atcah_gen_dig().

14.11.2 Field Documentation

14.11.2.1 counter

```
uint32_t counter
```

[in] counter for the GenDig command

14.11.2.2 is_key_nomac

```
bool is_key_nomac
```

[in] Set to true if the slot pointed to be key_id has the SotConfig.NoMac bit set

14.11.2.3 key_conf

```
uint16_t key_conf
```

[in] Key config for the GenDig command

14.11.2.4 key_id

```
uint16_t key_id
```

[in] KeyId/Param2 for the GenDig command

14.11.2.5 other_data

```
const uint8_t* other_data
```

[in] 32-byte value for shared nonce zone, 4-byte value if is_key_nomac is true, ignored and/or NULL otherwise

14.11.2.6 slot_conf

```
uint16_t slot_conf
```

[in] Slot config for the GenDig command

14.11.2.7 slot_locked

```
uint8_t slot_locked
```

[in] slot locked for the GenDig command

14.11.2.8 sn

```
const uint8_t* sn
```

[in] Device serial number SN[0:8]. Only SN[0:1] and SN[8] are required though.

14.11.2.9 stored_value

```
const uint8_t* stored_value
```

[in] 32-byte slot value, config block, OTP block as specified by the Zone/KeyId parameters

14.11.2.10 temp_key

```
struct atca_temp_key* temp_key
```

[inout] Current state of TempKey

14.11.2.11 zone

```
uint8_t zone
```

[in] Zone/Param1 for the GenDig command

14.12 atca_gen_key_in_out Struct Reference

Input/output parameters for calculating the PubKey digest put into TempKey by the GenKey command with the atcah_gen_key_msg() function.

```
#include <atca_host.h>
```

Data Fields

- uint8_t mode

  [in] GenKey Mode
- uint16_t key_id

  [in] GenKey KeyID
- const uint8_t ∗ public_key

  [in] Public key to be used in the PubKey digest. X and Y integers in big-endian format. 64 bytes for P256 curve.
- size_t public_key_size

  [in] Total number of bytes in the public key. 64 bytes for P256 curve.
- const uint8_t ∗ other_data

  [in] 3 bytes required when bit 4 of the mode is set. Can be NULL otherwise.
- const uint8_t ∗ sn

  [in] Device serial number SN[0:8] (9 bytes). Only SN[0:1] and SN[8] are required though.
- struct atca_temp_key ∗ temp_key

  [in,out] As input the current state of TempKey. As output, the resulting PubKEy digest.

14.12.1 Detailed Description

Input/output parameters for calculating the PubKey digest put into TempKey by the GenKey command with the atcah_gen_key_msg() function.

14.12.2 Field Documentation

14.12.2.1 key_id

```
uint16_t key_id
```

[in] GenKey KeyID

14.12.2.2 mode

```
uint8_t mode
```

[in] GenKey Mode

14.12.2.3 other_data

```
const uint8_t* other_data
```

[in] 3 bytes required when bit 4 of the mode is set. Can be NULL otherwise.

14.12.2.4 public_key

```
const uint8_t* public_key
```

[in] Public key to be used in the PubKey digest. X and Y integers in big-endian format. 64 bytes for P256 curve.

14.12.2.5 public_key_size

```
size_t public_key_size
```

[in] Total number of bytes in the public key. 64 bytes for P256 curve.

14.12.2.6 sn

```
const uint8_t* sn
```

[in] Device serial number SN[0:8] (9 bytes). Only SN[0:1] and SN[8] are required though.

14.12.2.7 temp_key

```
struct atca_temp_key* temp_key
```

[in,out] As input the current state of TempKey. As output, the resulting PubKEy digest.

14.13 atca_hmac_in_out Struct Reference

Input/output parameters for function atca_hmac().

```
#include <atca_host.h>
```

Data Fields

- uint8_t mode

  [in] Mode parameter used in HMAC command (Param1).
- uint16_t key_id

  [in] KeyID parameter used in HMAC command (Param2).
- const uint8_t ∗ key

  [in] Pointer to 32-byte key used to generate HMAC digest.
- const uint8_t ∗ otp

  [in] Pointer to 11-byte OTP, optionally included in HMAC digest, depending on mode.
- const uint8_t ∗ sn

  [in] Pointer to 9-byte SN, optionally included in HMAC digest, depending on mode.
- uint8_t ∗ response

  [out] Pointer to 32-byte SHA-256 HMAC digest.
- struct atca_temp_key ∗ temp_key

  [in,out] Pointer to TempKey structure.

14.13.1 Detailed Description

Input/output parameters for function atca_hmac().

14.14 atca_iface Struct Reference

atca_iface is the C object backing ATCAIface. See the atca_iface.h file for details on the ATCAIface methods

```
#include <atca_iface.h>
```

Data Fields

- ATCAIfaceType mType
- ATCAIfaceCfg ∗ mIfaceCFG
- ATCA_STATUS(∗ atinit)(void ∗hal, ATCAIfaceCfg ∗)
- ATCA_STATUS(∗ atpostinit)(ATCAIface hal)
- ATCA_STATUS(∗ atsend)(ATCAIface hal, uint8_t ∗txdata, int txlength)
- ATCA_STATUS(∗ atreceive)(ATCAIface hal, uint8_t ∗rxdata, uint16_t ∗rxlength)
- ATCA_STATUS(∗ atwake)(ATCAIface hal)
- ATCA_STATUS(∗ atidle)(ATCAIface hal)
- ATCA_STATUS(∗ atsleep)(ATCAIface hal)
- void ∗ hal_data

14.14.1 Detailed Description

atca_iface is the C object backing ATCAIface. See the atca_iface.h file for details on the ATCAIface methods

14.14.2 Field Documentation

14.14.2.1 atidle

```
ATCA_STATUS(* atidle) (ATCAIface hal)
```

14.14.2.2 atinit

```
ATCA_STATUS(* atinit) (void *hal, ATCAIfaceCfg *)
```

14.14.2.3 atpostinit

```
ATCA_STATUS(* atpostinit) (ATCAIface hal)
```

14.14.2.4 atreceive

```
ATCA_STATUS(* atreceive) (ATCAIface hal, uint8_t *rxdata, uint16_t *rxlength)
```

14.14.2.5 atsend

```
ATCA_STATUS(* atsend) (ATCAIface hal, uint8_t *txdata, int txlength)
```

14.14.2.6 atsleep

```
ATCA_STATUS(* atsleep) (ATCAIface hal)
```

14.14.2.7 atwake

```
ATCA_STATUS(* atwake) (ATCAIface hal)
```

14.14.2.8 hal_data

```
void* hal_data
```

14.14.2.9 mIfaceCFG

```
ATCAIfaceCfg* mIfaceCFG
```

14.14.2.10 mType

```
ATCAIfaceType mType
```

14.15 atca_include_data_in_out Struct Reference

Input / output parameters for function atca_include_data().

```
#include <atca_host.h>
```

Data Fields

- uint8_t * p_temp

  [out] pointer to output buffer
- const uint8_t * otp

  [in] pointer to one-time-programming data
- const uint8_t * sn

  [in] pointer to serial number data
- uint8_t mode

14.15.1 Detailed Description

Input / output parameters for function atca_include_data().

14.15.2 Field Documentation

14.15.2.1 mode

```
uint8_t mode
```

14.16 atca_io_decrypt_in_out Struct Reference

```
#include <atca_host.h>
```

Data Fields

- const uint8_t * io_key

  IO protection key (32 bytes).
- const uint8_t * out_nonce

  OutNonce returned from command (32 bytes).
- uint8_t * data

  As input, encrypted data. As output, decrypted data.
- size_t data_size

  Size of data in bytes (32 or 64).

14.16.1 Field Documentation

14.16.1.1 data

```
uint8_t* data
```

As input, encrypted data. As output, decrypted data.

14.16.1.2 data_size

```
size_t data_size
```

Size of data in bytes (32 or 64).

14.16.1.3 io_key

```
const uint8_t* io_key
```

IO protection key (32 bytes).

14.16.1.4 out_nonce

```
const uint8_t* out_nonce
```

OutNonce returned from command (32 bytes).

14.17 atca_jwt_t Struct Reference

Structure to hold metadata information about the jwt being built.

```
#include <atca_jwt.h>
```

Data Fields

- char * buf
- uint16_t buflen
- uint16_t cur

14.17.1 Detailed Description

Structure to hold metadata information about the jwt being built.

14.17.2 Field Documentation

14.17.2.1 buf

```
char* buf
```

14.17.2.2 buflen

```
uint16_t buflen
```

14.17.2.3 cur

```
uint16_t cur
```

14.18 atca_mac_in_out Struct Reference

Input/output parameters for function atca_mac().

```
#include <atca_host.h>
```

Data Fields

- uint8_t mode

  [in] Mode parameter used in MAC command (Param1).
- uint16_t key_id

  [in] KeyID parameter used in MAC command (Param2).
- const uint8_t ∗ challenge

  [in] Pointer to 32-byte Challenge data used in MAC command, depending on mode.
- const uint8_t ∗ key

  [in] Pointer to 32-byte key used to generate MAC digest.
- const uint8_t ∗ otp

  [in] Pointer to 11-byte OTP, optionally included in MAC digest, depending on mode.
- const uint8_t ∗ sn

  [in] Pointer to 9-byte SN, optionally included in MAC digest, depending on mode.
- uint8_t ∗ response

  [out] Pointer to 32-byte SHA-256 digest (MAC).
- struct atca_temp_key ∗ temp_key

  [in,out] Pointer to TempKey structure.

14.18.1 Detailed Description

Input/output parameters for function atca_mac().

14.19 atca_nonce_in_out Struct Reference

Input/output parameters for function atca_nonce().

```
#include <atca_host.h>
```

Data Fields

- uint8_t mode

  [in] Mode parameter used in Nonce command (Param1).
- uint16_t zero

  [in] Zero parameter used in Nonce command (Param2).
- const uint8_t ∗ num_in

  [in] Pointer to 20-byte NumIn data used in Nonce command.
- const uint8_t ∗ rand_out

  [in] Pointer to 32-byte RandOut data from Nonce command.
- struct atca_temp_key ∗ temp_key

  [in,out] Pointer to TempKey structure.

14.19.1 Detailed Description

Input/output parameters for function atca_nonce().

14.20 atca_secureboot_enc_in_out Struct Reference

```
#include <atca_host.h>
```

Data Fields

- const uint8_t * io_key

  IO protection key value (32 bytes)
- const struct atca_temp_key * temp_key

  Current value of TempKey.
- const uint8_t * digest

  Plaintext digest as input.
- uint8_t * hashed_key

  Calculated key is returned here (32 bytes)
- uint8_t * digest_enc

  Encrypted (ciphertext) digest is return here (32 bytes)

14.20.1 Field Documentation

14.20.1.1 digest

```
const uint8_t* digest
```

Plaintext digest as input.

14.20.1.2 digest_enc

```
uint8_t* digest_enc
```

Encrypted (ciphertext) digest is return here (32 bytes)

14.20.1.3 hashed_key

```
uint8_t* hashed_key
```

Calculated key is returned here (32 bytes)

14.20.1.4 io_key

```
const uint8_t* io_key
```

IO protection key value (32 bytes)

14.20.1.5 temp_key

```
const struct atca_temp_key* temp_key
```

Current value of TempKey.

14.21 atca_secureboot_mac_in_out Struct Reference

```
#include <atca_host.h>
```

Data Fields

- uint8_t mode

  SecureBoot mode (param1)
- uint16_t param2

  SecureBoot param2.
- uint16_t secure_boot_config

  SecureBootConfig value from configuration zone.
- const uint8_t ∗ hashed_key

  Hashed key. SHA256(IO Protection Key | TempKey)
- const uint8_t ∗ digest

  Digest (unencrypted)
- const uint8_t ∗ signature

  Signature (can be NULL if not required)
- uint8_t ∗ mac

  MAC is returned here.

14.21.1 Field Documentation

14.21.1.1 digest

```
const uint8_t* digest
```

Digest (unencrypted)

14.21.1.2 hashed_key

```
const uint8_t* hashed_key
```

Hashed key. SHA256(IO Protection Key | TempKey)

14.21.1.3 mac

```
uint8_t* mac
```

MAC is returned here.

14.21.1.4 mode

```
uint8_t mode
```

SecureBoot mode (param1)

14.21.1.5 param2

```
uint16_t param2
```

SecureBoot param2.

14.21.1.6 secure_boot_config

```
uint16_t secure_boot_config
```

SecureBootConfig value from configuration zone.

14.21.1.7 signature

```
const uint8_t* signature
```

Signature (can be NULL if not required)

14.22 atca_sha256_ctx Struct Reference

```
#include <atca_basic.h>
```

Data Fields

- uint32_t total_msg_size

  Total number of message bytes processed.
- uint32_t block_size

  Number of bytes in current block.
- uint8_t block [ATCA_SHA256_BLOCK_SIZE *2]

  Unprocessed message storage.

14.22.1 Field Documentation

14.22.1.1 block

```
uint8_t block[ATCA_SHA256_BLOCK_SIZE *2]
```

Unprocessed message storage.

14.22.1.2 block_size

```
uint32_t block_size
```

Number of bytes in current block.

14.22.1.3 total_msg_size

```
uint32_t total_msg_size
```

Total number of message bytes processed.

14.23 atca_sign_internal_in_out Struct Reference

Input/output parameters for calculating the message and digest used by the Sign(internal) command. Used with the atcah_sign_internal_msg() function.

```
#include <atca_host.h>
```

Data Fields

- uint8_t mode

    [in] Sign Mode
- uint16_t key_id

    [in] Sign KeyID
- uint16_t slot_config

    [in] SlotConfig[TempKeyFlags.keyId]
- uint16_t key_config

    [in] KeyConfig[TempKeyFlags.keyId]
- uint8_t use_flag

    [in] UseFlag[TempKeyFlags.keyId], 0x00 for slots 8 and above and for ATECC508A
- uint8_t update_count

    [in] UpdateCount[TempKeyFlags.keyId], 0x00 for slots 8 and above and for ATECC508A
- bool is_slot_locked

    [in] Is TempKeyFlags.keyId slot locked.
- bool for_invalidate

    [in] Set to true if this will be used for the Verify(Invalidate) command.
- const uint8_t ∗ sn

    [in] Device serial number SN[0:8] (9 bytes)
- const struct atca_temp_key ∗ temp_key

    [in] The current state of TempKey.
- uint8_t ∗ message

    [out] Full 55 byte message the Sign(internal) command will build. Can be NULL if not required.
- uint8_t ∗ verify_other_data

    [out] The 19 byte OtherData bytes to be used with the Verify(In/Validate) command. Can be NULL if not required.
- uint8_t ∗ digest

    [out] SHA256 digest of the full 55 byte message. Can be NULL if not required.

14.23.1 Detailed Description

Input/output parameters for calculating the message and digest used by the Sign(internal) command. Used with the atcah_sign_internal_msg() function.

14.23.2 Field Documentation

14.23.2.1 digest

```
uint8_t* digest
```

[out] SHA256 digest of the full 55 byte message. Can be NULL if not required.

14.23.2.2 for_invalidate

`bool for_invalidate`

[in] Set to true if this will be used for the Verify(Invalidate) command.

14.23.2.3 is_slot_locked

`bool is_slot_locked`

[in] Is TempKeyFlags.keyId slot locked.

14.23.2.4 key_config

`uint16_t key_config`

[in] KeyConfig[TempKeyFlags.keyId]

14.23.2.5 key_id

`uint16_t key_id`

[in] Sign KeyID

14.23.2.6 message

`uint8_t* message`

[out] Full 55 byte message the Sign(internal) command will build. Can be NULL if not required.

14.23.2.7 mode

`uint8_t mode`

[in] Sign Mode

14.23.2.8 slot_config

```
uint16_t slot_config
```

[in] SlotConfig[TempKeyFlags.keyId]

14.23.2.9 sn

```
const uint8_t* sn
```

[in] Device serial number SN[0:8] (9 bytes)

14.23.2.10 temp_key

```
const struct atca_temp_key* temp_key
```

[in] The current state of TempKey.

14.23.2.11 update_count

```
uint8_t update_count
```

[in] UpdateCount[TempKeyFlags.keyId], 0x00 for slots 8 and above and for ATECC508A

14.23.2.12 use_flag

```
uint8_t use_flag
```

[in] UseFlag[TempKeyFlags.keyId], 0x00 for slots 8 and above and for ATECC508A

14.23.2.13 verify_other_data

```
uint8_t* verify_other_data
```

[out] The 19 byte OtherData bytes to be used with the Verify(In/Validate) command. Can be NULL if not required.

14.24 atca_temp_key Struct Reference

Structure to hold TempKey fields.

`#include <atca_host.h>`

Data Fields

- uint8_t value [ATCA_KEY_SIZE *2]

  Value of TempKey (64 bytes for ATECC608A only)
- unsigned key_id: 4

  If TempKey was derived from a slot or transport key (GenDig or GenKey), that key ID is saved here.
- unsigned source_flag: 1

  Indicates id TempKey started from a random nonce (0) or not (1).
- unsigned gen_dig_data: 1

  TempKey was derived from the GenDig command.
- unsigned gen_key_data: 1

  TempKey was derived from the GenKey command (ATECC devices only).
- unsigned no_mac_flag: 1

  TempKey was derived from a key that has the NoMac bit set preventing the use of the MAC command. Known as CheckFlag in ATSHA devices).
- unsigned valid: 1

  TempKey is valid.
- uint8_t is_64

  TempKey has 64 bytes of valid data.

14.24.1 Detailed Description

Structure to hold TempKey fields.

14.24.2 Field Documentation

14.24.2.1 gen_dig_data

```
unsigned gen_dig_data
```

TempKey was derived from the GenDig command.

14.24.2.2 gen_key_data

```
unsigned gen_key_data
```

TempKey was derived from the GenKey command (ATECC devices only).

14.24.2.3 is_64

```
uint8_t is_64
```

TempKey has 64 bytes of valid data.

14.24.2.4 key_id

```
unsigned key_id
```

If TempKey was derived from a slot or transport key (GenDig or GenKey), that key ID is saved here.

14.24.2.5 no_mac_flag

```
unsigned no_mac_flag
```

TempKey was derived from a key that has the NoMac bit set preventing the use of the MAC command. Known as CheckFlag in ATSHA devices).

14.24.2.6 source_flag

```
unsigned source_flag
```

Indicates id TempKey started from a random nonce (0) or not (1).

14.24.2.7 valid

```
unsigned valid
```

TempKey is valid.

14.24.2.8 value

```
uint8_t value[ATCA_KEY_SIZE *2]
```

Value of TempKey (64 bytes for ATECC608A only)

14.25 atca_verify_in_out Struct Reference

Input/output parameters for function atcah_verify().

`#include <atca_host.h>`

Data Fields

- uint16_t curve_type

  [in] Curve type used in Verify command (Param2).
- const uint8_t ∗ signature

  [in] Pointer to ECDSA signature to be verified
- const uint8_t ∗ public_key

  [in] Pointer to the public key to be used for verification
- struct atca_temp_key ∗ temp_key

  [in,out] Pointer to TempKey structure.

14.25.1 Detailed Description

Input/output parameters for function atcah_verify().

14.26 atca_verify_mac Struct Reference

`#include <atca_host.h>`

Data Fields

- uint8_t mode

  Mode (Param1) parameter used in Verify command.
- uint16_t key_id

  KeyID (Param2) used in Verify command.
- const uint8_t ∗ signature

  Signature used in Verify command (64 bytes).
- const uint8_t ∗ other_data

  OtherData used in Verify command (19 bytes).
- const uint8_t ∗ msg_dig_buf

  Message digest buffer (64 bytes).
- const uint8_t ∗ io_key

  IO protection key value (32 bytes).
- const uint8_t ∗ sn

  Serial number (9 bytes).
- const atca_temp_key_t ∗ temp_key

  TempKey.
- uint8_t ∗ mac

  Calculated verification MAC is returned here (32 bytes).

14.26.1 Field Documentation

14.26.1.1 io_key

```
const uint8_t* io_key
```

IO protection key value (32 bytes).

14.26.1.2 key_id

```
uint16_t key_id
```

KeyID (Param2) used in Verify command.

14.26.1.3 mac

```
uint8_t* mac
```

Calculated verification MAC is returned here (32 bytes).

14.26.1.4 mode

```
uint8_t mode
```

Mode (Param1) parameter used in Verify command.

14.26.1.5 msg_dig_buf

```
const uint8_t* msg_dig_buf
```

Message digest buffer (64 bytes).

14.26.1.6 other_data

```
const uint8_t* other_data
```

OtherData used in Verify command (19 bytes).

14.26.1.7 signature

```
const uint8_t* signature
```

Signature used in Verify command (64 bytes).

14.26.1.8 sn

```
const uint8_t* sn
```

Serial number (9 bytes).

14.26.1.9 temp_key

```
const atca_temp_key_t* temp_key
```

TempKey.

14.27 atca_write_mac_in_out Struct Reference

Input/output parameters for function atcah_write_auth_mac() and atcah_privwrite_auth_mac().

```
#include <atca_host.h>
```

Data Fields

- uint8_t zone

  Zone/Param1 for the Write or PrivWrite command.
- uint16_t key_id

  KeyID/Param2 for the Write or PrivWrite command.
- const uint8_t * sn

  Device serial number SN[0:8]. Only SN[0:1] and SN[8] are required though.
- const uint8_t * input_data

  Data to be encrypted. 32 bytes for Write command, 36 bytes for PrivWrite command.
- uint8_t * encrypted_data

  Encrypted version of input_data will be returned here. 32 bytes for Write command, 36 bytes for PrivWrite command.
- uint8_t * auth_mac

  Write MAC will be returned here. 32 bytes.
- struct atca_temp_key * temp_key

  Current state of TempKey.

14.27.1 Detailed Description

Input/output parameters for function atcah_write_auth_mac() and atcah_privwrite_auth_mac().

14.27.2 Field Documentation

14.27.2.1 auth_mac

```
uint8_t* auth_mac
```

Write MAC will be returned here. 32 bytes.

14.27.2.2 encrypted_data

```
uint8_t* encrypted_data
```

Encrypted version of input_data will be returned here. 32 bytes for Write command, 36 bytes for PrivWrite command.

14.27.2.3 input_data

```
const uint8_t* input_data
```

Data to be encrypted. 32 bytes for Write command, 36 bytes for PrivWrite command.

14.27.2.4 key_id

```
uint16_t key_id
```

KeyID/Param2 for the Write or PrivWrite command.

14.27.2.5 sn

```
const uint8_t* sn
```

Device serial number SN[0:8]. Only SN[0:1] and SN[8] are required though.

14.27.2.6 temp_key

```
struct atca_temp_key* temp_key
```

Current state of TempKey.

14.27.2.7 zone

```
uint8_t zone
```

Zone/Param1 for the Write or PrivWrite command.

14.28 atcac_sha1_ctx Struct Reference

```
#include <atca_crypto_sw_sha1.h>
```

Data Fields

- uint32_t pad [32]
  Filler value to make sure the actual implementation has enough room to store its context. uint32_t is used to remove some alignment warnings.

14.28.1 Field Documentation

14.28.1.1 pad

```
uint32_t pad[32]
```

Filler value to make sure the actual implementation has enough room to store its context. uint32_t is used to remove some alignment warnings.

14.29 atcac_sha2_256_ctx Struct Reference

```
#include <atca_crypto_sw_sha2.h>
```

Data Fields

- uint32_t pad [48]
  Filler value to make sure the actual implementation has enough room to store its context. uint32_t is used to remove some alignment warnings.

14.29.1 Field Documentation

14.29.1.1 pad

```
uint32_t pad[48]
```

Filler value to make sure the actual implementation has enough room to store its context. uint32_t is used to remove some alignment warnings.

14.30 atcacdc Struct Reference

```
#include <hal_linux_kit_cdc.h>
```

Data Fields

- cdc_device_t kits [CDC_DEVICES_MAX]
- int8_t num_kits_found

14.30.1 Field Documentation

14.30.1.1 kits

```
cdc_device_t kits
```

14.30.1.2 num_kits_found

```
int8_t num_kits_found
```

14.31 atcacert_build_state_s Struct Reference

```
#include <atcacert_def.h>
```

Data Fields

- const atcacert_def_t ∗ cert_def

  Certificate definition for the certificate being rebuilt.
- uint8_t ∗ cert

  Buffer to contain the rebuilt certificate.
- size_t ∗ cert_size

  Current size of the certificate in bytes.
- size_t max_cert_size

  Max size of the cert buffer in bytes.
- uint8_t is_device_sn

  Indicates the structure contains the device SN.
- uint8_t device_sn [9]

  Storage for the device SN, when it's found.

14.31.1 Detailed Description

Tracks the state of a certificate as it's being rebuilt from device information.

14.31.2 Field Documentation

14.31.2.1 cert

```
uint8_t* cert
```

Buffer to contain the rebuilt certificate.

14.31.2.2 cert_def

```
const atcacert_def_t* cert_def
```

Certificate definition for the certificate being rebuilt.

14.31.2.3 cert_size

```
size_t* cert_size
```

Current size of the certificate in bytes.

14.31.2.4 device_sn

```
uint8_t device_sn[9]
```

Storage for the device SN, when it's found.

14.31.2.5 is_device_sn

```
uint8_t is_device_sn
```

Indicates the structure contains the device SN.

14.31.2.6 max_cert_size

```
size_t max_cert_size
```

Max size of the cert buffer in bytes.

14.32 atcacert_cert_element_s Struct Reference

```
#include <atcacert_def.h>
```

Data Fields

- char id [25]

  ID identifying this element.
- atcacert_device_loc_t device_loc

  Location in the device for the element.
- atcacert_cert_loc_t cert_loc

  Location in the certificate template for the element.
- atcacert_transform_t transforms [ATCA_MAX_TRANSFORMS]

  List of transforms from device to cert for this element.

14.32.1 Detailed Description

Defines a generic dynamic element for a certificate including the device and template locations.

14.32.2 Field Documentation

14.32.2.1 cert_loc

```
atcacert_cert_loc_t cert_loc
```

Location in the certificate template for the element.

14.32.2.2 device_loc

```
atcacert_device_loc_t device_loc
```

Location in the device for the element.

14.32.2.3 id

```
char id[25]
```

ID identifying this element.

14.32.2.4 transforms

```
atcacert_transform_t transforms[ATCA_MAX_TRANSFORMS]
```

List of transforms from device to cert for this element.

14.33 atcacert_cert_loc_s Struct Reference

```
#include <atcacert_def.h>
```

Data Fields

- uint16_t offset

    Byte offset in the certificate template.
- uint16_t count

    Byte count. Set to 0 if it doesn't exist.

14.33.1 Detailed Description

Defines a chunk of data in a certificate template.

14.33.2 Field Documentation

14.33.2.1 count

```
uint16_t count
```

Byte count. Set to 0 if it doesn't exist.

14.33.2.2 offset

```
uint16_t offset
```

Byte offset in the certificate template.

14.34 atcacert_def_s Struct Reference

```
#include <atcacert_def.h>
```

Data Fields

- atcacert_cert_type_t type
  Certificate type.
- uint8_t template_id
  ID for the this certificate definition (4-bit value).
- uint8_t chain_id
  ID for the certificate chain this definition is a part of (4-bit value).
- uint8_t private_key_slot
  If this is a device certificate template, this is the device slot for the device private key.
- atcacert_cert_sn_src_t sn_source
  Where the certificate serial number comes from (4-bit value).
- atcacert_device_loc_t cert_sn_dev_loc
  Only applies when sn_source is SNSRC_STORED or SNSRC_STORED_DYNAMIC. Describes where to get the certificate serial number on the device.
- atcacert_date_format_t issue_date_format
  Format of the issue date in the certificate.
- atcacert_date_format_t expire_date_format
  format of the expire date in the certificate.
- atcacert_cert_loc_t tbs_cert_loc
  Location in the certificate for the TBS (to be signed) portion.
- uint8_t expire_years
  Number of years the certificate is valid for (5-bit value). 0 means no expiration.
- atcacert_device_loc_t public_key_dev_loc
  Where on the device the public key can be found.

- atcacert_device_loc_t comp_cert_dev_loc

  Where on the device the compressed cert can be found.
- atcacert_cert_loc_t std_cert_elements [STDCERT_NUM_ELEMENTS]

  Where in the certificate template the standard cert elements are inserted.
- const atcacert_cert_element_t * cert_elements

  Additional certificate elements outside of the standard certificate contents.
- uint8_t cert_elements_count

  Number of additional certificate elements in cert_elements.
- const uint8_t * cert_template

  Pointer to the actual certificate template data.
- uint16_t cert_template_size

  Size of the certificate template in cert_template in bytes.
- const struct atcacert_def_s * ca_cert_def

  Certificate definition of the CA certificate.

14.34.1 Detailed Description

Defines a certificate and all the pieces to work with it.

If any of the standard certificate elements (std_cert_elements) are not a part of the certificate definition, set their count to 0 to indicate their absence.

14.34.2 Field Documentation

14.34.2.1 ca_cert_def

```
const struct atcacert_def_s* ca_cert_def
```

Certificate definition of the CA certificate.

14.34.2.2 cert_elements

```
const atcacert_cert_element_t* cert_elements
```

Additional certificate elements outside of the standard certificate contents.

14.34.2.3 cert_elements_count

```
uint8_t cert_elements_count
```

Number of additional certificate elements in cert_elements.

14.34.2.4 cert_sn_dev_loc

```
atcacert_device_loc_t cert_sn_dev_loc
```

Only applies when sn_source is SNSRC_STORED or SNSRC_STORED_DYNAMIC. Describes where to get the certificate serial number on the device.

14.34.2.5 cert_template

```
const uint8_t* cert_template
```

Pointer to the actual certificate template data.

14.34.2.6 cert_template_size

```
uint16_t cert_template_size
```

Size of the certificate template in cert_template in bytes.

14.34.2.7 chain_id

```
uint8_t chain_id
```

ID for the certificate chain this definition is a part of (4-bit value).

14.34.2.8 comp_cert_dev_loc

```
atcacert_device_loc_t comp_cert_dev_loc
```

Where on the device the compressed cert can be found.

14.34.2.9 expire_date_format

```
atcacert_date_format_t expire_date_format
```

format of the expire date in the certificate.

14.34.2.10 expire_years

`uint8_t expire_years`

Number of years the certificate is valid for (5-bit value). 0 means no expiration.

14.34.2.11 issue_date_format

`atcacert_date_format_t issue_date_format`

Format of the issue date in the certificate.

14.34.2.12 private_key_slot

`uint8_t private_key_slot`

If this is a device certificate template, this is the device slot for the device private key.

14.34.2.13 public_key_dev_loc

`atcacert_device_loc_t public_key_dev_loc`

Where on the device the public key can be found.

14.34.2.14 sn_source

`atcacert_cert_sn_src_t sn_source`

Where the certificate serial number comes from (4-bit value).

14.34.2.15 std_cert_elements

`atcacert_cert_loc_t std_cert_elements[STDCERT_NUM_ELEMENTS]`

Where in the certificate template the standard cert elements are inserted.

14.34.2.16 tbs_cert_loc

```
atcacert_cert_loc_t tbs_cert_loc
```

Location in the certificate for the TBS (to be signed) portion.

14.34.2.17 template_id

```
uint8_t template_id
```

ID for the this certificate definition (4-bit value).

14.34.2.18 type

```
atcacert_cert_type_t type
```

Certificate type.

14.35 atcacert_device_loc_s Struct Reference

```
#include <atcacert_def.h>
```

Data Fields

- atcacert_device_zone_t zone
  Zone in the device.
- uint8_t slot
  Slot within the data zone. Only applies if zone is DEVZONE_DATA.
- uint8_t is_genkey
  If true, use GenKey command to get the contents instead of Read.
- uint16_t offset
  Byte offset in the zone.
- uint16_t count
  Byte count.

14.35.1 Detailed Description

Defines a chunk of data in an ATECC device.

14.35.2 Field Documentation

14.35.2.1 count

```
uint16_t count
```

Byte count.

14.35.2.2 is_genkey

```
uint8_t is_genkey
```

If true, use GenKey command to get the contents instead of Read.

14.35.2.3 offset

```
uint16_t offset
```

Byte offset in the zone.

14.35.2.4 slot

```
uint8_t slot
```

Slot within the data zone. Only applies if zone is DEVZONE_DATA.

14.35.2.5 zone

```
atcacert_device_zone_t zone
```

Zone in the device.

14.36 atcacert_tm_utc_s Struct Reference

```
#include <atcacert_date.h>
```

Data Fields

- int tm_sec
- int tm_min
- int tm_hour
- int tm_mday
- int tm_mon
- int tm_year

14.36.1 Detailed Description

Holds a broken-down date in UTC. Mimics atcacert_tm_utc_t from time.h.

14.36.2 Field Documentation

14.36.2.1 tm_hour

```
int tm_hour
```

14.36.2.2 tm_mday

```
int tm_mday
```

14.36.2.3 tm_min

```
int tm_min
```

14.36.2.4 tm_mon

```
int tm_mon
```

14.36.2.5 tm_sec

```
int tm_sec
```

14.36.2.6 tm_year

```
int tm_year
```

14.37 ATCAHAL_t Struct Reference

an intermediary data structure to allow the HAL layer to point the standard API functions used by the upper layers to the HAL implementation for the interface. This isolates the upper layers and loosely couples the ATCAIface object from the physical implementation.

```
#include <atca_hal.h>
```

Data Fields

- ATCA_STATUS(* halinit)(void *hal, ATCAIfaceCfg *cfg)
- ATCA_STATUS(* halpostinit)(ATCAIface iface)
- ATCA_STATUS(* halsend)(ATCAIface iface, uint8_t *txdata, int txlength)
- ATCA_STATUS(* halreceive)(ATCAIface iface, uint8_t *rxdata, uint16_t *rxlength)
- ATCA_STATUS(* halwake)(ATCAIface iface)
- ATCA_STATUS(* halidle)(ATCAIface iface)
- ATCA_STATUS(* halsleep)(ATCAIface iface)
- ATCA_STATUS(* halrelease)(void *hal_data)
- void * hal_data

14.37.1 Detailed Description

an intermediary data structure to allow the HAL layer to point the standard API functions used by the upper layers to the HAL implementation for the interface. This isolates the upper layers and loosely couples the ATCAIface object from the physical implementation.

14.37.2 Field Documentation

14.37.2.1 hal_data

```
void* hal_data
```

14.37.2.2 halidle

```
ATCA_STATUS(* halidle) (ATCAIface iface)
```

14.37.2.3 halinit

```
ATCA_STATUS(* halinit) (void *hal, ATCAIfaceCfg *cfg)
```

14.37.2.4 halpostinit

```
ATCA_STATUS(* halpostinit) (ATCAIface iface)
```

14.37.2.5 halreceive

```
ATCA_STATUS(* halreceive) (ATCAIface iface, uint8_t *rxdata, uint16_t *rxlength)
```

14.37.2.6 halrelease

```
ATCA_STATUS(* halrelease) (void *hal_data)
```

14.37.2.7 halsend

```
ATCA_STATUS(* halsend) (ATCAIface iface, uint8_t *txdata, int txlength)
```

14.37.2.8 halsleep

```
ATCA_STATUS(* halsleep) (ATCAIface iface)
```

14.37.2.9 halwake

```
ATCA_STATUS(* halwake) (ATCAIface iface)
```

14.38 atcahid Struct Reference

```
#include <hal_all_platforms_kit_hidapi.h>
```

Data Fields

- hid_device * kits [HID_DEVICES_MAX]
- int8_t num_kits_found
- hid_device_t kits [HID_DEVICES_MAX]

14.38.1 Field Documentation

14.38.1.1 kits [1/2]

```
hid_device_t kits
```

14.38.1.2 kits [2/2]

```
hid_device_t kits[HID_DEVICES_MAX]
```

14.38.1.3 num_kits_found

```
int8_t num_kits_found
```

14.39 atcaI2Cmaster Struct Reference

this is the hal_data for ATCA HAL created using ASF

```
#include <hal_at90usb1287_i2c_asf.h>
```

Data Fields

- volatile void * i2c_master_instance
- int ref_ct
- int bus_index

  for conveniences during interface release phase
- int id
- uint32_t pin_sda
- uint32_t pin_scl
- struct i2c_m_sync_desc i2c_master_instance
- uint32_t sercom_core_freq
- char i2c_file [16]
- I2C_MODULE id
- uint32_t twi_id
- Twi * twi_master_instance
- I2C * i2c_sercom
- struct i2c_master_module i2c_master_instance
- Sercom * i2c_sercom
- Flexcom * twi_flexcom
- uint32_t twi_flexcom_id
- uint8_t twi_id
- Twihs * twi_module
- avr32_twi_t * twi_master_instance
- twi_master_t i2c_master_instance

14.39.1 Detailed Description

this is the hal_data for ATCA HAL created using ASF

this is the hal_data for ATCA HAL for ASF SERCOM

this is the hal_data for ATCA HAL for ASF

this is the hal_data for ATCA HAL

this is the hal_data for ATCA HAL for Atmel START SERCOM

This is the hal_data for ATCA HAL.

14.39.2 Field Documentation

14.39.2.1 bus_index

```
int bus_index
```

for conveniences during interface release phase

14.39.2.2 i2c_file

```
char i2c_file[16]
```

14.39.2.3 i2c_master_instance [1/4]

```
struct i2c_master_module i2c_master_instance
```

14.39.2.4 i2c_master_instance [2/4]

```
twi_master_t i2c_master_instance
```

14.39.2.5 i2c_master_instance [3/4]

```
struct i2c_master_module i2c_master_instance
```

14.39.2.6 i2c_master_instance [4/4]

```
struct i2c_m_sync_desc i2c_master_instance
```

14.39.2.7 i2c_sercom [1/2]

```
I2C* i2c_sercom
```

14.39.2.8 i2c_sercom [2/2]

```
Sercom* i2c_sercom
```

14.39.2.9 id [1/2]

```
int id
```

14.39.2.10 id [2/2]

```
I2C_MODULE id
```

14.39.2.11 pin_scl

```
uint32_t pin_scl
```

14.39.2.12 pin_sda

```
uint32_t pin_sda
```

14.39.2.13 ref_ct

```
int ref_ct
```

14.39.2.14 sercom_core_freq

```
uint32_t sercom_core_freq
```

14.39.2.15 twi_flexcom

```
Flexcom* twi_flexcom
```

14.39.2.16 twi_flexcom_id

```
uint32_t twi_flexcom_id
```

14.39.2.17 twi_id [1/2]

```
uint8_t twi_id
```

14.39.2.18 twi_id [2/2]

```
uint8_t twi_id
```

14.39.2.19 twi_master_instance [1/2]

```
avr32_twi_t* twi_master_instance
```

14.39.2.20 twi_master_instance [2/2]

```
Twi * twi_master_instance
```

14.39.2.21 twi_module

```
Twihs* twi_module
```

14.40 ATCAIfaceCfg Struct Reference

```
#include <atca_iface.h>
```

Data Fields

- ATCAIfaceType iface_type
- ATCADeviceType devtype
- union {
  struct **ATCAI2C** {
  uint8_t slave_address
  uint8_t bus
  uint32_t baud
  } atcai2c
  struct **ATCASWI** {
  uint8_t bus
  } atcaswi
  struct **ATCAUART** {
  int port
  uint32_t baud
  uint8_t wordsize
  uint8_t parity
  uint8_t stopbits
  } atcauart
  struct **ATCAHID** {
  int idx
  ATCAKitType dev_interface
  uint8_t dev_identity
  uint32_t vid
  uint32_t pid
  uint32_t packetsize
  uint8_t guid [16]
  } atcahid
  struct **ATCACUSTOM** {
  ATCA_STATUS(∗ halinit)(void ∗hal, void ∗cfg)
  ATCA_STATUS(∗ halpostinit)(void ∗iface)
  ATCA_STATUS(∗ halsend)(void ∗iface, uint8_t ∗txdata, int txlength)
  ATCA_STATUS(∗ halreceive)(void ∗iface, uint8_t ∗rxdata, uint16_t ∗rxlength)
  ATCA_STATUS(∗ halwake)(void ∗iface)
  ATCA_STATUS(∗ halidle)(void ∗iface)
  ATCA_STATUS(∗ halsleep)(void ∗iface)
  ATCA_STATUS(∗ halrelease)(void ∗hal_data)
  } atcacustom
  };
- uint16_t wake_delay
- int rx_retries
- void ∗ cfg_data

14.40.1 Field Documentation

14.40.1.1 "@1

```
union { ... }
```

14.40.1.2 atcacustom

```
struct { ... } ::ATCACUSTOM atcacustom
```

14.40.1.3 atcahid

```
struct { ... } ::ATCAHID atcahid
```

14.40.1.4 atcai2c

```
struct { ... } ::ATCAI2C atcai2c
```

14.40.1.5 atcaswi

```
struct { ... } ::ATCASWI atcaswi
```

14.40.1.6 atcauart

```
struct { ... } ::ATCAUART atcauart
```

14.40.1.7 baud

```
uint32_t baud
```

14.40.1.8 bus

```
uint8_t bus
```

14.40.1.9 cfg_data

```
void* cfg_data
```

14.40.1.10 dev_identity

```
uint8_t dev_identity
```

14.40.1.11 dev_interface

```
ATCAKitType dev_interface
```

14.40.1.12 devtype

```
ATCADeviceType devtype
```

14.40.1.13 guid

```
uint8_t guid[16]
```

14.40.1.14 halidle

```
ATCA_STATUS(* halidle) (void *iface)
```

14.40.1.15 halinit

```
ATCA_STATUS(* halinit) (void *hal, void *cfg)
```

14.40.1.16 halpostinit

```
ATCA_STATUS(* halpostinit) (void *iface)
```

14.40.1.17 halreceive

```
ATCA_STATUS(* halreceive) (void *iface, uint8_t *rxdata, uint16_t *rxlength)
```

14.40.1.18 halrelease

```
ATCA_STATUS(* halrelease) (void *hal_data)
```

14.40.1.19 halsend

```
ATCA_STATUS(* halsend) (void *iface, uint8_t *txdata, int txlength)
```

14.40.1.20 halsleep

```
ATCA_STATUS(* halsleep) (void *iface)
```

14.40.1.21 halwake

```
ATCA_STATUS(* halwake) (void *iface)
```

14.40.1.22 idx

```
int idx
```

14.40.1.23 iface_type

```
ATCAIfaceType iface_type
```

14.40.1.24 packetsize

```
uint32_t packetsize
```

14.40.1.25 parity

```
uint8_t parity
```

14.40.1.26 pid

```
uint32_t pid
```

14.40.1.27 port

```
int port
```

14.40.1.28 rx_retries

```
int rx_retries
```

14.40.1.29 slave_address

```
uint8_t slave_address
```

14.40.1.30 stopbits

```
uint8_t stopbits
```

14.40.1.31 vid

```
uint32_t vid
```

14.40.1.32 wake_delay

```
uint16_t wake_delay
```

14.40.1.33 wordsize

```
uint8_t wordsize
```

14.41 ATCAPacket Struct Reference

```
#include <atca_command.h>
```

Data Fields

- uint8_t _reserved
- uint8_t txsize
- uint8_t opcode
- uint8_t param1
- uint16_t param2
- uint8_t data [192]
- uint8_t execTime

14.41.1 Field Documentation

14.41.1.1 _reserved

```
uint8_t _reserved
```

14.41.1.2 data

```
uint8_t data[192]
```

14.41.1.3 execTime

```
uint8_t execTime
```

14.41.1.4 opcode

```
uint8_t opcode
```

14.41.1.5 param1

```
uint8_t param1
```

14.41.1.6 param2

```
uint16_t param2
```

14.41.1.7 txsize

```
uint8_t txsize
```

14.42 atcaSWImaster Struct Reference

This is the hal_data for ATCA HAL.

```
#include <hal_swi_bitbang.h>
```

Data Fields

- uint8_t pin_sda
- int ref_ct
- int bus_index

  for conveniences during interface release phase
- usart_if usart_instance
- struct usart_module usart_instance
- struct usart_sync_descriptor USART_SWI
- uint32_t sercom_core_freq

14.42.1 Detailed Description

This is the hal_data for ATCA HAL.

this is the hal_data for ATCA HAL for ASF SERCOM

this is the hal_data for ATCA HAL for SWI UART

14.42.2 Field Documentation

14.42.2.1 bus_index

```
int bus_index
```

for conveniences during interface release phase

14.42.2.2 pin_sda

```
uint8_t pin_sda
```

14.42.2.3 ref_ct

```
int ref_ct
```

14.42.2.4 sercom_core_freq

```
uint32_t sercom_core_freq
```

14.42.2.5 usart_instance [1/2]

```
usart_if usart_instance
```

14.42.2.6 usart_instance [2/2]

```
struct usart_module usart_instance
```

14.42.2.7 USART_SWI

```
struct usart_sync_descriptor USART_SWI
```

14.43 cdc_device Struct Reference

```
#include <hal_linux_kit_cdc.h>
```

Data Fields

- HANDLE read_handle
- HANDLE write_handle

    The kit USB read file handle.

14.43.1 Field Documentation

14.43.1.1 read_handle

```
HANDLE read_handle
```

14.43.1.2 write_handle

```
HANDLE write_handle
```

The kit USB read file handle.

14.44 CL_HashContext Struct Reference

```
#include <sha1_routines.h>
```

Data Fields

- U32 h [20/4]
- U32 buf [64/4]
- U32 byteCount
- U32 byteCountHi

14.44.1 Field Documentation

14.44.1.1 buf

```
U32 buf[64/4]
```

14.44.1.2 byteCount

```
U32 byteCount
```

14.44.1.3 byteCountHi

```
U32 byteCountHi
```

14.44.1.4 h

```
U32 h[20/4]
```

14.45 DRV_I2C_Object Struct Reference

```
#include <hal_pic32mz2048efm_i2c.h>
```

Data Fields

- volatile uintptr_t i2cDriverInstance
- uint32_t i2cDriverInstanceIndex
- void * i2cDriverInit

14.45.1 Field Documentation

14.45.1.1 i2cDriverInit

```
void* i2cDriverInit
```

14.45.1.2 i2cDriverInstance

```
volatile uintptr_t i2cDriverInstance
```

14.45.1.3 i2cDriverInstanceIndex

```
uint32_t i2cDriverInstanceIndex
```

14.46 hid_device Struct Reference

```
#include <hal_linux_kit_hid.h>
```

Data Fields

- FILE ∗ read_handle
- FILE ∗ write_handle

  The kit USB read file handle.
- HANDLE read_handle
- HANDLE write_handle

  The kit USB read file handle.

14.46.1 Field Documentation

14.46.1.1 read_handle [1/2]

```
FILE* read_handle
```

14.46.1.2 read_handle [2/2]

```
HANDLE read_handle
```

14.46.1.3 write_handle [1/2]

```
FILE* write_handle
```

The kit USB read file handle.

14.46.1.4 write_handle [2/2]

```
HANDLE write_handle
```

The kit USB read file handle.

14.47 hw_sha256_ctx Struct Reference

Data Fields

- uint32_t total_msg_size

  Total number of message bytes processed.
- uint32_t block_size

  Number of bytes in current block.
- uint8_t block [ATCA_SHA256_BLOCK_SIZE ∗2]

  Unprocessed message storage.

14.47.1 Field Documentation

14.47.1.1 block

```
uint8_t block[ATCA_SHA256_BLOCK_SIZE *2]
```

Unprocessed message storage.

14.47.1.2 block_size

```
uint32_t block_size
```

Number of bytes in current block.

14.47.1.3 total_msg_size

```
uint32_t total_msg_size
```

Total number of message bytes processed.

14.48 I2CBuses Struct Reference

```
#include <i2c_bitbang_samd21.h>
```

Data Fields

- uint8_t pin_sda [MAX_I2C_BUSES]
- uint8_t pin_scl [MAX_I2C_BUSES]

14.48.1 Field Documentation

14.48.1.1 pin_scl

```
uint8_t pin_scl[MAX_I2C_BUSES]
```

14.48.1.2 pin_sda

```
uint8_t pin_sda[MAX_I2C_BUSES]
```

14.49 memory_parameters Struct Reference

```
#include <secure_boot_memory.h>
```

Data Fields

- uint32_t start_address
- uint32_t memory_size
- uint32_t version_info
- uint8_t reserved [52]
- uint8_t signature [ATCA_SIG_SIZE]

14.49.1 Field Documentation

14.49.1.1 memory_size

```
uint32_t memory_size
```

14.49.1.2 reserved

```
uint8_t reserved[52]
```

14.49.1.3 signature

```
uint8_t signature[ATCA_SIG_SIZE]
```

14.49.1.4 start_address

```
uint32_t start_address
```

14.49.1.5 version_info

```
uint32_t version_info
```

14.50 secure_boot_config_bits Struct Reference

```
#include <secure_boot.h>
```

Data Fields

- uint16_t secure_boot_mode: 2
- uint16_t secure_boot_reserved1: 1
- uint16_t secure_boot_persistent_enable: 1
- uint16_t secure_boot_rand_nonce: 1
- uint16_t secure_boot_reserved2: 3
- uint16_t secure_boot_sig_dig: 4
- uint16_t secure_boot_pub_key: 4

14.50.1 Field Documentation

14.50.1.1 secure_boot_mode

```
uint16_t secure_boot_mode
```

14.50.1.2 secure_boot_persistent_enable

```
uint16_t secure_boot_persistent_enable
```

14.50.1.3 secure_boot_pub_key

```
uint16_t secure_boot_pub_key
```

14.50.1.4 secure_boot_rand_nonce

```
uint16_t secure_boot_rand_nonce
```

14.50.1.5 secure_boot_reserved1

```
uint16_t secure_boot_reserved1
```

14.50.1.6 secure_boot_reserved2

```
uint16_t secure_boot_reserved2
```

14.50.1.7 secure_boot_sig_dig

```
uint16_t secure_boot_sig_dig
```

14.51 secure_boot_parameters Struct Reference

```
#include <secure_boot.h>
```

Data Fields

- memory_parameters memory_params
- atcac_sha2_256_ctx s_sha_context
- uint8_t app_digest [ATCA_SHA_DIGEST_SIZE]

14.51.1 Field Documentation

14.51.1.1 app_digest

```
uint8_t app_digest[ATCA_SHA_DIGEST_SIZE]
```

14.51.1.2 memory_params

```
memory_parameters memory_params
```

14.51.1.3 s_sha_context

```
atcac_sha2_256_ctx s_sha_context
```

14.52 sw_sha256_ctx Struct Reference

```
#include <sha2_routines.h>
```

Data Fields

- uint32_t total_msg_size

  Total number of message bytes processed.
- uint32_t block_size

  Number of bytes in current block.
- uint8_t block [SHA256_BLOCK_SIZE ∗2]

  Unprocessed message storage.
- uint32_t hash [8]

  Hash state.

14.52.1 Field Documentation

14.52.1.1 block

```
uint8_t block[SHA256_BLOCK_SIZE *2]
```

Unprocessed message storage.

14.52.1.2 block_size

```
uint32_t block_size
```

Number of bytes in current block.

14.52.1.3 hash

```
uint32_t hash[8]
```

Hash state.

14.52.1.4 total_msg_size

```
uint32_t total_msg_size
```

Total number of message bytes processed.

14.53 SWIBuses Struct Reference

```
#include <swi_bitbang_samd21.h>
```

Data Fields

- uint32_t pin_sda [MAX_SWI_BUSES]

14.53.1 Field Documentation

14.53.1.1 pin_sda

```
uint32_t pin_sda[MAX_SWI_BUSES]
```

Chapter 15

File Documentation

15.1 atca_basic.c File Reference

CryptoAuthLib Basic API methods. These methods provide a simpler way to access the core crypto methods.

```
#include "atca_basic.h"
#include "host/atca_host.h"
```

Macros

- #define MAX_BUSES 4

Functions

- ATCA_STATUS atcab_version (char ∗ver_str)

  basic API methods are all prefixed with atcab_ (CryptoAuthLib Basic) the fundamental premise of the basic API is it is based on a single interface instance and that instance is global, so all basic API commands assume that one global device is the one to operate on.
- ATCA_STATUS atcab_init (ATCAIfaceCfg ∗cfg)

  Creates a global ATCADevice object used by Basic API.
- ATCA_STATUS atcab_init_device (ATCADevice ca_device)

  Initialize the global ATCADevice object to point to one of your choosing for use with all the atcab_ basic API.
- ATCA_STATUS atcab_release (void)

  release (free) the global ATCADevice instance. This must be called in order to release or free up the interface.
- ATCADevice atcab_get_device (void)

  Get the global device object.
- ATCADeviceType atcab_get_device_type (void)

  Get the current device type.
- ATCA_STATUS atcab_wakeup (void)

  wakeup the CryptoAuth device
- ATCA_STATUS atcab_idle (void)

  idle the CryptoAuth device
- ATCA_STATUS atcab_sleep (void)

  invoke sleep on the CryptoAuth device

- ATCA_STATUS atcab_cfg_discover (ATCAIfaceCfg cfg_array[ ], int max_ifaces)

    auto discovery of crypto auth devices
- ATCA_STATUS _atcab_exit (void)

    common cleanup code which idles the device after any operation
- ATCA_STATUS atcab_get_addr (uint8_t zone, uint16_t slot, uint8_t block, uint8_t offset, uint16_t ∗addr)

    Compute the address given the zone, slot, block, and offset.
- ATCA_STATUS atcab_get_zone_size (uint8_t zone, uint16_t slot, size_t ∗size)

    Gets the size of the specified zone in bytes.

Variables

- const char atca_version [ ] = { "20190517" }
- ATCADevice _gDevice = NULL

15.1.1 Detailed Description

CryptoAuthLib Basic API methods. These methods provide a simpler way to access the core crypto methods.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.1.2 Macro Definition Documentation

15.1.2.1 MAX_BUSES

```
#define MAX_BUSES 4
```

15.1.3 Variable Documentation

15.1.3.1 atca_version

```
const char atca_version[] = { "20190517" }
```

15.2 atca_basic.h File Reference

CryptoAuthLib Basic API methods - a simple crypto authentication API. These methods manage a global ATCA←Device object behind the scenes. They also manage the wake/idle state transitions so callers don't need to.

```
#include "cryptoauthlib.h"
#include "crypto/atca_crypto_sw_sha2.h"
```

Data Structures

- struct atca_aes_cbc_ctx
- struct atca_aes_cmac_ctx
- struct atca_aes_ctr_ctx
- struct atca_sha256_ctx

Macros

- #define BLOCK_NUMBER(a) (a / 32)
- #define WORD_OFFSET(a) ((a % 32) / 4)
- #define ATCA_AES_GCM_IV_STD_LENGTH 12

Typedefs

- typedef struct atca_aes_cbc_ctx atca_aes_cbc_ctx_t
- typedef struct atca_aes_cmac_ctx atca_aes_cmac_ctx_t
- typedef struct atca_aes_ctr_ctx atca_aes_ctr_ctx_t
- typedef struct atca_sha256_ctx atca_sha256_ctx_t
- typedef atca_sha256_ctx_t atca_hmac_sha256_ctx_t

Functions

- ATCA_STATUS atcab_version (char ∗ver_str)

  basic API methods are all prefixed with atcab_ (CryptoAuthLib Basic) the fundamental premise of the basic API is it is based on a single interface instance and that instance is global, so all basic API commands assume that one global device is the one to operate on.
- ATCA_STATUS atcab_init (ATCAIfaceCfg ∗cfg)

  Creates a global ATCADevice object used by Basic API.
- ATCA_STATUS atcab_init_device (ATCADevice ca_device)

  Initialize the global ATCADevice object to point to one of your choosing for use with all the atcab_ basic API.
- ATCA_STATUS atcab_release (void)

  release (free) the global ATCADevice instance. This must be called in order to release or free up the interface.
- ATCADevice atcab_get_device (void)

  Get the global device object.
- ATCADeviceType atcab_get_device_type (void)

  Get the current device type.
- ATCA_STATUS _atcab_exit (void)

  common cleanup code which idles the device after any operation
- ATCA_STATUS atcab_wakeup (void)

  wakeup the CryptoAuth device
- ATCA_STATUS atcab_idle (void)

  idle the CryptoAuth device
- ATCA_STATUS atcab_sleep (void)

  invoke sleep on the CryptoAuth device
- ATCA_STATUS atcab_cfg_discover (ATCAIfaceCfg cfg_array[ ], int max)

  auto discovery of crypto auth devices
- ATCA_STATUS atcab_get_addr (uint8_t zone, uint16_t slot, uint8_t block, uint8_t offset, uint16_t ∗addr)

  Compute the address given the zone, slot, block, and offset.
- ATCA_STATUS atcab_get_zone_size (uint8_t zone, uint16_t slot, size_t ∗size)

Gets the size of the specified zone in bytes.

- ATCA_STATUS atcab_aes (uint8_t mode, uint16_t key_id, const uint8_t ∗aes_in, uint8_t ∗aes_out)

  Compute the AES-128 encrypt, decrypt, or GFM calculation.
- ATCA_STATUS atcab_aes_encrypt (uint16_t key_id, uint8_t key_block, const uint8_t ∗plaintext, uint8_t ∗ciphertext)

  Perform an AES-128 encrypt operation with a key in the device.
- ATCA_STATUS atcab_aes_decrypt (uint16_t key_id, uint8_t key_block, const uint8_t ∗ciphertext, uint8_t ∗plaintext)

  Perform an AES-128 decrypt operation with a key in the device.
- ATCA_STATUS atcab_aes_gfm (const uint8_t ∗h, const uint8_t ∗input, uint8_t ∗output)

  Perform a Galois Field Multiply (GFM) operation.
- ATCA_STATUS atcab_aes_cbc_init (atca_aes_cbc_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, const uint8_t ∗iv)

  Initialize context for AES CBC operation.
- ATCA_STATUS atcab_aes_cbc_encrypt_block (atca_aes_cbc_ctx_t ∗ctx, const uint8_t ∗plaintext, uint8_t ∗ciphertext)

  Encrypt a block of data using CBC mode and a key within the ATECC608A. atcab_aes_cbc_init() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_cbc_decrypt_block (atca_aes_cbc_ctx_t ∗ctx, const uint8_t ∗ciphertext, uint8_t ∗plaintext)

  Decrypt a block of data using CBC mode and a key within the ATECC608A. atcab_aes_cbc_init() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_cmac_init (atca_aes_cmac_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block)

  Initialize a CMAC calculation using an AES-128 key in the ATECC608A.
- ATCA_STATUS atcab_aes_cmac_update (atca_aes_cmac_ctx_t ∗ctx, const uint8_t ∗data, uint32_t data_size)

  Add data to an initialized CMAC calculation.
- ATCA_STATUS atcab_aes_cmac_finish (atca_aes_cmac_ctx_t ∗ctx, uint8_t ∗cmac, uint32_t cmac_size)

  Finish a CMAC operation returning the CMAC value.
- ATCA_STATUS atcab_aes_ctr_init (atca_aes_ctr_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, uint8_t counter_size, const uint8_t ∗iv)

  Initialize context for AES CTR operation with an existing IV, which is common when start a decrypt operation.
- ATCA_STATUS atcab_aes_ctr_init_rand (atca_aes_ctr_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, uint8_t counter_size, uint8_t ∗iv)

  Initialize context for AES CTR operation with a random nonce and counter set to 0 as the IV, which is common when starting an encrypt operation.
- ATCA_STATUS atcab_aes_ctr_block (atca_aes_ctr_ctx_t ∗ctx, const uint8_t ∗input, uint8_t ∗output)

  Process a block of data using CTR mode and a key within the ATECC608A device. atcab_aes_ctr_init() or atcab_aes_ctr_init_rand() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_ctr_encrypt_block (atca_aes_ctr_ctx_t ∗ctx, const uint8_t ∗plaintext, uint8_t ∗ciphertext)

  Encrypt a block of data using CTR mode and a key within the ATECC608A device. atcab_aes_ctr_init() or atcab_aes_ctr_init_rand() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_ctr_decrypt_block (atca_aes_ctr_ctx_t ∗ctx, const uint8_t ∗ciphertext, uint8_t ∗plaintext)

  Decrypt a block of data using CTR mode and a key within the ATECC608A device. atcab_aes_ctr_init() or atcab_aes_ctr_init_rand() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_ctr_increment (atca_aes_ctr_ctx_t ∗ctx)

  Increments AES CTR counter value.
- ATCA_STATUS atcab_checkmac (uint8_t mode, uint16_t key_id, const uint8_t ∗challenge, const uint8_t ∗response, const uint8_t ∗other_data)

  Compares a MAC response with input values.
- ATCA_STATUS atcab_counter (uint8_t mode, uint16_t counter_id, uint32_t ∗counter_value)

Compute the Counter functions.
- ATCA_STATUS atcab_counter_increment (uint16_t counter_id, uint32_t ∗counter_value)

    Increments one of the device's monotonic counters.
- ATCA_STATUS atcab_counter_read (uint16_t counter_id, uint32_t ∗counter_value)

    Read one of the device's monotonic counters.
- ATCA_STATUS atcab_derivekey (uint8_t mode, uint16_t key_id, const uint8_t ∗mac)

    Executes the DeviveKey command for deriving a new key from a nonce (TempKey) and an existing key.
- ATCA_STATUS atcab_ecdh_base (uint8_t mode, uint16_t key_id, const uint8_t ∗public_key, uint8_t ∗pms, uint8_t ∗out_nonce)

    Base function for generating premaster secret key using ECDH.
- ATCA_STATUS atcab_ecdh (uint16_t key_id, const uint8_t ∗public_key, uint8_t ∗pms)

    ECDH command with a private key in a slot and the premaster secret is returned in the clear.
- ATCA_STATUS atcab_ecdh_enc (uint16_t key_id, const uint8_t ∗public_key, uint8_t ∗pms, const uint8_t ∗read_key, uint16_t read_key_id)

    ECDH command with a private key in a slot and the premaster secret is read from the next slot.
- ATCA_STATUS atcab_ecdh_ioenc (uint16_t key_id, const uint8_t ∗public_key, uint8_t ∗pms, const uint8_t ∗io_key)

    ECDH command with a private key in a slot and the premaster secret is returned encrypted using the IO protection key.
- ATCA_STATUS atcab_ecdh_tempkey (const uint8_t ∗public_key, uint8_t ∗pms)

    ECDH command with a private key in TempKey and the premaster secret is returned in the clear.
- ATCA_STATUS atcab_ecdh_tempkey_ioenc (const uint8_t ∗public_key, uint8_t ∗pms, const uint8_t ∗io_key)

    ECDH command with a private key in TempKey and the premaster secret is returned encrypted using the IO protection key.
- ATCA_STATUS atcab_gendig (uint8_t zone, uint16_t key_id, const uint8_t ∗other_data, uint8_t other_data←_size)

    Issues a GenDig command, which performs a SHA256 hash on the source data indicated by zone with the contents of TempKey. See the CryptoAuth datasheet for your chip to see what the values of zone correspond to.
- ATCA_STATUS atcab_genkey_base (uint8_t mode, uint16_t key_id, const uint8_t ∗other_data, uint8_←t ∗public_key)

    Issues GenKey command, which can generate a private key, compute a public key, nd/or compute a digest of a public key.
- ATCA_STATUS atcab_genkey (uint16_t key_id, uint8_t ∗public_key)

    Issues GenKey command, which generates a new random private key in slot and returns the public key.
- ATCA_STATUS atcab_get_pubkey (uint16_t key_id, uint8_t ∗public_key)

    Uses GenKey command to calculate the public key from an existing private key in a slot.
- ATCA_STATUS atcab_hmac (uint8_t mode, uint16_t key_id, uint8_t ∗digest)

    Issues a HMAC command, which computes an HMAC/SHA-256 digest of a key stored in the device, a challenge, and other information on the device.
- ATCA_STATUS atcab_info_base (uint8_t mode, uint16_t param2, uint8_t ∗out_data)

    Issues an Info command, which return internal device information and can control GPIO and the persistent latch.
- ATCA_STATUS atcab_info (uint8_t ∗revision)

    Use the Info command to get the device revision (DevRev).
- ATCA_STATUS atcab_info_set_latch (bool state)

    Use the Info command to set the persistent latch state for an ATECC608A device.
- ATCA_STATUS atcab_info_get_latch (bool ∗state)

    Use the Info command to get the persistent latch current state for an ATECC608A device.
- ATCA_STATUS atcab_kdf (uint8_t mode, uint16_t key_id, const uint32_t details, const uint8_t ∗message, uint8_t ∗out_data, uint8_t ∗out_nonce)

    Executes the KDF command, which derives a new key in PRF, AES, or HKDF modes.
- ATCA_STATUS atcab_lock (uint8_t mode, uint16_t summary_crc)

The Lock command prevents future modifications of the Configuration and/or Data and OTP zones. If the device is so configured, then this command can be used to lock individual data slots. This command fails if the designated area is already locked.

- ATCA_STATUS atcab_lock_config_zone (void)

  Unconditionally (no CRC required) lock the config zone.
- ATCA_STATUS atcab_lock_config_zone_crc (uint16_t summary_crc)

  Lock the config zone with summary CRC.
- ATCA_STATUS atcab_lock_data_zone (void)

  Unconditionally (no CRC required) lock the data zone (slots and OTP).
- ATCA_STATUS atcab_lock_data_zone_crc (uint16_t summary_crc)

  Lock the data zone (slots and OTP) with summary CRC.
- ATCA_STATUS atcab_lock_data_slot (uint16_t slot)

  Lock an individual slot in the data zone on an ATECC device. Not available for ATSHA devices. Slot must be configured to be slot lockable (KeyConfig.Lockable=1).
- ATCA_STATUS atcab_mac (uint8_t mode, uint16_t key_id, const uint8_t ∗challenge, uint8_t ∗digest)

  Executes MAC command, which computes a SHA-256 digest of a key stored in the device, a challenge, and other information on the device.
- ATCA_STATUS atcab_nonce_base (uint8_t mode, uint16_t zero, const uint8_t ∗num_in, uint8_t ∗rand_out)

  Executes Nonce command, which loads a random or fixed nonce/data into the device for use by subsequent commands.
- ATCA_STATUS atcab_nonce (const uint8_t ∗num_in)

  Execute a Nonce command in pass-through mode to initialize TempKey to a specified value.
- ATCA_STATUS atcab_nonce_load (uint8_t target, const uint8_t ∗num_in, uint16_t num_in_size)

  Execute a Nonce command in pass-through mode to load one of the device's internal buffers with a fixed value.
- ATCA_STATUS atcab_nonce_rand (const uint8_t ∗num_in, uint8_t ∗rand_out)

  Execute a Nonce command to generate a random nonce combining a host nonce (num_in) and a device random number.
- ATCA_STATUS atcab_challenge (const uint8_t ∗num_in)

  Execute a Nonce command in pass-through mode to initialize TempKey to a specified value.
- ATCA_STATUS atcab_challenge_seed_update (const uint8_t ∗num_in, uint8_t ∗rand_out)

  Execute a Nonce command to generate a random challenge combining a host nonce (num_in) and a device random number.
- ATCA_STATUS atcab_priv_write (uint16_t key_id, const uint8_t priv_key[36], uint16_t write_key_id, const uint8_t write_key[32])

  Executes PrivWrite command, to write externally generated ECC private keys into the device.
- ATCA_STATUS atcab_random (uint8_t ∗rand_out)

  Executes Random command, which generates a 32 byte random number from the CryptoAuth device.
- ATCA_STATUS atcab_read_zone (uint8_t zone, uint16_t slot, uint8_t block, uint8_t offset, uint8_t ∗data, uint8_t len)

  Executes Read command, which reads either 4 or 32 bytes of data from a given slot, configuration zone, or the OTP zone.
- ATCA_STATUS atcab_is_locked (uint8_t zone, bool ∗is_locked)

  Executes Read command, which reads the configuration zone to see if the specified zone is locked.
- ATCA_STATUS atcab_is_slot_locked (uint16_t slot, bool ∗is_locked)

  Executes Read command, which reads the configuration zone to see if the specified slot is locked.
- ATCA_STATUS atcab_read_bytes_zone (uint8_t zone, uint16_t slot, size_t offset, uint8_t ∗data, size_↩ t length)

  Used to read an arbitrary number of bytes from any zone configured for clear reads.
- ATCA_STATUS atcab_read_serial_number (uint8_t ∗serial_number)

  Executes Read command, which reads the 9 byte serial number of the device from the config zone.
- ATCA_STATUS atcab_read_pubkey (uint16_t slot, uint8_t ∗public_key)

  Executes Read command to read an ECC P256 public key from a slot configured for clear reads.
- ATCA_STATUS atcab_read_sig (uint16_t slot, uint8_t ∗sig)

Executes Read command to read a 64 byte ECDSA P256 signature from a slot configured for clear reads.

- ATCA_STATUS atcab_read_config_zone (uint8_t ∗config_data)

  Executes Read command to read the complete device configuration zone.
- ATCA_STATUS atcab_cmp_config_zone (uint8_t ∗config_data, bool ∗same_config)

  Compares a specified configuration zone with the configuration zone currently on the device.
- ATCA_STATUS atcab_read_enc (uint16_t key_id, uint8_t block, uint8_t ∗data, const uint8_t ∗enc_key, const uint16_t enc_key_id)

  Executes Read command on a slot configured for encrypted reads and decrypts the data to return it as plaintext.
- ATCA_STATUS atcab_secureboot (uint8_t mode, uint16_t param2, const uint8_t ∗digest, const uint8_↩ t ∗signature, uint8_t ∗mac)

  Executes Secure Boot command, which provides support for secure boot of an external MCU or MPU.
- ATCA_STATUS atcab_secureboot_mac (uint8_t mode, const uint8_t ∗digest, const uint8_t ∗signature, const uint8_t ∗num_in, const uint8_t ∗io_key, bool ∗is_verified)

  Executes Secure Boot command with encrypted digest and validated MAC response using the IO protection key.
- ATCA_STATUS atcab_selftest (uint8_t mode, uint16_t param2, uint8_t ∗result)

  Executes the SelfTest command, which performs a test of one or more of the cryptographic engines within the AT↩ ECC608A chip.
- ATCA_STATUS atcab_sha_base (uint8_t mode, uint16_t length, const uint8_t ∗data_in, uint8_t ∗data_out, uint16_t ∗data_out_size)

  Executes SHA command, which computes a SHA-256 or HMAC/SHA-256 digest for general purpose use by the host system.
- ATCA_STATUS atcab_sha_start (void)

  Executes SHA command to initialize SHA-256 calculation engine.
- ATCA_STATUS atcab_sha_update (const uint8_t ∗message)

  Executes SHA command to add 64 bytes of message data to the current context.
- ATCA_STATUS atcab_sha_end (uint8_t ∗digest, uint16_t length, const uint8_t ∗message)

  Executes SHA command to complete SHA-256 or HMAC/SHA-256 operation.
- ATCA_STATUS atcab_sha_read_context (uint8_t ∗context, uint16_t ∗context_size)

  Executes SHA command to read the SHA-256 context back. Only for ATECC608A with SHA-256 contexts. HMAC not supported.
- ATCA_STATUS atcab_sha_write_context (const uint8_t ∗context, uint16_t context_size)

  Executes SHA command to write (restore) a SHA-256 context into the the device. Only supported for ATECC608A with SHA-256 contexts.
- ATCA_STATUS atcab_sha (uint16_t length, const uint8_t ∗message, uint8_t ∗digest)

  Use the SHA command to compute a SHA-256 digest.
- ATCA_STATUS atcab_hw_sha2_256 (const uint8_t ∗data, size_t data_size, uint8_t ∗digest)

  Use the SHA command to compute a SHA-256 digest.
- ATCA_STATUS atcab_hw_sha2_256_init (atca_sha256_ctx_t ∗ctx)

  Initialize a SHA context for performing a hardware SHA-256 operation on a device. Note that only one SHA operation can be run at a time.
- ATCA_STATUS atcab_hw_sha2_256_update (atca_sha256_ctx_t ∗ctx, const uint8_t ∗data, size_t data_↩ size)

  Add message data to a SHA context for performing a hardware SHA-256 operation on a device.
- ATCA_STATUS atcab_hw_sha2_256_finish (atca_sha256_ctx_t ∗ctx, uint8_t ∗digest)

  Finish SHA-256 digest for a SHA context for performing a hardware SHA-256 operation on a device.
- ATCA_STATUS atcab_sha_hmac_init (atca_hmac_sha256_ctx_t ∗ctx, uint16_t key_slot)

  Executes SHA command to start an HMAC/SHA-256 operation.
- ATCA_STATUS atcab_sha_hmac_update (atca_hmac_sha256_ctx_t ∗ctx, const uint8_t ∗data, size_t data↩ _size)

  Executes SHA command to add an arbitrary amount of message data to a HMAC/SHA-256 operation.
- ATCA_STATUS atcab_sha_hmac_finish (atca_hmac_sha256_ctx_t ∗ctx, uint8_t ∗digest, uint8_t target)

  Executes SHA command to complete a HMAC/SHA-256 operation.

- ATCA_STATUS atcab_sha_hmac (const uint8_t ∗data, size_t data_size, uint16_t key_slot, uint8_t ∗digest, uint8_t target)

  Use the SHA command to compute an HMAC/SHA-256 operation.
- ATCA_STATUS atcab_sign_base (uint8_t mode, uint16_t key_id, uint8_t ∗signature)

  Executes the Sign command, which generates a signature using the ECDSA algorithm.
- ATCA_STATUS atcab_sign (uint16_t key_id, const uint8_t ∗msg, uint8_t ∗signature)

  Executes Sign command, to sign a 32-byte external message using the private key in the specified slot. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608A device or TempKey for other devices.
- ATCA_STATUS atcab_sign_internal (uint16_t key_id, bool is_invalidate, bool is_full_sn, uint8_t ∗signature)

  Executes Sign command to sign an internally generated message.
- ATCA_STATUS atcab_updateextra (uint8_t mode, uint16_t new_value)

  Executes UpdateExtra command to update the values of the two extra bytes within the Configuration zone (bytes 84 and 85).
- ATCA_STATUS atcab_verify (uint8_t mode, uint16_t key_id, const uint8_t ∗signature, const uint8_t ∗public_key, const uint8_t ∗other_data, uint8_t ∗mac)

  Executes the Verify command, which takes an ECDSA [R,S] signature and verifies that it is correctly generated from a given message and public key. In all cases, the signature is an input to the command.
- ATCA_STATUS atcab_verify_extern (const uint8_t ∗message, const uint8_t ∗signature, const uint8_t ∗public_key, bool ∗is_verified)

  Executes the Verify command, which verifies a signature (ECDSA verify operation) with all components (message, signature, and public key) supplied. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608A device or TempKey for other devices.
- ATCA_STATUS atcab_verify_extern_mac (const uint8_t ∗message, const uint8_t ∗signature, const uint8_t ∗public_key, const uint8_t ∗num_in, const uint8_t ∗io_key, bool ∗is_verified)

  Executes the Verify command with verification MAC, which verifies a signature (ECDSA verify operation) with all components (message, signature, and public key) supplied. This function is only available on the ATECC608A.
- ATCA_STATUS atcab_verify_stored (const uint8_t ∗message, const uint8_t ∗signature, uint16_t key_id, bool ∗is_verified)

  Executes the Verify command, which verifies a signature (ECDSA verify operation) with a public key stored in the device. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608A device or TempKey for other devices.
- ATCA_STATUS atcab_verify_stored_mac (const uint8_t ∗message, const uint8_t ∗signature, uint16_t key_id, const uint8_t ∗num_in, const uint8_t ∗io_key, bool ∗is_verified)

  Executes the Verify command with verification MAC, which verifies a signature (ECDSA verify operation) with a public key stored in the device. This function is only available on the ATECC608A.
- ATCA_STATUS atcab_verify_validate (uint16_t key_id, const uint8_t ∗signature, const uint8_t ∗other_data, bool ∗is_verified)

  Executes the Verify command in Validate mode to validate a public key stored in a slot.
- ATCA_STATUS atcab_verify_invalidate (uint16_t key_id, const uint8_t ∗signature, const uint8_t ∗other_data, bool ∗is_verified)

  Executes the Verify command in Invalidate mode which invalidates a previously validated public key stored in a slot.
- ATCA_STATUS atcab_write (uint8_t zone, uint16_t address, const uint8_t ∗value, const uint8_t ∗mac)

  Executes the Write command, which writes either one four byte word or a 32-byte block to one of the EEPROM zones on the device. Depending upon the value of the WriteConfig byte for this slot, the data may be required to be encrypted by the system prior to being sent to the device. This command cannot be used to write slots configured as ECC private keys.
- ATCA_STATUS atcab_write_zone (uint8_t zone, uint16_t slot, uint8_t block, uint8_t offset, const uint8_t ∗data, uint8_t len)

  Executes the Write command, which writes either 4 or 32 bytes of data into a device zone.
- ATCA_STATUS atcab_write_bytes_zone (uint8_t zone, uint16_t slot, size_t offset_bytes, const uint8_t ∗data, size_t length)

  Executes the Write command, which writes data into the configuration, otp, or data zones with a given byte offset and length. Offset and length must be multiples of a word (4 bytes).
- ATCA_STATUS atcab_write_pubkey (uint16_t slot, const uint8_t ∗public_key)

  Uses the write command to write a public key to a slot in the proper format.

- ATCA_STATUS atcab_write_config_zone (const uint8_t ∗config_data)

    Executes the Write command, which writes the configuration zone.
- ATCA_STATUS atcab_write_enc (uint16_t key_id, uint8_t block, const uint8_t ∗data, const uint8_t ∗enc_key, const uint16_t enc_key_id)

    Executes the Write command, which performs an encrypted write of a 32 byte block into given slot.
- ATCA_STATUS atcab_write_config_counter (uint16_t counter_id, uint32_t counter_value)

    Initialize one of the monotonic counters in device with a specific value.

Variables

- ATCADevice _gDevice

15.2.1 Detailed Description

CryptoAuthLib Basic API methods - a simple crypto authentication API. These methods manage a global ATCA←Device object behind the scenes. They also manage the wake/idle state transitions so callers don't need to.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.3 atca_basic_aes.c File Reference

CryptoAuthLib Basic API methods for AES command.

```
#include "atca_basic.h"
#include "atca_execution.h"
```

Functions

- ATCA_STATUS atcab_aes (uint8_t mode, uint16_t key_id, const uint8_t ∗aes_in, uint8_t ∗aes_out)

    Compute the AES-128 encrypt, decrypt, or GFM calculation.
- ATCA_STATUS atcab_aes_encrypt (uint16_t key_id, uint8_t key_block, const uint8_t ∗plaintext, uint8_←t ∗ciphertext)

    Perform an AES-128 encrypt operation with a key in the device.
- ATCA_STATUS atcab_aes_decrypt (uint16_t key_id, uint8_t key_block, const uint8_t ∗ciphertext, uint8_t ∗plaintext)

    Perform an AES-128 decrypt operation with a key in the device.
- ATCA_STATUS atcab_aes_gfm (const uint8_t ∗h, const uint8_t ∗input, uint8_t ∗output)

    Perform a Galois Field Multiply (GFM) operation.

15.3.1 Detailed Description

CryptoAuthLib Basic API methods for AES command.

The AES command supports 128-bit AES encryption or decryption of small messages or data packets in ECB mode. Also can perform GFM (Galois Field Multiply) calculation in support of AES-GCM.

Note

List of devices that support this command - ATECC608A. Refer to device datasheet for full details.

Copyright

15.4 atca_basic_aes_cbc.c File Reference

CryptoAuthLib Basic API methods for AES CBC mode.

```
#include "atca_basic.h"
```

Functions

- ATCA_STATUS atcab_aes_cbc_init (atca_aes_cbc_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, const uint8_t ∗iv)

  Initialize context for AES CBC operation.
- ATCA_STATUS atcab_aes_cbc_encrypt_block (atca_aes_cbc_ctx_t ∗ctx, const uint8_t ∗plaintext, uint8_t ∗ciphertext)

  Encrypt a block of data using CBC mode and a key within the ATECC608A. atcab_aes_cbc_init() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_cbc_decrypt_block (atca_aes_cbc_ctx_t ∗ctx, const uint8_t ∗ciphertext, uint8_t ∗plaintext)

  Decrypt a block of data using CBC mode and a key within the ATECC608A. atcab_aes_cbc_init() should be called before the first use of this function.

15.4.1 Detailed Description

CryptoAuthLib Basic API methods for AES CBC mode.

The AES command supports 128-bit AES encryption or decryption of small messages or data packets in ECB mode. Also can perform GFM (Galois Field Multiply) calculation in support of AES-GCM.

Note

List of devices that support this command - ATECC608A. Refer to device datasheet for full details.

Copyright

15.5 atca_basic_aes_cmac.c File Reference

CryptoAuthLib Basic API methods for AES CBC_MAC mode.

```
#include "atca_basic.h"
```

Functions

- ATCA_STATUS atcab_aes_cmac_init (atca_aes_cmac_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block)

  Initialize a CMAC calculation using an AES-128 key in the ATECC608A.
- ATCA_STATUS atcab_aes_cmac_update (atca_aes_cmac_ctx_t ∗ctx, const uint8_t ∗data, uint32_t data_←size)

  Add data to an initialized CMAC calculation.
- ATCA_STATUS atcab_aes_cmac_finish (atca_aes_cmac_ctx_t ∗ctx, uint8_t ∗cmac, uint32_t cmac_size)

  Finish a CMAC operation returning the CMAC value.

15.5.1 Detailed Description

CryptoAuthLib Basic API methods for AES CBC_MAC mode.

The AES command supports 128-bit AES encryption or decryption of small messages or data packets in ECB mode. Also can perform GFM (Galois Field Multiply) calculation in support of AES-GCM.

Note

List of devices that support this command - ATECC608A. Refer to device datasheet for full details.

Copyright

15.6 atca_basic_aes_ctr.c File Reference

CryptoAuthLib Basic API methods for AES CTR mode.

```
#include "basic/atca_basic.h"
```

Functions

- ATCA_STATUS atcab_aes_ctr_init (atca_aes_ctr_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, uint8_↩ t counter_size, const uint8_t ∗iv)

  Initialize context for AES CTR operation with an existing IV, which is common when start a decrypt operation.
- ATCA_STATUS atcab_aes_ctr_init_rand (atca_aes_ctr_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, uint8_t counter_size, uint8_t ∗iv)

  Initialize context for AES CTR operation with a random nonce and counter set to 0 as the IV, which is common when starting an encrypt operation.
- ATCA_STATUS atcab_aes_ctr_increment (atca_aes_ctr_ctx_t ∗ctx)

  Increments AES CTR counter value.
- ATCA_STATUS atcab_aes_ctr_block (atca_aes_ctr_ctx_t ∗ctx, const uint8_t ∗input, uint8_t ∗output)

  Process a block of data using CTR mode and a key within the ATECC608A device. atcab_aes_ctr_init() or atcab_aes_ctr_init_rand() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_ctr_encrypt_block (atca_aes_ctr_ctx_t ∗ctx, const uint8_t ∗plaintext, uint8_↩ t ∗ciphertext)

  Encrypt a block of data using CTR mode and a key within the ATECC608A device. atcab_aes_ctr_init() or atcab_aes_ctr_init_rand() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_ctr_decrypt_block (atca_aes_ctr_ctx_t ∗ctx, const uint8_t ∗ciphertext, uint8_↩ t ∗plaintext)

  Decrypt a block of data using CTR mode and a key within the ATECC608A device. atcab_aes_ctr_init() or atcab_aes_ctr_init_rand() should be called before the first use of this function.

15.6.1 Detailed Description

CryptoAuthLib Basic API methods for AES CTR mode.

The AES command supports 128-bit AES encryption or decryption of small messages or data packets in ECB mode. Also can perform GFM (Galois Field Multiply) calculation in support of AES-GCM.

Note

List of devices that support this command - ATECC608A. Refer to device datasheet for full details.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.7 atca_basic_aes_gcm.c File Reference

CryptoAuthLib Basic API methods for AES GCM mode.

```
#include "atca_basic_aes_gcm.h"
#include "atca_compiler.h"
```

- const char ∗ atca_basic_aes_gcm_version = "1.0"
- ATCA_STATUS atcab_aes_gcm_init (atca_aes_gcm_ctx_t ∗ctx, uint16_t key_id, uint8_t key_block, const uint8_t ∗iv, size_t iv_size)

  Initialize context for AES GCM operation with an existing IV, which is common when starting a decrypt operation.

- ATCA_STATUS atcab_aes_gcm_init_rand (atca_aes_gcm_ctx_t *ctx, uint16_t key_id, uint8_t key_block, size_t rand_size, const uint8_t *free_field, size_t free_field_size, uint8_t *iv)

  Initialize context for AES GCM operation with a IV composed of a random and optional fixed(free) field, which is common when starting an encrypt operation.
- ATCA_STATUS atcab_aes_gcm_aad_update (atca_aes_gcm_ctx_t *ctx, const uint8_t *aad, uint32_t aad↩_size)

  Process Additional Authenticated Data (AAD) using GCM mode and a key within the ATECC608A device.
- ATCA_STATUS atcab_aes_gcm_encrypt_update (atca_aes_gcm_ctx_t *ctx, const uint8_t *plaintext, uint32_t plaintext_size, uint8_t *ciphertext)

  Encrypt data using GCM mode and a key within the ATECC608A device. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_gcm_encrypt_finish (atca_aes_gcm_ctx_t *ctx, uint8_t *tag, size_t tag_size)

  Complete a GCM encrypt operation returning the authentication tag.
- ATCA_STATUS atcab_aes_gcm_decrypt_update (atca_aes_gcm_ctx_t *ctx, const uint8_t *ciphertext, uint32_t ciphertext_size, uint8_t *plaintext)

  Decrypt data using GCM mode and a key within the ATECC608A device. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_gcm_decrypt_finish (atca_aes_gcm_ctx_t *ctx, const uint8_t *tag, size_t tag↩_size, bool *is_verified)

  Complete a GCM decrypt operation verifying the authentication tag.

15.7.1 Detailed Description

CryptoAuthLib Basic API methods for AES GCM mode.

The AES command supports 128-bit AES encryption or decryption of small messages or data packets in ECB mode. Also can perform GFM (Galois Field Multiply) calculation in support of AES-GCM.

Note

List of devices that support this command - ATECC608A. Refer to device datasheet for full details.

Copyright

15.8 atca_basic_aes_gcm.h File Reference

Unity tests for the cryptoauthlib AES GCM functions.

```
#include "cryptoauthlib.h"
```

Data Structures

- struct atca_aes_gcm_ctx

- typedef struct atca_aes_gcm_ctx atca_aes_gcm_ctx_t
- const char * atca_basic_aes_gcm_version
- ATCA_STATUS atcab_aes_gcm_init (atca_aes_gcm_ctx_t *ctx, uint16_t key_id, uint8_t key_block, const uint8_t *iv, size_t iv_size)

  Initialize context for AES GCM operation with an existing IV, which is common when starting a decrypt operation.
- ATCA_STATUS atcab_aes_gcm_init_rand (atca_aes_gcm_ctx_t *ctx, uint16_t key_id, uint8_t key_block, size_t rand_size, const uint8_t *free_field, size_t free_field_size, uint8_t *iv)

  Initialize context for AES GCM operation with a IV composed of a random and optional fixed(free) field, which is common when starting an encrypt operation.
- ATCA_STATUS atcab_aes_gcm_aad_update (atca_aes_gcm_ctx_t *ctx, const uint8_t *aad, uint32_t aad←_size)

  Process Additional Authenticated Data (AAD) using GCM mode and a key within the ATECC608A device.
- ATCA_STATUS atcab_aes_gcm_encrypt_update (atca_aes_gcm_ctx_t *ctx, const uint8_t *plaintext, uint32_t plaintext_size, uint8_t *ciphertext)

  Encrypt data using GCM mode and a key within the ATECC608A device. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_gcm_encrypt_finish (atca_aes_gcm_ctx_t *ctx, uint8_t *tag, size_t tag_size)

  Complete a GCM encrypt operation returning the authentication tag.
- ATCA_STATUS atcab_aes_gcm_decrypt_update (atca_aes_gcm_ctx_t *ctx, const uint8_t *ciphertext, uint32_t ciphertext_size, uint8_t *plaintext)

  Decrypt data using GCM mode and a key within the ATECC608A device. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function.
- ATCA_STATUS atcab_aes_gcm_decrypt_finish (atca_aes_gcm_ctx_t *ctx, const uint8_t *tag, size_t tag←_size, bool *is_verified)

  Complete a GCM decrypt operation verifying the authentication tag.

15.8.1 Detailed Description

Unity tests for the cryptoauthlib AES GCM functions.

Copyright

15.8.2 Typedef Documentation

15.8.2.1 atca_aes_gcm_ctx_t

```
typedef struct atca_aes_gcm_ctx atca_aes_gcm_ctx_t
```

Context structure for AES GCM operations.

15.8.3 Function Documentation

15.8.3.1 atcab_aes_gcm_aad_update()

```
ATCA_STATUS atcab_aes_gcm_aad_update (
            atca_aes_gcm_ctx_t * ctx,
            const uint8_t * aad,
            uint32_t aad_size )
```

Process Additional Authenticated Data (AAD) using GCM mode and a key within the ATECC608A device.

This can be called multiple times. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function. When there is AAD to include, this should be called before atcab_aes_gcm_encrypt_update() or atcab_aes_gcm_decrypt_update().

Parameters

in	*ctx*	AES GCM context
in	*aad*	Additional authenticated data to be added
in	*aad_size*	Size of aad in bytes

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.8.3.2 atcab_aes_gcm_decrypt_finish()

```
ATCA_STATUS atcab_aes_gcm_decrypt_finish (
            atca_aes_gcm_ctx_t * ctx,
            const uint8_t * tag,
            size_t tag_size,
            bool * is_verified )
```

Complete a GCM decrypt operation verifying the authentication tag.

Parameters

in	*ctx*	AES GCM context structure.
in	*tag*	Expected authentication tag.
in	*tag_size*	Size of tag in bytes (12 to 16 bytes).
out	*is_verified*	Returns whether or not the tag verified.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.8.3.3 atcab_aes_gcm_decrypt_update()

```
ATCA_STATUS atcab_aes_gcm_decrypt_update (
            atca_aes_gcm_ctx_t * ctx,
            const uint8_t * ciphertext,
            uint32_t ciphertext_size,
            uint8_t * plaintext )
```

Decrypt data using GCM mode and a key within the ATECC608A device. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function.

Parameters

in	*ctx*	AES GCM context structure.
in	*ciphertext*	Ciphertext to be decrypted.
in	*ciphertext_size*	Size of ciphertext in bytes.
out	*plaintext*	Decrypted data is returned here.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.8.3.4 atcab_aes_gcm_encrypt_finish()

```
ATCA_STATUS atcab_aes_gcm_encrypt_finish (
            atca_aes_gcm_ctx_t * ctx,
            uint8_t * tag,
            size_t tag_size )
```

Complete a GCM encrypt operation returning the authentication tag.

Parameters

in	*ctx*	AES GCM context structure.
out	*tag*	Authentication tag is returned here.
in	*tag_size*	Tag size in bytes (12 to 16 bytes).

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.8.3.5 atcab_aes_gcm_encrypt_update()

```
ATCA_STATUS atcab_aes_gcm_encrypt_update (
            atca_aes_gcm_ctx_t * ctx,
            const uint8_t * plaintext,
            uint32_t plaintext_size,
            uint8_t * ciphertext )
```

Encrypt data using GCM mode and a key within the ATECC608A device. atcab_aes_gcm_init() or atcab_aes_gcm_init_rand() should be called before the first use of this function.

Parameters

in	*ctx*	AES GCM context structure.
in	*plaintext*	Plaintext to be encrypted (16 bytes).
in	*plaintext_size*	Size of plaintext in bytes.
out	*ciphertext*	Encrypted data is returned here.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.8.3.6 atcab_aes_gcm_init()

```
ATCA_STATUS atcab_aes_gcm_init (
            atca_aes_gcm_ctx_t * ctx,
            uint16_t key_id,
            uint8_t key_block,
            const uint8_t * iv,
            size_t iv_size )
```

Initialize context for AES GCM operation with an existing IV, which is common when starting a decrypt operation.

Parameters

in	*ctx*	AES GCM context to be initialized.
in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.
in	*iv*	Initialization vector.
in	*iv_size*	Size of IV in bytes. Standard is 12 bytes.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.8.3.7 atcab_aes_gcm_init_rand()

```
ATCA_STATUS atcab_aes_gcm_init_rand (
            atca_aes_gcm_ctx_t * ctx,
            uint16_t key_id,
            uint8_t key_block,
            size_t rand_size,
            const uint8_t * free_field,
            size_t free_field_size,
            uint8_t * iv )
```

Initialize context for AES GCM operation with a IV composed of a random and optional fixed(free) field, which is common when starting an encrypt operation.

Parameters

in	*ctx*	AES CTR context to be initialized.
in	*key_id*	Key location. Can either be a slot number or ATCA_TEMPKEY_KEYID for TempKey.
in	*key_block*	Index of the 16-byte block to use within the key location for the actual key.
in	*rand_size*	Size of the random field in bytes. Minimum and recommended size is 12 bytes. Max is 32 bytes.
in	*free_field*	Fixed data to include in the IV after the random field. Can be NULL if not used.
in	*free_field_size*	Size of the free field in bytes.
out	*iv*	Initialization vector is returned here. Its size will be rand_size and free_field_size combined.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.9 atca_basic_checkmac.c File Reference

CryptoAuthLib Basic API methods for CheckMAC command.

```
#include "atca_basic.h"
#include "atca_execution.h"
```

Functions

- ATCA_STATUS atcab_checkmac (uint8_t mode, uint16_t key_id, const uint8_t *challenge, const uint8_t *response, const uint8_t *other_data)

  Compares a MAC response with input values.

15.9.1 Detailed Description

CryptoAuthLib Basic API methods for CheckMAC command.

The CheckMac command calculates a MAC response that would have been generated on a different CryptoAuthentication device and then compares the result with input value.

Note

List of devices that support this command - ATSHA204A, ATECC108A, ATECC508A, and ATECC608A. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

15.10 atca_basic_counter.c File Reference

CryptoAuthLib Basic API methods for Counter command.

```
#include "atca_basic.h"
#include "atca_execution.h"
```

Functions

- ATCA_STATUS atcab_counter (uint8_t mode, uint16_t counter_id, uint32_t ∗counter_value)
  Compute the Counter functions.
- ATCA_STATUS atcab_counter_increment (uint16_t counter_id, uint32_t ∗counter_value)
  Increments one of the device's monotonic counters.
- ATCA_STATUS atcab_counter_read (uint16_t counter_id, uint32_t ∗counter_value)
  Read one of the device's monotonic counters.

15.10.1 Detailed Description

CryptoAuthLib Basic API methods for Counter command.

The Counter command reads or increments the binary count value for one of the two monotonic counters

Note

List of devices that support this command - ATECC508A and ATECC608A. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

15.11 atca_basic_derivekey.c File Reference

CryptoAuthLib Basic API methods for DeriveKey command.

```
#include "atca_basic.h"
#include "atca_execution.h"
```

Functions

- ATCA_STATUS atcab_derivekey (uint8_t mode, uint16_t target_key, const uint8_t ∗mac)

  Executes the DeviveKey command for deriving a new key from a nonce (TempKey) and an existing key.

15.11.1 Detailed Description

CryptoAuthLib Basic API methods for DeriveKey command.

The DeriveKey command combines the current value of a key with the nonce stored in TempKey using SHA-256 and derives a new key.

Note

List of devices that support this command - ATSHA204A, ATECC108A, ATECC508A, and ATECC608A. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.12 atca_basic_ecdh.c File Reference

CryptoAuthLib Basic API methods for ECDH command.

```
#include "atca_basic.h"
#include "atca_execution.h"
#include "host/atca_host.h"
```

Functions

- ATCA_STATUS atcab_ecdh_base (uint8_t mode, uint16_t key_id, const uint8_t ∗public_key, uint8_t ∗pms, uint8_t ∗out_nonce)

  Base function for generating premaster secret key using ECDH.
- ATCA_STATUS atcab_ecdh (uint16_t key_id, const uint8_t ∗public_key, uint8_t ∗pms)

  ECDH command with a private key in a slot and the premaster secret is returned in the clear.
- ATCA_STATUS atcab_ecdh_enc (uint16_t key_id, const uint8_t ∗public_key, uint8_t ∗pms, const uint8_t ∗read_key, uint16_t read_key_id)

  ECDH command with a private key in a slot and the premaster secret is read from the next slot.
- ATCA_STATUS atcab_ecdh_ioenc (uint16_t key_id, const uint8_t ∗public_key, uint8_t ∗pms, const uint8_t ∗io_key)

  ECDH command with a private key in a slot and the premaster secret is returned encrypted using the IO protection key.
- ATCA_STATUS atcab_ecdh_tempkey (const uint8_t ∗public_key, uint8_t ∗pms)

  ECDH command with a private key in TempKey and the premaster secret is returned in the clear.
- ATCA_STATUS atcab_ecdh_tempkey_ioenc (const uint8_t ∗public_key, uint8_t ∗pms, const uint8_t ∗io_key)

  ECDH command with a private key in TempKey and the premaster secret is returned encrypted using the IO protection key.

15.12.1 Detailed Description

CryptoAuthLib Basic API methods for ECDH command.

The ECDH command implements the Elliptic Curve Diffie-Hellman algorithm to combine an internal private key with an external public key to calculate a shared secret.

Note

List of devices that support this command - ATECC508A, ATECC608A. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.13 atca_basic_gendig.c File Reference

CryptoAuthLib Basic API methods for GenDig command.

```
#include "atca_basic.h"
#include "atca_execution.h"
```

Functions

- ATCA_STATUS atcab_gendig (uint8_t zone, uint16_t key_id, const uint8_t *other_data, uint8_t other_data_size)

    Issues a GenDig command, which performs a SHA256 hash on the source data indicated by zone with the contents of TempKey. See the CryptoAuth datasheet for your chip to see what the values of zone correspond to.

15.13.1 Detailed Description

CryptoAuthLib Basic API methods for GenDig command.

The GenDig command uses SHA-256 to combine a stored value with the contents of TempKey, which must have been valid prior to the execution of this command.

Note

List of devices that support this command - ATSHA204A, ATECC108A, ATECC508A, and ATECC608A. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.14 atca_basic_genkey.c File Reference

CryptoAuthLib Basic API methods for GenKey command.

```
#include "atca_basic.h"
#include "atca_execution.h"
```

Functions

- ATCA_STATUS atcab_genkey_base (uint8_t mode, uint16_t key_id, const uint8_t *other_data, uint8_t *public_key)

  Issues GenKey command, which can generate a private key, compute a public key, nd/or compute a digest of a public key.
- ATCA_STATUS atcab_genkey (uint16_t key_id, uint8_t *public_key)

  Issues GenKey command, which generates a new random private key in slot and returns the public key.
- ATCA_STATUS atcab_get_pubkey (uint16_t key_id, uint8_t *public_key)

  Uses GenKey command to calculate the public key from an existing private key in a slot.

15.14.1 Detailed Description

CryptoAuthLib Basic API methods for GenKey command.

The GenKey command is used for creating ECC private keys, generating ECC public keys, and for digest calculations involving public keys.

Note

List of devices that support this command - ATECC108A, ATECC508A, ATECC608A. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.15 atca_basic_hmac.c File Reference

CryptoAuthLib Basic API methods for HMAC command.

```
#include "atca_basic.h"
#include "atca_execution.h"
```

Functions

- ATCA_STATUS atcab_hmac (uint8_t mode, uint16_t key_id, uint8_t *digest)

  Issues a HMAC command, which computes an HMAC/SHA-256 digest of a key stored in the device, a challenge, and other information on the device.

15.15.1 Detailed Description

CryptoAuthLib Basic API methods for HMAC command.

The HMAC command computes an HMAC/SHA-256 digest using a key stored in the device over a challenge stored in the TempKey register, and/or other information stored within the device.

Note

List of devices that support this command - ATSHA204A, ATECC108A, and ATECC508A . There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

15.16 atca_basic_info.c File Reference

CryptoAuthLib Basic API methods for Info command.

```
#include "atca_basic.h"
#include "atca_execution.h"
```

Functions

- ATCA_STATUS atcab_info_base (uint8_t mode, uint16_t param2, uint8_t ∗out_data)

  Issues an Info command, which return internal device information and can control GPIO and the persistent latch.
- ATCA_STATUS atcab_info (uint8_t ∗revision)

  Use the Info command to get the device revision (DevRev).
- ATCA_STATUS atcab_info_get_latch (bool ∗state)

  Use the Info command to get the persistent latch current state for an ATECC608A device.
- ATCA_STATUS atcab_info_set_latch (bool state)

  Use the Info command to set the persistent latch state for an ATECC608A device.

15.16.1 Detailed Description

CryptoAuthLib Basic API methods for Info command.

Info command returns a variety of static and dynamic information about the device and its state. Also is used to control the GPIO pin and the persistent latch.

Note

The ATSHA204A refers to this command as DevRev instead of Info, however, the OpCode and operation is the same.

List of devices that support this command - ATSHA204A, ATECC108A, ATECC508A & ATECC608A. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

15.17 atca_basic_kdf.c File Reference

CryptoAuthLib Basic API methods for KDF command.

```
#include "atca_basic.h"
#include "atca_execution.h"
```

Functions

- ATCA_STATUS atcab_kdf (uint8_t mode, uint16_t key_id, const uint32_t details, const uint8_t ∗message, uint8_t ∗out_data, uint8_t ∗out_nonce)

    Executes the KDF command, which derives a new key in PRF, AES, or HKDF modes.

15.17.1 Detailed Description

CryptoAuthLib Basic API methods for KDF command.

The KDF command implements one of a number of Key Derivation Functions (KDF). Generally this function combines a source key with an input string and creates a result key/digest/array. Three algorithms are currently supported: PRF, HKDF and AES.

Note

List of devices that support this command - ATECC608A. Refer to device datasheet for full details.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.18 atca_basic_lock.c File Reference

CryptoAuthLib Basic API methods for Lock command.

```
#include "atca_basic.h"
#include "atca_execution.h"
```

Functions

- ATCA_STATUS atcab_lock (uint8_t mode, uint16_t summary_crc)

    The Lock command prevents future modifications of the Configuration and/or Data and OTP zones. If the device is so configured, then this command can be used to lock individual data slots. This command fails if the designated area is already locked.
- ATCA_STATUS atcab_lock_config_zone (void)

    Unconditionally (no CRC required) lock the config zone.
- ATCA_STATUS atcab_lock_config_zone_crc (uint16_t summary_crc)

    Lock the config zone with summary CRC.
- ATCA_STATUS atcab_lock_data_zone (void)

    Unconditionally (no CRC required) lock the data zone (slots and OTP).
- ATCA_STATUS atcab_lock_data_zone_crc (uint16_t summary_crc)

    Lock the data zone (slots and OTP) with summary CRC.
- ATCA_STATUS atcab_lock_data_slot (uint16_t slot)

    Lock an individual slot in the data zone on an ATECC device. Not available for ATSHA devices. Slot must be configured to be slot lockable (KeyConfig.Lockable=1).

15.18.1 Detailed Description

CryptoAuthLib Basic API methods for Lock command.

The Lock command prevents future modifications of the Configuration zone, enables configured policies for Data and OTP zones, and can render individual slots read-only regardless of configuration.

Note

List of devices that support this command - ATSHA204A, ATECC108A, ATECC508A, ATECC608A. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

15.19 atca_basic_mac.c File Reference

CryptoAuthLib Basic API methods for MAC command.

```
#include "atca_basic.h"
#include "atca_execution.h"
```

Functions

- ATCA_STATUS atcab_mac (uint8_t mode, uint16_t key_id, const uint8_t ∗challenge, uint8_t ∗digest)

  Executes MAC command, which computes a SHA-256 digest of a key stored in the device, a challenge, and other information on the device.

15.19.1 Detailed Description

CryptoAuthLib Basic API methods for MAC command.

The MAC command computes a SHA-256 digest of a key stored in the device, a challenge, and other information on the device. The output of this command is the digest of this message.

Note

List of devices that support this command - ATSHA204A, ATECC108A, ATECC508A, and ATECC608A. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

15.20 atca_basic_nonce.c File Reference

CryptoAuthLib Basic API methods for Nonce command.

```
#include "atca_basic.h"
#include "atca_execution.h"
```

Functions

- ATCA_STATUS atcab_nonce_base (uint8_t mode, uint16_t zero, const uint8_t ∗num_in, uint8_t ∗rand_out)
  Executes Nonce command, which loads a random or fixed nonce/data into the device for use by subsequent commands.
- ATCA_STATUS atcab_nonce (const uint8_t ∗num_in)
  Execute a Nonce command in pass-through mode to initialize TempKey to a specified value.
- ATCA_STATUS atcab_nonce_load (uint8_t target, const uint8_t ∗num_in, uint16_t num_in_size)
  Execute a Nonce command in pass-through mode to load one of the device's internal buffers with a fixed value.
- ATCA_STATUS atcab_nonce_rand (const uint8_t ∗num_in, uint8_t ∗rand_out)
  Execute a Nonce command to generate a random nonce combining a host nonce (num_in) and a device random number.
- ATCA_STATUS atcab_challenge (const uint8_t ∗num_in)
  Execute a Nonce command in pass-through mode to initialize TempKey to a specified value.
- ATCA_STATUS atcab_challenge_seed_update (const uint8_t ∗num_in, uint8_t ∗rand_out)
  Execute a Nonce command to generate a random challenge combining a host nonce (num_in) and a device random number.

15.20.1 Detailed Description

CryptoAuthLib Basic API methods for Nonce command.

The Nonce command generates a nonce for use by a subsequent commands of the device by combining an internally generated random number with an input value from the system.

Note

List of devices that support this command - ATSHA204A, ATECC108A, ATECC508A, and ATECC608A. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.21 atca_basic_privwrite.c File Reference

CryptoAuthLib Basic API methods for PrivWrite command.

```
#include "atca_basic.h"
#include "atca_execution.h"
#include "host/atca_host.h"
```

Functions

- ATCA_STATUS atcab_priv_write (uint16_t key_id, const uint8_t priv_key[36], uint16_t write_key_id, const uint8_t write_key[32])

    Executes PrivWrite command, to write externally generated ECC private keys into the device.

15.21.1 Detailed Description

CryptoAuthLib Basic API methods for PrivWrite command.

The PrivWrite command is used to write externally generated ECC private keys into the device.

Note

List of devices that support this command - ATECC108A, ATECC508A, and ATECC608A. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

15.22 atca_basic_random.c File Reference

CryptoAuthLib Basic API methods for Random command.

```
#include "atca_basic.h"
#include "atca_execution.h"
```

Functions

- ATCA_STATUS atcab_random (uint8_t *rand_out)

    Executes Random command, which generates a 32 byte random number from the CryptoAuth device.

15.22.1 Detailed Description

CryptoAuthLib Basic API methods for Random command.

The Random command generates a random number for use by the system.

Note

List of devices that support this command - ATSHA204A, ATECC108A, ATECC508A, ATECC608A. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

15.23 atca_basic_read.c File Reference

CryptoAuthLib Basic API methods for Read command.

```
#include "atca_basic.h"
#include "atca_execution.h"
#include "host/atca_host.h"
```

Functions

- ATCA_STATUS atcab_read_zone (uint8_t zone, uint16_t slot, uint8_t block, uint8_t offset, uint8_t ∗data, uint8_t len)

  Executes Read command, which reads either 4 or 32 bytes of data from a given slot, configuration zone, or the OTP zone.
- ATCA_STATUS atcab_read_serial_number (uint8_t ∗serial_number)

  Executes Read command, which reads the 9 byte serial number of the device from the config zone.
- ATCA_STATUS atcab_is_slot_locked (uint16_t slot, bool ∗is_locked)

  Executes Read command, which reads the configuration zone to see if the specified slot is locked.
- ATCA_STATUS atcab_is_locked (uint8_t zone, bool ∗is_locked)

  Executes Read command, which reads the configuration zone to see if the specified zone is locked.
- ATCA_STATUS atcab_read_enc (uint16_t key_id, uint8_t block, uint8_t ∗data, const uint8_t ∗enc_key, const uint16_t enc_key_id)

  Executes Read command on a slot configured for encrypted reads and decrypts the data to return it as plaintext.
- ATCA_STATUS atcab_read_config_zone (uint8_t ∗config_data)

  Executes Read command to read the complete device configuration zone.
- ATCA_STATUS atcab_cmp_config_zone (uint8_t ∗config_data, bool ∗same_config)

  Compares a specified configuration zone with the configuration zone currently on the device.
- ATCA_STATUS atcab_read_sig (uint16_t slot, uint8_t ∗sig)

  Executes Read command to read a 64 byte ECDSA P256 signature from a slot configured for clear reads.
- ATCA_STATUS atcab_read_pubkey (uint16_t slot, uint8_t ∗public_key)

  Executes Read command to read an ECC P256 public key from a slot configured for clear reads.
- ATCA_STATUS atcab_read_bytes_zone (uint8_t zone, uint16_t slot, size_t offset, uint8_t ∗data, size_↩ t length)

  Used to read an arbitrary number of bytes from any zone configured for clear reads.

15.23.1 Detailed Description

CryptoAuthLib Basic API methods for Read command.

The Read command reads words either 4-byte words or 32-byte blocks from one of the memory zones of the device. The data may optionally be encrypted before being returned to the system.

Note

List of devices that support this command - ATSHA204A, ATECC108A, ATECC508A, ATECC608A. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

15.24 atca_basic_secureboot.c File Reference

CryptoAuthLib Basic API methods for SecureBoot command.

```
#include "atca_basic.h"
#include "atca_execution.h"
#include "host/atca_host.h"
```

Functions

- ATCA_STATUS atcab_secureboot (uint8_t mode, uint16_t param2, const uint8_t ∗digest, const uint8_↩ t ∗signature, uint8_t ∗mac)

  Executes Secure Boot command, which provides support for secure boot of an external MCU or MPU.
- ATCA_STATUS atcab_secureboot_mac (uint8_t mode, const uint8_t ∗digest, const uint8_t ∗signature, const uint8_t ∗num_in, const uint8_t ∗io_key, bool ∗is_verified)

  Executes Secure Boot command with encrypted digest and validated MAC response using the IO protection key.

15.24.1 Detailed Description

CryptoAuthLib Basic API methods for SecureBoot command.

The SecureBoot command provides support for secure boot of an external MCU or MPU.

Note

List of devices that support this command - ATECC608A. Refer to device datasheet for full details.

Copyright

15.25 atca_basic_selftest.c File Reference

CryptoAuthLib Basic API methods for SelfTest command.

```
#include "atca_basic.h"
#include "atca_execution.h"
```

Functions

- ATCA_STATUS atcab_selftest (uint8_t mode, uint16_t param2, uint8_t ∗result)

  Executes the SelfTest command, which performs a test of one or more of the cryptographic engines within the AT↩ ECC608A chip.

15.25.1 Detailed Description

CryptoAuthLib Basic API methods for SelfTest command.

The SelfTest command performs a test of one or more of the cryptographic engines within the device.

Note

List of devices that support this command - ATECC608A. Refer to device datasheet for full details.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.26 atca_basic_sha.c File Reference

CryptoAuthLib Basic API methods for SHA command.

```
#include "atca_basic.h"
#include "atca_execution.h"
```

Data Structures

- struct hw_sha256_ctx

Functions

- ATCA_STATUS atcab_sha_base (uint8_t mode, uint16_t length, const uint8_t ∗message, uint8_t ∗data_out, uint16_t ∗data_out_size)

  Executes SHA command, which computes a SHA-256 or HMAC/SHA-256 digest for general purpose use by the host system.
- ATCA_STATUS atcab_sha_start (void)

  Executes SHA command to initialize SHA-256 calculation engine.
- ATCA_STATUS atcab_sha_update (const uint8_t ∗message)

  Executes SHA command to add 64 bytes of message data to the current context.
- ATCA_STATUS atcab_sha_end (uint8_t ∗digest, uint16_t length, const uint8_t ∗message)

  Executes SHA command to complete SHA-256 or HMAC/SHA-256 operation.
- ATCA_STATUS atcab_sha_read_context (uint8_t ∗context, uint16_t ∗context_size)

  Executes SHA command to read the SHA-256 context back. Only for ATECC608A with SHA-256 contexts. HMAC not supported.
- ATCA_STATUS atcab_sha_write_context (const uint8_t ∗context, uint16_t context_size)

  Executes SHA command to write (restore) a SHA-256 context into the the device. Only supported for ATECC608A with SHA-256 contexts.
- ATCA_STATUS atcab_sha (uint16_t length, const uint8_t ∗message, uint8_t ∗digest)

  Use the SHA command to compute a SHA-256 digest.
- ATCA_STATUS atcab_hw_sha2_256_init (atca_sha256_ctx_t ∗ctx)

  Initialize a SHA context for performing a hardware SHA-256 operation on a device. Note that only one SHA operation can be run at a time.

- ATCA_STATUS atcab_hw_sha2_256_update (atca_sha256_ctx_t ∗ctx, const uint8_t ∗data, size_t data_size)
  Add message data to a SHA context for performing a hardware SHA-256 operation on a device.
- ATCA_STATUS atcab_hw_sha2_256_finish (atca_sha256_ctx_t ∗ctx, uint8_t ∗digest)
  Finish SHA-256 digest for a SHA context for performing a hardware SHA-256 operation on a device.
- ATCA_STATUS atcab_hw_sha2_256 (const uint8_t ∗data, size_t data_size, uint8_t ∗digest)
  Use the SHA command to compute a SHA-256 digest.
- ATCA_STATUS atcab_sha_hmac_init (atca_hmac_sha256_ctx_t ∗ctx, uint16_t key_slot)
  Executes SHA command to start an HMAC/SHA-256 operation.
- ATCA_STATUS atcab_sha_hmac_update (atca_hmac_sha256_ctx_t ∗ctx, const uint8_t ∗data, size_t data_size)
  Executes SHA command to add an arbitrary amount of message data to a HMAC/SHA-256 operation.
- ATCA_STATUS atcab_sha_hmac_finish (atca_hmac_sha256_ctx_t ∗ctx, uint8_t ∗digest, uint8_t target)
  Executes SHA command to complete a HMAC/SHA-256 operation.
- ATCA_STATUS atcab_sha_hmac (const uint8_t ∗data, size_t data_size, uint16_t key_slot, uint8_t ∗digest, uint8_t target)
  Use the SHA command to compute an HMAC/SHA-256 operation.

15.26.1 Detailed Description

CryptoAuthLib Basic API methods for SHA command.

The SHA command Computes a SHA-256 or HMAC/SHA digest for general purpose use by the host system.

Note

List of devices that support this command - ATSHA204A, ATECC108A, ATECC508A, and ATECC608A. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.27 atca_basic_sign.c File Reference

CryptoAuthLib Basic API methods for Sign command.

```
#include "atca_basic.h"
#include "atca_execution.h"
```

Functions

- ATCA_STATUS atcab_sign_base (uint8_t mode, uint16_t key_id, uint8_t ∗signature)
  Executes the Sign command, which generates a signature using the ECDSA algorithm.
- ATCA_STATUS atcab_sign (uint16_t key_id, const uint8_t ∗msg, uint8_t ∗signature)
  Executes Sign command, to sign a 32-byte external message using the private key in the specified slot. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608A device or TempKey for other devices.
- ATCA_STATUS atcab_sign_internal (uint16_t key_id, bool is_invalidate, bool is_full_sn, uint8_t ∗signature)
  Executes Sign command to sign an internally generated message.

15.27.1 Detailed Description

CryptoAuthLib Basic API methods for Sign command.

The Sign command generates a signature using the private key in slot with ECDSA algorithm.

Note

List of devices that support this command - ATECC108A, ATECC508A, and ATECC608A. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

15.28 atca_basic_updateextra.c File Reference

CryptoAuthLib Basic API methods for UpdateExtra command.

```
#include "atca_basic.h"
#include "atca_execution.h"
```

Functions

- ATCA_STATUS atcab_updateextra (uint8_t mode, uint16_t new_value)

  Executes UpdateExtra command to update the values of the two extra bytes within the Configuration zone (bytes 84 and 85).

15.28.1 Detailed Description

CryptoAuthLib Basic API methods for UpdateExtra command.

The UpdateExtra command is used to update the values of the two extra bytes within the Configuration zone after the Configuration zone has been locked.

Note

List of devices that support this command - ATSHA204A, ATECC108A, ATECC508A, and ATECC608A. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

15.29 atca_basic_verify.c File Reference

CryptoAuthLib Basic API methods for Verify command.

```
#include "atca_basic.h"
#include "atca_execution.h"
#include "host/atca_host.h"
```

Functions

- ATCA_STATUS atcab_verify (uint8_t mode, uint16_t key_id, const uint8_t ∗signature, const uint8_t ∗public↩_key, const uint8_t ∗other_data, uint8_t ∗mac)

  Executes the Verify command, which takes an ECDSA [R,S] signature and verifies that it is correctly generated from a given message and public key. In all cases, the signature is an input to the command.
- ATCA_STATUS atcab_verify_extern (const uint8_t ∗message, const uint8_t ∗signature, const uint8_↩t ∗public_key, bool ∗is_verified)

  Executes the Verify command, which verifies a signature (ECDSA verify operation) with all components (message, signature, and public key) supplied. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608A device or TempKey for other devices.
- ATCA_STATUS atcab_verify_extern_mac (const uint8_t ∗message, const uint8_t ∗signature, const uint8_t ∗public_key, const uint8_t ∗num_in, const uint8_t ∗io_key, bool ∗is_verified)

  Executes the Verify command with verification MAC, which verifies a signature (ECDSA verify operation) with all components (message, signature, and public key) supplied. This function is only available on the ATECC608A.
- ATCA_STATUS atcab_verify_stored (const uint8_t ∗message, const uint8_t ∗signature, uint16_t key_id, bool ∗is_verified)

  Executes the Verify command, which verifies a signature (ECDSA verify operation) with a public key stored in the device. The message to be signed will be loaded into the Message Digest Buffer to the ATECC608A device or TempKey for other devices.
- ATCA_STATUS atcab_verify_stored_mac (const uint8_t ∗message, const uint8_t ∗signature, uint16_t key↩_id, const uint8_t ∗num_in, const uint8_t ∗io_key, bool ∗is_verified)

  Executes the Verify command with verification MAC, which verifies a signature (ECDSA verify operation) with a public key stored in the device. This function is only available on the ATECC608A.
- ATCA_STATUS atcab_verify_validate (uint16_t key_id, const uint8_t ∗signature, const uint8_t ∗other_data, bool ∗is_verified)

  Executes the Verify command in Validate mode to validate a public key stored in a slot.
- ATCA_STATUS atcab_verify_invalidate (uint16_t key_id, const uint8_t ∗signature, const uint8_t ∗other_data, bool ∗is_verified)

  Executes the Verify command in Invalidate mode which invalidates a previously validated public key stored in a slot.

15.29.1 Detailed Description

CryptoAuthLib Basic API methods for Verify command.

The Verify command takes an ECDSA [R,S] signature and verifies that it is correctly generated given an input message digest and public key.

Note

List of devices that support this command - ATECC108A, ATECC508A, and ATECC608A. There are differences in the modes that they support. Refer to device datasheet for full details.

Copyright

15.30 atca_basic_write.c File Reference

CryptoAuthLib Basic API methods for Write command.

```
#include "atca_basic.h"
#include "atca_execution.h"
#include "host/atca_host.h"
```

Functions

- ATCA_STATUS atcab_write (uint8_t zone, uint16_t address, const uint8_t ∗value, const uint8_t ∗mac)

    Executes the Write command, which writes either one four byte word or a 32-byte block to one of the EEPROM zones on the device. Depending upon the value of the WriteConfig byte for this slot, the data may be required to be encrypted by the system prior to being sent to the device. This command cannot be used to write slots configured as ECC private keys.
- ATCA_STATUS atcab_write_zone (uint8_t zone, uint16_t slot, uint8_t block, uint8_t offset, const uint8_t ∗data, uint8_t len)

    Executes the Write command, which writes either 4 or 32 bytes of data into a device zone.
- ATCA_STATUS atcab_write_enc (uint16_t key_id, uint8_t block, const uint8_t ∗data, const uint8_t ∗enc_key, const uint16_t enc_key_id)

    Executes the Write command, which performs an encrypted write of a 32 byte block into given slot.
- ATCA_STATUS atcab_write_config_zone (const uint8_t ∗config_data)

    Executes the Write command, which writes the configuration zone.
- ATCA_STATUS atcab_write_pubkey (uint16_t slot, const uint8_t ∗public_key)

    Uses the write command to write a public key to a slot in the proper format.
- ATCA_STATUS atcab_write_bytes_zone (uint8_t zone, uint16_t slot, size_t offset_bytes, const uint8_t ∗data, size_t length)

    Executes the Write command, which writes data into the configuration, otp, or data zones with a given byte offset and length. Offset and length must be multiples of a word (4 bytes).
- ATCA_STATUS atcab_write_config_counter (uint16_t counter_id, uint32_t counter_value)

    Initialize one of the monotonic counters in device with a specific value.

15.30.1 Detailed Description

CryptoAuthLib Basic API methods for Write command.

The Write command writes either one 4-byte word or a 32-byte block to one of the EEPROM zones on the device. Depending upon the value of the WriteConfig byte for a slot, the data may be required to be encrypted by the system prior to being sent to the device

Note

List of devices that support this command - ATSHA204A, ATECC108A, ATECC508A, and ATECC608A. There are differences in the modes that they support. Refer to device datasheets for full details.

Copyright

15.31 atca_bool.h File Reference

bool define for systems that don't have it

```
#include <stdbool.h>
```

15.31.1 Detailed Description

bool define for systems that don't have it

Copyright

15.32 atca_cfgs.c File Reference

a set of default configurations for various ATCA devices and interfaces

```
#include <stddef.h>
#include "atca_cfgs.h"
#include "atca_iface.h"
#include "atca_device.h"
```

Variables

- ATCAIfaceCfg cfg_ateccx08a_i2c_default

  default configuration for an ECCx08A device
- ATCAIfaceCfg cfg_ateccx08a_swi_default

  default configuration for an ECCx08A device on the logical SWI bus over UART
- ATCAIfaceCfg cfg_ateccx08a_kitcdc_default

  default configuration for Kit protocol over the device's async interface
- ATCAIfaceCfg cfg_ateccx08a_kithid_default

  default configuration for Kit protocol over the device's async interface
- ATCAIfaceCfg cfg_atsha204a_i2c_default

  default configuration for a SHA204A device on the first logical I2C bus
- ATCAIfaceCfg cfg_atsha204a_swi_default

  default configuration for an SHA204A device on the logical SWI bus over UART
- ATCAIfaceCfg cfg_atsha204a_kitcdc_default

  default configuration for Kit protocol over the device's async interface
- ATCAIfaceCfg cfg_atsha204a_kithid_default

  default configuration for Kit protocol over the device's async interface

15.32.1 Detailed Description

a set of default configurations for various ATCA devices and interfaces

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.33 atca_cfgs.h File Reference

a set of default configurations for various ATCA devices and interfaces

```
#include "atca_iface.h"
```

Variables

- ATCAIfaceCfg cfg_ateccx08a_i2c_default
  default configuration for an ECCx08A device on the first logical I2C bus
- ATCAIfaceCfg cfg_ateccx08a_swi_default
  default configuration for an ECCx08A device on the logical SWI bus over UART
- ATCAIfaceCfg cfg_ateccx08a_kitcdc_default
  default configuration for Kit protocol over a CDC interface
- ATCAIfaceCfg cfg_ateccx08a_kithid_default
  default configuration for Kit protocol over a HID interface
- ATCAIfaceCfg cfg_atsha204a_i2c_default
  default configuration for a SHA204A device on the first logical I2C bus
- ATCAIfaceCfg cfg_atsha204a_swi_default
  default configuration for an SHA204A device on the logical SWI bus over UART
- ATCAIfaceCfg cfg_atsha204a_kitcdc_default
  default configuration for Kit protocol over a CDC interface
- ATCAIfaceCfg cfg_atsha204a_kithid_default
  default configuration for Kit protocol over a HID interface for SHA204

15.33.1 Detailed Description

a set of default configurations for various ATCA devices and interfaces

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.34 atca_command.c File Reference

Microchip CryptoAuthentication device command builder - this is the main object that builds the command byte strings for the given device. It does not execute the command. The basic flow is to call a command method to build the command you want given the parameters and then send that byte string through the device interface.

```
#include <stdlib.h>
#include <string.h>
#include "atca_command.h"
#include "atca_devtypes.h"
```

Functions

- ATCA_STATUS atCheckMAC (ATCACommand ca_cmd, ATCAPacket ∗packet)
  ATCACommand CheckMAC method.
- ATCA_STATUS atCounter (ATCACommand ca_cmd, ATCAPacket ∗packet)
  ATCACommand Counter method.
- ATCA_STATUS atDeriveKey (ATCACommand ca_cmd, ATCAPacket ∗packet, bool has_mac)
  ATCACommand DeriveKey method.
- ATCA_STATUS atECDH (ATCACommand ca_cmd, ATCAPacket ∗packet)
  ATCACommand ECDH method.
- ATCA_STATUS atGenDig (ATCACommand ca_cmd, ATCAPacket ∗packet, bool is_no_mac_key)
  ATCACommand Generate Digest method.
- ATCA_STATUS atGenKey (ATCACommand ca_cmd, ATCAPacket ∗packet)
  ATCACommand Generate Key method.
- ATCA_STATUS atHMAC (ATCACommand ca_cmd, ATCAPacket ∗packet)
  ATCACommand HMAC method.
- ATCA_STATUS atInfo (ATCACommand ca_cmd, ATCAPacket ∗packet)
  ATCACommand Info method.
- ATCA_STATUS atLock (ATCACommand ca_cmd, ATCAPacket ∗packet)
  ATCACommand Lock method.
- ATCA_STATUS atMAC (ATCACommand ca_cmd, ATCAPacket ∗packet)
  ATCACommand MAC method.
- ATCA_STATUS atNonce (ATCACommand ca_cmd, ATCAPacket ∗packet)
  ATCACommand Nonce method.
- ATCA_STATUS atPause (ATCACommand ca_cmd, ATCAPacket ∗packet)
  ATCACommand Pause method.
- ATCA_STATUS atPrivWrite (ATCACommand ca_cmd, ATCAPacket ∗packet)
  ATCACommand PrivWrite method.
- ATCA_STATUS atRandom (ATCACommand ca_cmd, ATCAPacket ∗packet)
  ATCACommand Random method.
- ATCA_STATUS atRead (ATCACommand ca_cmd, ATCAPacket ∗packet)
  ATCACommand Read method.
- ATCA_STATUS atSecureBoot (ATCACommand ca_cmd, ATCAPacket ∗packet)
  ATCACommand SecureBoot method.
- ATCA_STATUS atSHA (ATCACommand ca_cmd, ATCAPacket ∗packet, uint16_t write_context_size)
  ATCACommand SHA method.
- ATCA_STATUS atSign (ATCACommand ca_cmd, ATCAPacket ∗packet)
  ATCACommand Sign method.
- ATCA_STATUS atUpdateExtra (ATCACommand ca_cmd, ATCAPacket ∗packet)

ATCACommand UpdateExtra method.

- ATCA_STATUS atVerify (ATCACommand ca_cmd, ATCAPacket *packet)

  ATCACommand ECDSA Verify method.
- ATCA_STATUS atWrite (ATCACommand ca_cmd, ATCAPacket *packet, bool has_mac)

  ATCACommand Write method.
- ATCA_STATUS atAES (ATCACommand ca_cmd, ATCAPacket *packet)

  ATCACommand AES method.
- ATCA_STATUS atSelfTest (ATCACommand ca_cmd, ATCAPacket *packet)

  ATCACommand AES method.
- ATCA_STATUS atKDF (ATCACommand ca_cmd, ATCAPacket *packet)

  ATCACommand KDF method.
- ATCA_STATUS initATCACommand (ATCADeviceType device_type, ATCACommand ca_cmd)

  Initializer for ATCACommand.
- ATCACommand newATCACommand (ATCADeviceType device_type)

  constructor for ATCACommand
- void deleteATCACommand (ATCACommand *ca_cmd)

  ATCACommand destructor.
- void atCRC (size_t length, const uint8_t *data, uint8_t *crc_le)

  Calculates CRC over the given raw data and returns the CRC in little-endian byte order.
- void atCalcCrc (ATCAPacket *packet)

  This function calculates CRC and adds it to the correct offset in the packet data.
- ATCA_STATUS atCheckCrc (const uint8_t *response)

  This function checks the consistency of a response.
- bool atIsSHAFamily (ATCADeviceType device_type)

  determines if a given device type is a SHA device or a superset of a SHA device
- bool atIsECCFamily (ATCADeviceType device_type)

  determines if a given device type is an ECC device or a superset of a ECC device
- ATCA_STATUS isATCAError (uint8_t *data)

  checks for basic error frame in data

15.34.1 Detailed Description

Microchip CryptoAuthentication device command builder - this is the main object that builds the command byte strings for the given device. It does not execute the command. The basic flow is to call a command method to build the command you want given the parameters and then send that byte string through the device interface.

The primary goal of the command builder is to wrap the given parameters with the correct packet size and CRC. The caller should first fill in the parameters required in the ATCAPacket parameter given to the command. The command builder will deal with the mechanics of creating a valid packet using the parameter information.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.35 atca_command.h File Reference

Microchip Crypto Auth device command object - this is a command builder only, it does not send the command. The result of a command method is a fully formed packet, ready to send to the ATCAIFace object to dispatch.

```
#include "atca_compiler.h"
#include "atca_status.h"
#include "atca_devtypes.h"
#include <stddef.h>
```

Data Structures

- struct atca_command

  atca_command is the C object backing ATCACommand.
- struct ATCAPacket

Macros

- #define ATCA_CMD_SIZE_MIN ((uint8_t)7)

  minimum number of bytes in command (from count byte to second CRC byte)
- #define ATCA_CMD_SIZE_MAX ((uint8_t)4 ∗ 36 + 7)

  maximum size of command packet (Verify)
- #define CMD_STATUS_SUCCESS ((uint8_t)0x00)

  status byte for success
- #define CMD_STATUS_WAKEUP ((uint8_t)0x11)

  status byte after wake-up
- #define CMD_STATUS_BYTE_PARSE ((uint8_t)0x03)

  command parse error
- #define CMD_STATUS_BYTE_ECC ((uint8_t)0x05)

  command ECC error
- #define CMD_STATUS_BYTE_EXEC ((uint8_t)0x0F)

  command execution error
- #define CMD_STATUS_BYTE_COMM ((uint8_t)0xFF)

  communication error

Opcodes for Crypto Authentication device commands

- #define ATCA_CHECKMAC ((uint8_t)0x28)

  CheckMac command op-code.
- #define ATCA_DERIVE_KEY ((uint8_t)0x1C)

  DeriveKey command op-code.
- #define ATCA_INFO ((uint8_t)0x30)

  Info command op-code.
- #define ATCA_GENDIG ((uint8_t)0x15)

  GenDig command op-code.
- #define ATCA_GENKEY ((uint8_t)0x40)

  GenKey command op-code.
- #define ATCA_HMAC ((uint8_t)0x11)

  HMAC command op-code.
- #define ATCA_LOCK ((uint8_t)0x17)

  Lock command op-code.
- #define ATCA_MAC ((uint8_t)0x08)

  MAC command op-code.
- #define ATCA_NONCE ((uint8_t)0x16)

  Nonce command op-code.
- #define ATCA_PAUSE ((uint8_t)0x01)

  Pause command op-code.
- #define ATCA_PRIVWRITE ((uint8_t)0x46)

  PrivWrite command op-code.
- #define ATCA_RANDOM ((uint8_t)0x1B)

  Random command op-code.
- #define ATCA_READ ((uint8_t)0x02)

  Read command op-code.
- #define ATCA_SIGN ((uint8_t)0x41)

Sign command op-code.
- #define ATCA_UPDATE_EXTRA ((uint8_t)0x20)
  UpdateExtra command op-code.
- #define ATCA_VERIFY ((uint8_t)0x45)
  GenKey command op-code.
- #define ATCA_WRITE ((uint8_t)0x12)
  Write command op-code.
- #define ATCA_ECDH ((uint8_t)0x43)
  ECDH command op-code.
- #define ATCA_COUNTER ((uint8_t)0x24)
  Counter command op-code.
- #define ATCA_SHA ((uint8_t)0x47)
  SHA command op-code.
- #define ATCA_AES ((uint8_t)0x51)
  AES command op-code.
- #define ATCA_KDF ((uint8_t)0x56)
  KDF command op-code.
- #define ATCA_SECUREBOOT ((uint8_t)0x80)
  Secure Boot command op-code.
- #define ATCA_SELFTEST ((uint8_t)0x77)
  Self test command op-code.

Definitions of Data and Packet Sizes

- #define ATCA_BLOCK_SIZE (32)
  size of a block
- #define ATCA_WORD_SIZE (4)
  size of a word
- #define ATCA_PUB_KEY_PAD (4)
  size of the public key pad
- #define ATCA_SERIAL_NUM_SIZE (9)
  number of bytes in the device serial number
- #define ATCA_RSP_SIZE_VAL ((uint8_t)7)
  size of response packet containing four bytes of data
- #define ATCA_KEY_COUNT (16)
  number of keys
- #define ATCA_ECC_CONFIG_SIZE (128)
  size of configuration zone
- #define ATCA_SHA_CONFIG_SIZE (88)
  size of configuration zone
- #define ATCA_OTP_SIZE (64)
  size of OTP zone
- #define ATCA_DATA_SIZE (ATCA_KEY_COUNT * ATCA_KEY_SIZE)
  size of data zone
- #define ATCA_AES_GFM_SIZE ATCA_BLOCK_SIZE
  size of GFM data
- #define ATCA_CHIPMODE_OFFSET (19)
  ChipMode byte offset within the configuration zone.
- #define ATCA_CHIPMODE_I2C_ADDRESS_FLAG ((uint8_t)0x01)
  ChipMode I2C Address in UserExtraAdd flag.
- #define ATCA_CHIPMODE_TTL_ENABLE_FLAG ((uint8_t)0x02)
  ChipMode TTLenable flag.
- #define ATCA_CHIPMODE_WATCHDOG_MASK ((uint8_t)0x04)
  ChipMode watchdog duration mask.
- #define ATCA_CHIPMODE_WATCHDOG_SHORT ((uint8_t)0x00)
  ChipMode short watchdog (∼1.3s)
- #define ATCA_CHIPMODE_WATCHDOG_LONG ((uint8_t)0x04)
  ChipMode long watchdog (∼13s)

- #define ATCA_CHIPMODE_CLOCK_DIV_MASK ((uint8_t)0xF8)
  ChipMode clock divider mask.
- #define ATCA_CHIPMODE_CLOCK_DIV_M0 ((uint8_t)0x00)
  ChipMode clock divider M0.
- #define ATCA_CHIPMODE_CLOCK_DIV_M1 ((uint8_t)0x28)
  ChipMode clock divider M1.
- #define ATCA_CHIPMODE_CLOCK_DIV_M2 ((uint8_t)0x68)
  ChipMode clock divider M2.
- #define ATCA_COUNT_SIZE ((uint8_t)1)
  Number of bytes in the command packet Count.
- #define ATCA_CRC_SIZE ((uint8_t)2)
  Number of bytes in the command packet CRC.
- #define ATCA_PACKET_OVERHEAD (ATCA_COUNT_SIZE + ATCA_CRC_SIZE)
  Number of bytes in the command packet.
- #define ATCA_PUB_KEY_SIZE (64)
  size of a p256 public key
- #define ATCA_PRIV_KEY_SIZE (32)
  size of a p256 private key
- #define ATCA_SIG_SIZE (64)
  size of a p256 signature
- #define ATCA_KEY_SIZE (32)
  size of a symmetric SHA key
- #define RSA2048_KEY_SIZE (256)
  size of a RSA private key
- #define ATCA_RSP_SIZE_MIN ((uint8_t)4)
  minimum number of bytes in response
- #define ATCA_RSP_SIZE_4 ((uint8_t)7)
  size of response packet containing 4 bytes data
- #define ATCA_RSP_SIZE_72 ((uint8_t)75)
  size of response packet containing 64 bytes data
- #define ATCA_RSP_SIZE_64 ((uint8_t)67)
  size of response packet containing 64 bytes data
- #define ATCA_RSP_SIZE_32 ((uint8_t)35)
  size of response packet containing 32 bytes data
- #define ATCA_RSP_SIZE_16 ((uint8_t)19)
  size of response packet containing 16 bytes data
- #define ATCA_RSP_SIZE_MAX ((uint8_t)75)
  maximum size of response packet (GenKey and Verify command)
- #define OUTNONCE_SIZE (32)
  Size of the OutNonce response expected from several commands.

Definitions for Command Parameter Ranges

- #define ATCA_KEY_ID_MAX ((uint8_t)15)
  maximum value for key id
- #define ATCA_OTP_BLOCK_MAX ((uint8_t)1)
  maximum value for OTP block

Definitions for Indexes Common to All Commands

- #define ATCA_COUNT_IDX (0)
  command packet index for count
- #define ATCA_OPCODE_IDX (1)
  command packet index for op-code
- #define ATCA_PARAM1_IDX (2)
  command packet index for first parameter
- #define ATCA_PARAM2_IDX (3)
  command packet index for second parameter

- #define ATCA_DATA_IDX (5)
  command packet index for data load
- #define ATCA_RSP_DATA_IDX (1)
  buffer index of data in response

Definitions for Zone and Address Parameters

- #define ATCA_ZONE_CONFIG ((uint8_t)0x00)
  Configuration zone.
- #define ATCA_ZONE_OTP ((uint8_t)0x01)
  OTP (One Time Programming) zone.
- #define ATCA_ZONE_DATA ((uint8_t)0x02)
  Data zone.
- #define ATCA_ZONE_MASK ((uint8_t)0x03)
  Zone mask.
- #define ATCA_ZONE_ENCRYPTED ((uint8_t)0x40)
  Zone bit 6 set: Write is encrypted with an unlocked data zone.
- #define ATCA_ZONE_READWRITE_32 ((uint8_t)0x80)
  Zone bit 7 set: Access 32 bytes, otherwise 4 bytes.
- #define ATCA_ADDRESS_MASK_CONFIG (0x001F)
  Address bits 5 to 7 are 0 for Configuration zone.
- #define ATCA_ADDRESS_MASK_OTP (0x000F)
  Address bits 4 to 7 are 0 for OTP zone.
- #define ATCA_ADDRESS_MASK (0x007F)
  Address bit 7 to 15 are always 0.
- #define ATCA_TEMPKEY_KEYID (0xFFFF)
  KeyID when referencing TempKey.

Definitions for Key types

- #define ATCA_B283_KEY_TYPE 0
  B283 NIST ECC key.
- #define ATCA_K283_KEY_TYPE 1
  K283 NIST ECC key.
- #define ATCA_P256_KEY_TYPE 4
  P256 NIST ECC key.
- #define ATCA_AES_KEY_TYPE 6
  AES-128 Key.
- #define ATCA_SHA_KEY_TYPE 7
  SHA key or other data.

Definitions for the AES Command

- #define AES_MODE_IDX ATCA_PARAM1_IDX
  AES command index for mode.
- #define AES_KEYID_IDX ATCA_PARAM2_IDX
  AES command index for key id.
- #define AES_INPUT_IDX ATCA_DATA_IDX
  AES command index for input data.
- #define AES_COUNT (23)
  AES command packet size.
- #define AES_MODE_MASK ((uint8_t)0xC7)
  AES mode bits 3 to 5 are 0.
- #define AES_MODE_KEY_BLOCK_MASK ((uint8_t)0xC0)
  AES mode mask for key block field.
- #define AES_MODE_OP_MASK ((uint8_t)0x07)
  AES mode operation mask.

- #define AES_MODE_ENCRYPT ((uint8_t)0x00)
  AES mode: Encrypt.
- #define AES_MODE_DECRYPT ((uint8_t)0x01)
  AES mode: Decrypt.
- #define AES_MODE_GFM ((uint8_t)0x03)
  AES mode: GFM calculation.
- #define AES_MODE_KEY_BLOCK_POS (6)
  Bit shift for key block in mode.
- #define AES_DATA_SIZE (16)
  size of AES encrypt/decrypt data
- #define AES_RSP_SIZE ATCA_RSP_SIZE_16
  AES command response packet size.

Definitions for the CheckMac Command

- #define CHECKMAC_MODE_IDX ATCA_PARAM1_IDX
  CheckMAC command index for mode.
- #define CHECKMAC_KEYID_IDX ATCA_PARAM2_IDX
  CheckMAC command index for key identifier.
- #define CHECKMAC_CLIENT_CHALLENGE_IDX ATCA_DATA_IDX
  CheckMAC command index for client challenge.
- #define CHECKMAC_CLIENT_RESPONSE_IDX (37)
  CheckMAC command index for client response.
- #define CHECKMAC_DATA_IDX (69)
  CheckMAC command index for other data.
- #define CHECKMAC_COUNT (84)
  CheckMAC command packet size.
- #define CHECKMAC_MODE_CHALLENGE ((uint8_t)0x00)
  CheckMAC mode 0: first SHA block from key id.
- #define CHECKMAC_MODE_BLOCK2_TEMPKEY ((uint8_t)0x01)
  CheckMAC mode bit 0: second SHA block from TempKey.
- #define CHECKMAC_MODE_BLOCK1_TEMPKEY ((uint8_t)0x02)
  CheckMAC mode bit 1: first SHA block from TempKey.
- #define CHECKMAC_MODE_SOURCE_FLAG_MATCH ((uint8_t)0x04)
  CheckMAC mode bit 2: match TempKey.SourceFlag.
- #define CHECKMAC_MODE_INCLUDE_OTP_64 ((uint8_t)0x20)
  CheckMAC mode bit 5: include first 64 OTP bits.
- #define CHECKMAC_MODE_MASK ((uint8_t)0x27)
  CheckMAC mode bits 3, 4, 6, and 7 are 0.
- #define CHECKMAC_CLIENT_CHALLENGE_SIZE (32)
  CheckMAC size of client challenge.
- #define CHECKMAC_CLIENT_RESPONSE_SIZE (32)
  CheckMAC size of client response.
- #define CHECKMAC_OTHER_DATA_SIZE (13)
  CheckMAC size of "other data".
- #define CHECKMAC_CLIENT_COMMAND_SIZE (4)
  CheckMAC size of client command header size inside "other data".
- #define CHECKMAC_CMD_MATCH (0)
  CheckMAC return value when there is a match.
- #define CHECKMAC_CMD_MISMATCH (1)
  CheckMAC return value when there is a mismatch.
- #define CHECKMAC_RSP_SIZE ATCA_RSP_SIZE_MIN
  CheckMAC response packet size.

Definitions for the Counter command

- #define COUNTER_COUNT ATCA_CMD_SIZE_MIN
- #define COUNTER_MODE_IDX ATCA_PARAM1_IDX

Counter command index for mode.
- #define COUNTER_KEYID_IDX ATCA_PARAM2_IDX
  Counter command index for key id.
- #define COUNTER_MODE_MASK ((uint8_t)0x01)
  Counter mode bits 1 to 7 are 0.
- #define COUNTER_MAX_VALUE ((uint32_t)2097151)
  Counter maximum value of the counter.
- #define COUNTER_MODE_READ ((uint8_t)0x00)
  Counter command mode for reading.
- #define COUNTER_MODE_INCREMENT ((uint8_t)0x01)
  Counter command mode for incrementing.
- #define COUNTER_RSP_SIZE ATCA_RSP_SIZE_4
  Counter command response packet size.
- #define COUNTER_SIZE ATCA_RSP_SIZE_MIN
  Counter size in binary.

Definitions for the DeriveKey Command

- #define DERIVE_KEY_RANDOM_IDX ATCA_PARAM1_IDX
  DeriveKey command index for random bit.
- #define DERIVE_KEY_TARGETKEY_IDX ATCA_PARAM2_IDX
  DeriveKey command index for target slot.
- #define DERIVE_KEY_MAC_IDX ATCA_DATA_IDX
  DeriveKey command index for optional MAC.
- #define DERIVE_KEY_COUNT_SMALL ATCA_CMD_SIZE_MIN
  DeriveKey command packet size without MAC.
- #define DERIVE_KEY_MODE ((uint8_t)0x04)
  DeriveKey command mode set to 4 as in datasheet.
- #define DERIVE_KEY_COUNT_LARGE (39)
  DeriveKey command packet size with MAC.
- #define DERIVE_KEY_RANDOM_FLAG ((uint8_t)4)
  DeriveKey 1. parameter; has to match TempKey.SourceFlag.
- #define DERIVE_KEY_MAC_SIZE (32)
  DeriveKey MAC size.
- #define DERIVE_KEY_RSP_SIZE ATCA_RSP_SIZE_MIN
  DeriveKey response packet size.

Definitions for the ECDH Command

- #define ECDH_PREFIX_MODE ((uint8_t)0x00)
- #define ECDH_COUNT (ATCA_CMD_SIZE_MIN + ATCA_PUB_KEY_SIZE)
- #define ECDH_MODE_SOURCE_MASK ((uint8_t)0x01)
- #define ECDH_MODE_SOURCE_EEPROM_SLOT ((uint8_t)0x00)
- #define ECDH_MODE_SOURCE_TEMPKEY ((uint8_t)0x01)
- #define ECDH_MODE_OUTPUT_MASK ((uint8_t)0x02)
- #define ECDH_MODE_OUTPUT_CLEAR ((uint8_t)0x00)
- #define ECDH_MODE_OUTPUT_ENC ((uint8_t)0x02)
- #define ECDH_MODE_COPY_MASK ((uint8_t)0x0C)
- #define ECDH_MODE_COPY_COMPATIBLE ((uint8_t)0x00)
- #define ECDH_MODE_COPY_EEPROM_SLOT ((uint8_t)0x04)
- #define ECDH_MODE_COPY_TEMP_KEY ((uint8_t)0x08)
- #define ECDH_MODE_COPY_OUTPUT_BUFFER ((uint8_t)0x0C)
- #define ECDH_KEY_SIZE ATCA_BLOCK_SIZE
  ECDH output data size.
- #define ECDH_RSP_SIZE ATCA_RSP_SIZE_64
  ECDH command packet size.

Definitions for the GenDig Command

- #define GENDIG_ZONE_IDX ATCA_PARAM1_IDX
  GenDig command index for zone.
- #define GENDIG_KEYID_IDX ATCA_PARAM2_IDX
  GenDig command index for key id.
- #define GENDIG_DATA_IDX ATCA_DATA_IDX
  GenDig command index for optional data.
- #define GENDIG_COUNT ATCA_CMD_SIZE_MIN
  GenDig command packet size without "other data".
- #define GENDIG_ZONE_CONFIG ((uint8_t)0)
  GenDig zone id config. Use KeyID to specify any of the four 256-bit blocks of the Configuration zone.
- #define GENDIG_ZONE_OTP ((uint8_t)1)
  GenDig zone id OTP. Use KeyID to specify either the first or second 256-bit block of the OTP zone.
- #define GENDIG_ZONE_DATA ((uint8_t)2)
  GenDig zone id data. Use KeyID to specify a slot in the Data zone or a transport key in the hardware array.
- #define GENDIG_ZONE_SHARED_NONCE ((uint8_t)3)
  GenDig zone id shared nonce. KeyID specifies the location of the input value in the message generation.
- #define GENDIG_ZONE_COUNTER ((uint8_t)4)
  GenDig zone id counter. KeyID specifies the monotonic counter ID to be included in the message generation.
- #define GENDIG_ZONE_KEY_CONFIG ((uint8_t)5)
  GenDig zone id key config. KeyID specifies the slot for which the configuration information is to be included in the message generation.
- #define GENDIG_RSP_SIZE ATCA_RSP_SIZE_MIN
  GenDig command response packet size.

Definitions for the GenKey Command

- #define GENKEY_MODE_IDX ATCA_PARAM1_IDX
  GenKey command index for mode.
- #define GENKEY_KEYID_IDX ATCA_PARAM2_IDX
  GenKey command index for key id.
- #define GENKEY_DATA_IDX (5)
  GenKey command index for other data.
- #define GENKEY_COUNT ATCA_CMD_SIZE_MIN
  GenKey command packet size without "other data".
- #define GENKEY_COUNT_DATA (10)
  GenKey command packet size with "other data".
- #define GENKEY_OTHER_DATA_SIZE (3)
  GenKey size of "other data".
- #define GENKEY_MODE_MASK ((uint8_t)0x1C)
  GenKey mode bits 0 to 1 and 5 to 7 are 0.
- #define GENKEY_MODE_PRIVATE ((uint8_t)0x04)
  GenKey mode: private key generation.
- #define GENKEY_MODE_PUBLIC ((uint8_t)0x00)
  GenKey mode: public key calculation.
- #define GENKEY_MODE_DIGEST ((uint8_t)0x08)
  GenKey mode: PubKey digest will be created after the public key is calculated.
- #define GENKEY_MODE_PUBKEY_DIGEST ((uint8_t)0x10)
  GenKey mode: Calculate PubKey digest on the public key in KeyId.
- #define GENKEY_PRIVATE_TO_TEMPKEY ((uint16_t)0xFFFF)
  GenKey Create private key and store to tempkey (608 only)
- #define GENKEY_RSP_SIZE_SHORT ATCA_RSP_SIZE_MIN
  GenKey response packet size in Digest mode.
- #define GENKEY_RSP_SIZE_LONG ATCA_RSP_SIZE_64
  GenKey response packet size when returning a public key.

Definitions for the HMAC Command

- #define HMAC_MODE_IDX ATCA_PARAM1_IDX

HMAC command index for mode.
- #define HMAC_KEYID_IDX ATCA_PARAM2_IDX
  HMAC command index for key id.
- #define HMAC_COUNT ATCA_CMD_SIZE_MIN
  HMAC command packet size.
- #define HMAC_MODE_FLAG_TK_RAND ((uint8_t)0x00)
  HMAC mode bit 2: The value of this bit must match the value in TempKey.SourceFlag or the command will return an error.
- #define HMAC_MODE_FLAG_TK_NORAND ((uint8_t)0x04)
  HMAC mode bit 2: The value of this bit must match the value in TempKey.SourceFlag or the command will return an error.
- #define HMAC_MODE_FLAG_OTP88 ((uint8_t)0x10)
  HMAC mode bit 4: Include the first 88 OTP bits (OTP[0] through OTP[10]) in the message.; otherwise, the corresponding message bits are set to zero. Not applicable for ATECC508A.
- #define HMAC_MODE_FLAG_OTP64 ((uint8_t)0x20)
  HMAC mode bit 5: Include the first 64 OTP bits (OTP[0] through OTP[7]) in the message.; otherwise, the corresponding message bits are set to zero. If Mode[4] is set, the value of this mode bit is ignored. Not applicable for ATECC508A.
- #define HMAC_MODE_FLAG_FULLSN ((uint8_t)0x40)
  HMAC mode bit 6: If set, include the 48 bits SN[2:3] and SN[4:7] in the message.; otherwise, the corresponding message bits are set to zero.
- #define HMAC_MODE_MASK ((uint8_t)0x74)
  HMAC mode bits 0, 1, 3, and 7 are 0.
- #define HMAC_DIGEST_SIZE (32)
  HMAC size of digest response.
- #define HMAC_RSP_SIZE ATCA_RSP_SIZE_32
  HMAC command response packet size.

Definitions for the Info Command

- #define INFO_PARAM1_IDX ATCA_PARAM1_IDX
  Info command index for 1. parameter.
- #define INFO_PARAM2_IDX ATCA_PARAM2_IDX
  Info command index for 2. parameter.
- #define INFO_COUNT ATCA_CMD_SIZE_MIN
  Info command packet size.
- #define INFO_MODE_REVISION ((uint8_t)0x00)
  Info mode Revision.
- #define INFO_MODE_KEY_VALID ((uint8_t)0x01)
  Info mode KeyValid.
- #define INFO_MODE_STATE ((uint8_t)0x02)
  Info mode State.
- #define INFO_MODE_GPIO ((uint8_t)0x03)
  Info mode GPIO.
- #define INFO_MODE_VOL_KEY_PERMIT ((uint8_t)0x04)
  Info mode GPIO.
- #define INFO_MODE_MAX ((uint8_t)0x03)
  Info mode maximum value.
- #define INFO_NO_STATE ((uint8_t)0x00)
  Info mode is not the state mode.
- #define INFO_OUTPUT_STATE_MASK ((uint8_t)0x01)
  Info output state mask.
- #define INFO_DRIVER_STATE_MASK ((uint8_t)0x02)
  Info driver state mask.
- #define INFO_PARAM2_SET_LATCH_STATE ((uint16_t)0x0002)
  Info param2 to set the persistent latch state.
- #define INFO_PARAM2_LATCH_SET ((uint16_t)0x0001)
  Info param2 to set the persistent latch.
- #define INFO_PARAM2_LATCH_CLEAR ((uint16_t)0x0000)

Info param2 to clear the persistent latch.
- #define INFO_SIZE ((uint8_t)0x04)

  Info return size.
- #define INFO_RSP_SIZE ATCA_RSP_SIZE_VAL

  Info command response packet size.

Definitions for the KDF Command

- #define KDF_MODE_IDX ATCA_PARAM1_IDX

  KDF command index for mode.
- #define KDF_KEYID_IDX ATCA_PARAM2_IDX

  KDF command index for key id.
- #define KDF_DETAILS_IDX ATCA_DATA_IDX

  KDF command index for details.
- #define KDF_DETAILS_SIZE 4

  KDF details (param3) size.
- #define KDF_MESSAGE_IDX (ATCA_DATA_IDX + KDF_DETAILS_SIZE)
- #define KDF_MODE_SOURCE_MASK ((uint8_t)0x03)

  KDF mode source key mask.
- #define KDF_MODE_SOURCE_TEMPKEY ((uint8_t)0x00)

  KDF mode source key in TempKey.
- #define KDF_MODE_SOURCE_TEMPKEY_UP ((uint8_t)0x01)

  KDF mode source key in upper TempKey.
- #define KDF_MODE_SOURCE_SLOT ((uint8_t)0x02)

  KDF mode source key in a slot.
- #define KDF_MODE_SOURCE_ALTKEYBUF ((uint8_t)0x03)

  KDF mode source key in alternate key buffer.
- #define KDF_MODE_TARGET_MASK ((uint8_t)0x1C)

  KDF mode target key mask.
- #define KDF_MODE_TARGET_TEMPKEY ((uint8_t)0x00)

  KDF mode target key in TempKey.
- #define KDF_MODE_TARGET_TEMPKEY_UP ((uint8_t)0x04)

  KDF mode target key in upper TempKey.
- #define KDF_MODE_TARGET_SLOT ((uint8_t)0x08)

  KDF mode target key in slot.
- #define KDF_MODE_TARGET_ALTKEYBUF ((uint8_t)0x0C)

  KDF mode target key in alternate key buffer.
- #define KDF_MODE_TARGET_OUTPUT ((uint8_t)0x10)

  KDF mode target key in output buffer.
- #define KDF_MODE_TARGET_OUTPUT_ENC ((uint8_t)0x14)

  KDF mode target key encrypted in output buffer.
- #define KDF_MODE_ALG_MASK ((uint8_t)0x60)

  KDF mode algorithm mask.
- #define KDF_MODE_ALG_PRF ((uint8_t)0x00)

  KDF mode PRF algorithm.
- #define KDF_MODE_ALG_AES ((uint8_t)0x20)

  KDF mode AES algorithm.
- #define KDF_MODE_ALG_HKDF ((uint8_t)0x40)

  KDF mode HKDF algorithm.
- #define KDF_DETAILS_PRF_KEY_LEN_MASK ((uint32_t)0x00000003)

  KDF details for PRF, source key length mask.
- #define KDF_DETAILS_PRF_KEY_LEN_16 ((uint32_t)0x00000000)

  KDF details for PRF, source key length is 16 bytes.
- #define KDF_DETAILS_PRF_KEY_LEN_32 ((uint32_t)0x00000001)

  KDF details for PRF, source key length is 32 bytes.
- #define KDF_DETAILS_PRF_KEY_LEN_48 ((uint32_t)0x00000002)

  KDF details for PRF, source key length is 48 bytes.
- #define KDF_DETAILS_PRF_KEY_LEN_64 ((uint32_t)0x00000003)

KDF details for PRF, source key length is 64 bytes.
- #define KDF_DETAILS_PRF_TARGET_LEN_MASK ((uint32_t)0x00000100)

  KDF details for PRF, target length mask.
- #define KDF_DETAILS_PRF_TARGET_LEN_32 ((uint32_t)0x00000000)

  KDF details for PRF, target length is 32 bytes.
- #define KDF_DETAILS_PRF_TARGET_LEN_64 ((uint32_t)0x00000100)

  KDF details for PRF, target length is 64 bytes.
- #define KDF_DETAILS_PRF_AEAD_MASK ((uint32_t)0x00000600)

  KDF details for PRF, AEAD processing mask.
- #define KDF_DETAILS_PRF_AEAD_MODE0 ((uint32_t)0x00000000)

  KDF details for PRF, AEAD no processing.
- #define KDF_DETAILS_PRF_AEAD_MODE1 ((uint32_t)0x00000200)

  KDF details for PRF, AEAD First 32 go to target, second 32 go to output buffer.
- #define KDF_DETAILS_AES_KEY_LOC_MASK ((uint32_t)0x00000003)

  KDF details for AES, key location mask.
- #define KDF_DETAILS_HKDF_MSG_LOC_MASK ((uint32_t)0x00000003)

  KDF details for HKDF, message location mask.
- #define KDF_DETAILS_HKDF_MSG_LOC_SLOT ((uint32_t)0x00000000)

  KDF details for HKDF, message location in slot.
- #define KDF_DETAILS_HKDF_MSG_LOC_TEMPKEY ((uint32_t)0x00000001)

  KDF details for HKDF, message location in TempKey.
- #define KDF_DETAILS_HKDF_MSG_LOC_INPUT ((uint32_t)0x00000002)

  KDF details for HKDF, message location in input parameter.
- #define KDF_DETAILS_HKDF_MSG_LOC_IV ((uint32_t)0x00000003)

  KDF details for HKDF, message location is a special IV function.
- #define KDF_DETAILS_HKDF_ZERO_KEY ((uint32_t)0x00000004)

  KDF details for HKDF, key is 32 bytes of zero.

Definitions for the Lock Command

- #define LOCK_ZONE_IDX ATCA_PARAM1_IDX

  Lock command index for zone.
- #define LOCK_SUMMARY_IDX ATCA_PARAM2_IDX

  Lock command index for summary.
- #define LOCK_COUNT ATCA_CMD_SIZE_MIN

  Lock command packet size.
- #define LOCK_ZONE_CONFIG ((uint8_t)0x00)

  Lock zone is Config.
- #define LOCK_ZONE_DATA ((uint8_t)0x01)

  Lock zone is OTP or Data.
- #define LOCK_ZONE_DATA_SLOT ((uint8_t)0x02)

  Lock slot of Data.
- #define LOCK_ZONE_NO_CRC ((uint8_t)0x80)

  Lock command: Ignore summary.
- #define LOCK_ZONE_MASK (0xBF)

  Lock parameter 1 bits 6 are 0.
- #define ATCA_UNLOCKED (0x55)

  Value indicating an unlocked zone.
- #define ATCA_LOCKED (0x00)

  Value indicating a locked zone.
- #define LOCK_RSP_SIZE ATCA_RSP_SIZE_MIN

  Lock command response packet size.

Definitions for the MAC Command

- #define MAC_MODE_IDX ATCA_PARAM1_IDX

  MAC command index for mode.
- #define MAC_KEYID_IDX ATCA_PARAM2_IDX

MAC command index for key id.
- #define MAC_CHALLENGE_IDX ATCA_DATA_IDX
  MAC command index for optional challenge.
- #define MAC_COUNT_SHORT ATCA_CMD_SIZE_MIN
  MAC command packet size without challenge.
- #define MAC_COUNT_LONG (39)
  MAC command packet size with challenge.
- #define MAC_MODE_CHALLENGE ((uint8_t)0x00)
  MAC mode 0: first SHA block from data slot.
- #define MAC_MODE_BLOCK2_TEMPKEY ((uint8_t)0x01)
  MAC mode bit 0: second SHA block from TempKey.
- #define MAC_MODE_BLOCK1_TEMPKEY ((uint8_t)0x02)
  MAC mode bit 1: first SHA block from TempKey.
- #define MAC_MODE_SOURCE_FLAG_MATCH ((uint8_t)0x04)
  MAC mode bit 2: match TempKey.SourceFlag.
- #define MAC_MODE_PTNONCE_TEMPKEY ((uint8_t)0x06)
  MAC mode bit 0: second SHA block from TempKey.
- #define MAC_MODE_PASSTHROUGH ((uint8_t)0x07)
  MAC mode bit 0-2: pass-through mode.
- #define MAC_MODE_INCLUDE_OTP_88 ((uint8_t)0x10)
  MAC mode bit 4: include first 88 OTP bits.
- #define MAC_MODE_INCLUDE_OTP_64 ((uint8_t)0x20)
  MAC mode bit 5: include first 64 OTP bits.
- #define MAC_MODE_INCLUDE_SN ((uint8_t)0x40)
  MAC mode bit 6: include serial number.
- #define MAC_CHALLENGE_SIZE (32)
  MAC size of challenge.
- #define MAC_SIZE (32)
  MAC size of response.
- #define MAC_MODE_MASK ((uint8_t)0x77)
  MAC mode bits 3 and 7 are 0.
- #define MAC_RSP_SIZE ATCA_RSP_SIZE_32
  MAC command response packet size.

Definitions for the Nonce Command

- #define NONCE_MODE_IDX ATCA_PARAM1_IDX
  Nonce command index for mode.
- #define NONCE_PARAM2_IDX ATCA_PARAM2_IDX
  Nonce command index for 2. parameter.
- #define NONCE_INPUT_IDX ATCA_DATA_IDX
  Nonce command index for input data.
- #define NONCE_COUNT_SHORT (ATCA_CMD_SIZE_MIN + 20)
  Nonce command packet size for 20 bytes of NumIn.
- #define NONCE_COUNT_LONG (ATCA_CMD_SIZE_MIN + 32)
  Nonce command packet size for 32 bytes of NumIn.
- #define NONCE_COUNT_LONG_64 (ATCA_CMD_SIZE_MIN + 64)
  Nonce command packet size for 64 bytes of NumIn.
- #define NONCE_MODE_MASK ((uint8_t)0x03)
  Nonce mode bits 2 to 7 are 0.
- #define NONCE_MODE_SEED_UPDATE ((uint8_t)0x00)
  Nonce mode: update seed.
- #define NONCE_MODE_NO_SEED_UPDATE ((uint8_t)0x01)
  Nonce mode: do not update seed.
- #define NONCE_MODE_INVALID ((uint8_t)0x02)
  Nonce mode 2 is invalid.
- #define NONCE_MODE_PASSTHROUGH ((uint8_t)0x03)
  Nonce mode: pass-through.

- #define NONCE_MODE_INPUT_LEN_MASK ((uint8_t)0x20)
  Nonce mode: input size mask.
- #define NONCE_MODE_INPUT_LEN_32 ((uint8_t)0x00)
  Nonce mode: input size is 32 bytes.
- #define NONCE_MODE_INPUT_LEN_64 ((uint8_t)0x20)
  Nonce mode: input size is 64 bytes.
- #define NONCE_MODE_TARGET_MASK ((uint8_t)0xC0)
  Nonce mode: target mask.
- #define NONCE_MODE_TARGET_TEMPKEY ((uint8_t)0x00)
  Nonce mode: target is TempKey.
- #define NONCE_MODE_TARGET_MSGDIGBUF ((uint8_t)0x40)
  Nonce mode: target is Message Digest Buffer.
- #define NONCE_MODE_TARGET_ALTKEYBUF ((uint8_t)0x80)
  Nonce mode: target is Alternate Key Buffer.
- #define NONCE_ZERO_CALC_MASK ((uint16_t)0x8000)
  Nonce zero (param2): calculation mode mask.
- #define NONCE_ZERO_CALC_RANDOM ((uint16_t)0x0000)
  Nonce zero (param2): calculation mode random, use RNG in calculation and return RNG output.
- #define NONCE_ZERO_CALC_TEMPKEY ((uint16_t)0x8000)
  Nonce zero (param2): calculation mode TempKey, use TempKey in calculation and return new TempKey value.
- #define NONCE_NUMIN_SIZE (20)
  Nonce NumIn size for random modes.
- #define NONCE_NUMIN_SIZE_PASSTHROUGH (32)
  Nonce NumIn size for 32-byte pass-through mode.
- #define NONCE_RSP_SIZE_SHORT ATCA_RSP_SIZE_MIN
  Nonce command response packet size with no output.
- #define NONCE_RSP_SIZE_LONG ATCA_RSP_SIZE_32
  Nonce command response packet size with output.

Definitions for the Pause Command

- #define PAUSE_SELECT_IDX ATCA_PARAM1_IDX
  Pause command index for Selector.
- #define PAUSE_PARAM2_IDX ATCA_PARAM2_IDX
  Pause command index for 2. parameter.
- #define PAUSE_COUNT ATCA_CMD_SIZE_MIN
  Pause command packet size.
- #define PAUSE_RSP_SIZE ATCA_RSP_SIZE_MIN
  Pause command response packet size.

Definitions for the PrivWrite Command

- #define PRIVWRITE_ZONE_IDX ATCA_PARAM1_IDX
  PrivWrite command index for zone.
- #define PRIVWRITE_KEYID_IDX ATCA_PARAM2_IDX
  PrivWrite command index for KeyID.
- #define PRIVWRITE_VALUE_IDX (5)
  PrivWrite command index for value.
- #define PRIVWRITE_MAC_IDX (41)
  PrivWrite command index for MAC.
- #define PRIVWRITE_COUNT (75)
  PrivWrite command packet size.
- #define PRIVWRITE_ZONE_MASK ((uint8_t)0x40)
  PrivWrite zone bits 0 to 5 and 7 are 0.
- #define PRIVWRITE_MODE_ENCRYPT ((uint8_t)0x40)
  PrivWrite mode: encrypted.
- #define PRIVWRITE_RSP_SIZE ATCA_RSP_SIZE_MIN
  PrivWrite command response packet size.

Definitions for the Random Command

- #define RANDOM_MODE_IDX ATCA_PARAM1_IDX
  Random command index for mode.
- #define RANDOM_PARAM2_IDX ATCA_PARAM2_IDX
  Random command index for 2. parameter.
- #define RANDOM_COUNT ATCA_CMD_SIZE_MIN
  Random command packet size.
- #define RANDOM_SEED_UPDATE ((uint8_t)0x00)
  Random mode for automatic seed update.
- #define RANDOM_NO_SEED_UPDATE ((uint8_t)0x01)
  Random mode for no seed update.
- #define RANDOM_NUM_SIZE ((uint8_t)32)
  Number of bytes in the data packet of a random command.
- #define RANDOM_RSP_SIZE ATCA_RSP_SIZE_32
  Random command response packet size.

Definitions for the Read Command

- #define READ_ZONE_IDX ATCA_PARAM1_IDX
  Read command index for zone.
- #define READ_ADDR_IDX ATCA_PARAM2_IDX
  Read command index for address.
- #define READ_COUNT ATCA_CMD_SIZE_MIN
  Read command packet size.
- #define READ_ZONE_MASK ((uint8_t)0x83)
  Read zone bits 2 to 6 are 0.
- #define READ_4_RSP_SIZE ATCA_RSP_SIZE_VAL
  Read command response packet size when reading 4 bytes.
- #define READ_32_RSP_SIZE ATCA_RSP_SIZE_32
  Read command response packet size when reading 32 bytes.

Definitions for the SecureBoot Command

- #define SECUREBOOT_MODE_IDX ATCA_PARAM1_IDX
  SecureBoot command index for mode.
- #define SECUREBOOT_DIGEST_SIZE (32)
  SecureBoot digest input size.
- #define SECUREBOOT_SIGNATURE_SIZE (64)
  SecureBoot signature input size.
- #define SECUREBOOT_COUNT_DIG (ATCA_CMD_SIZE_MIN + SECUREBOOT_DIGEST_SIZE)
  SecureBoot command packet size for just a digest.
- #define SECUREBOOT_COUNT_DIG_SIG (ATCA_CMD_SIZE_MIN + SECUREBOOT_DIGEST_SIZE + SECUREBOOT_SIGNATURE_SIZE)
  SecureBoot command packet size for a digest and signature.
- #define SECUREBOOT_MAC_SIZE (32)
  SecureBoot MAC output size.
- #define SECUREBOOT_RSP_SIZE_NO_MAC ATCA_RSP_SIZE_MIN
  SecureBoot response packet size for no MAC.
- #define SECUREBOOT_RSP_SIZE_MAC (ATCA_PACKET_OVERHEAD + SECUREBOOT_MAC_SIZE)
  SecureBoot response packet size with MAC.
- #define SECUREBOOT_MODE_MASK ((uint8_t)0x07)
  SecureBoot mode mask.
- #define SECUREBOOT_MODE_FULL ((uint8_t)0x05)
  SecureBoot mode Full.
- #define SECUREBOOT_MODE_FULL_STORE ((uint8_t)0x06)
  SecureBoot mode FullStore.
- #define SECUREBOOT_MODE_FULL_COPY ((uint8_t)0x07)

SecureBoot mode FullCopy.
- #define SECUREBOOT_MODE_PROHIBIT_FLAG ((uint8_t)0x40)

  SecureBoot mode flag to prohibit SecureBoot until next power cycle.
- #define SECUREBOOT_MODE_ENC_MAC_FLAG ((uint8_t)0x80)

  SecureBoot mode flag for encrypted digest and returning validating MAC.
- #define SECUREBOOTCONFIG_OFFSET (70)

  SecureBootConfig byte offset into the configuration zone.
- #define SECUREBOOTCONFIG_MODE_MASK ((uint16_t)0x0003)

  Mask for SecureBootMode field in SecureBootConfig value.
- #define SECUREBOOTCONFIG_MODE_DISABLED ((uint16_t)0x0000)

  Disabled SecureBootMode in SecureBootConfig value.
- #define SECUREBOOTCONFIG_MODE_FULL_BOTH ((uint16_t)0x0001)

  Both digest and signature always required SecureBootMode in SecureBootConfig value.
- #define SECUREBOOTCONFIG_MODE_FULL_SIG ((uint16_t)0x0002)

  Signature stored SecureBootMode in SecureBootConfig value.
- #define SECUREBOOTCONFIG_MODE_FULL_DIG ((uint16_t)0x0003)

  Digest stored SecureBootMode in SecureBootConfig value.

Definitions for the SelfTest Command

- #define SELFTEST_MODE_IDX ATCA_PARAM1_IDX

  SelfTest command index for mode.
- #define SELFTEST_COUNT ATCA_CMD_SIZE_MIN

  SelfTest command packet size.
- #define SELFTEST_MODE_RNG ((uint8_t)0x01)

  SelfTest mode RNG DRBG function.
- #define SELFTEST_MODE_ECDSA_SIGN_VERIFY ((uint8_t)0x02)

  SelfTest mode ECDSA verify function.
- #define SELFTEST_MODE_ECDH ((uint8_t)0x08)

  SelfTest mode ECDH function.
- #define SELFTEST_MODE_AES ((uint8_t)0x10)

  SelfTest mode AES encrypt function.
- #define SELFTEST_MODE_SHA ((uint8_t)0x20)

  SelfTest mode SHA function.
- #define SELFTEST_MODE_ALL ((uint8_t)0x3B)

  SelfTest mode all algorithms.
- #define SELFTEST_RSP_SIZE ATCA_RSP_SIZE_MIN

  SelfTest command response packet size.

Definitions for the SHA Command

- #define SHA_COUNT_SHORT ATCA_CMD_SIZE_MIN
- #define SHA_COUNT_LONG ATCA_CMD_SIZE_MIN

  Just a starting size.
- #define ATCA_SHA_DIGEST_SIZE (32)
- #define SHA_DATA_MAX (64)
- #define ATCA_SHA256_BLOCK_SIZE (64)
- #define SHA_CONTEXT_MAX_SIZE (99)
- #define SHA_MODE_MASK ((uint8_t)0x07)

  Mask the bit 0-2.
- #define SHA_MODE_SHA256_START ((uint8_t)0x00)

  Initialization, does not accept a message.
- #define SHA_MODE_SHA256_UPDATE ((uint8_t)0x01)

  Add 64 bytes in the meesage to the SHA context.
- #define SHA_MODE_SHA256_END ((uint8_t)0x02)

  Complete the calculation and return the digest.
- #define SHA_MODE_SHA256_PUBLIC ((uint8_t)0x03)

  Add 64 byte ECC public key in the slot to the SHA context.

- #define SHA_MODE_HMAC_START ((uint8_t)0x04)
  Initialization, HMAC calculation.
- #define SHA_MODE_HMAC_UPDATE ((uint8_t)0x01)
  Add 64 bytes in the meesage to the SHA context.
- #define SHA_MODE_HMAC_END ((uint8_t)0x05)
  Complete the HMAC computation and return digest.
- #define SHA_MODE_608_HMAC_END ((uint8_t)0x02)
  Complete the HMAC computation and return digest... Different command on 608.
- #define SHA_MODE_READ_CONTEXT ((uint8_t)0x06)
  Read current SHA-256 context out of the device.
- #define SHA_MODE_WRITE_CONTEXT ((uint8_t)0x07)
  Restore a SHA-256 context into the device.
- #define SHA_MODE_TARGET_MASK ((uint8_t)0xC0)
  Resulting digest target location mask.
- #define SHA_MODE_TARGET_TEMPKEY ((uint8_t)0x00)
  Place resulting digest both in Output buffer and TempKey.
- #define SHA_MODE_TARGET_MSGDIGBUF ((uint8_t)0x40)
  Place resulting digest both in Output buffer and Message Digest Buffer.
- #define SHA_MODE_TARGET_OUT_ONLY ((uint8_t)0xC0)
  Place resulting digest both in Output buffer ONLY.
- #define SHA_RSP_SIZE ATCA_RSP_SIZE_32
  SHA command response packet size.
- #define SHA_RSP_SIZE_SHORT ATCA_RSP_SIZE_MIN
  SHA command response packet size only status code.
- #define SHA_RSP_SIZE_LONG ATCA_RSP_SIZE_32
  SHA command response packet size.

Definitions for the Sign Command

- #define SIGN_MODE_IDX ATCA_PARAM1_IDX
  Sign command index for mode.
- #define SIGN_KEYID_IDX ATCA_PARAM2_IDX
  Sign command index for key id.
- #define SIGN_COUNT ATCA_CMD_SIZE_MIN
  Sign command packet size.
- #define SIGN_MODE_MASK ((uint8_t)0xE1)
  Sign mode bits 1 to 4 are 0.
- #define SIGN_MODE_INTERNAL ((uint8_t)0x00)
  Sign mode 0: internal.
- #define SIGN_MODE_INVALIDATE ((uint8_t)0x01)
  Sign mode bit 1: Signature will be used for Verify(Invalidate)
- #define SIGN_MODE_INCLUDE_SN ((uint8_t)0x40)
  Sign mode bit 6: include serial number.
- #define SIGN_MODE_EXTERNAL ((uint8_t)0x80)
  Sign mode bit 7: external.
- #define SIGN_MODE_SOURCE_MASK ((uint8_t)0x20)
  Sign mode message source mask.
- #define SIGN_MODE_SOURCE_TEMPKEY ((uint8_t)0x00)
  Sign mode message source is TempKey.
- #define SIGN_MODE_SOURCE_MSGDIGBUF ((uint8_t)0x20)
  Sign mode message source is the Message Digest Buffer.
- #define SIGN_RSP_SIZE ATCA_RSP_SIZE_MAX
  Sign command response packet size.

Definitions for the UpdateExtra Command

- #define UPDATE_MODE_IDX ATCA_PARAM1_IDX
  UpdateExtra command index for mode.

- #define UPDATE_VALUE_IDX ATCA_PARAM2_IDX
  UpdateExtra command index for new value.
- #define UPDATE_COUNT ATCA_CMD_SIZE_MIN
  UpdateExtra command packet size.
- #define UPDATE_MODE_USER_EXTRA ((uint8_t)0x00)
  UpdateExtra mode update UserExtra (config byte 84)
- #define UPDATE_MODE_SELECTOR ((uint8_t)0x01)
  UpdateExtra mode update Selector (config byte 85)
- #define UPDATE_MODE_USER_EXTRA_ADD UPDATE_MODE_SELECTOR
  UpdateExtra mode update UserExtraAdd (config byte 85)
- #define UPDATE_MODE_DEC_COUNTER ((uint8_t)0x02)
  UpdateExtra mode: decrement counter.
- #define UPDATE_RSP_SIZE ATCA_RSP_SIZE_MIN
  UpdateExtra command response packet size.

Definitions for the Verify Command

- #define VERIFY_MODE_IDX ATCA_PARAM1_IDX
  Verify command index for mode.
- #define VERIFY_KEYID_IDX ATCA_PARAM2_IDX
  Verify command index for key id.
- #define VERIFY_DATA_IDX (5)
  Verify command index for data.
- #define VERIFY_256_STORED_COUNT (71)
  Verify command packet size for 256-bit key in stored mode.
- #define VERIFY_283_STORED_COUNT (79)
  Verify command packet size for 283-bit key in stored mode.
- #define VERIFY_256_VALIDATE_COUNT (90)
  Verify command packet size for 256-bit key in validate mode.
- #define VERIFY_283_VALIDATE_COUNT (98)
  Verify command packet size for 283-bit key in validate mode.
- #define VERIFY_256_EXTERNAL_COUNT (135)
  Verify command packet size for 256-bit key in external mode.
- #define VERIFY_283_EXTERNAL_COUNT (151)
  Verify command packet size for 283-bit key in external mode.
- #define VERIFY_256_KEY_SIZE (64)
  Verify key size for 256-bit key.
- #define VERIFY_283_KEY_SIZE (72)
  Verify key size for 283-bit key.
- #define VERIFY_256_SIGNATURE_SIZE (64)
  Verify signature size for 256-bit key.
- #define VERIFY_283_SIGNATURE_SIZE (72)
  Verify signature size for 283-bit key.
- #define VERIFY_OTHER_DATA_SIZE (19)
  Verify size of "other data".
- #define VERIFY_MODE_MASK ((uint8_t)0x03)
  Verify mode bits 2 to 7 are 0.
- #define VERIFY_MODE_STORED ((uint8_t)0x00)
  Verify mode: stored.
- #define VERIFY_MODE_VALIDATE_EXTERNAL ((uint8_t)0x01)
  Verify mode: validate external.
- #define VERIFY_MODE_EXTERNAL ((uint8_t)0x02)
  Verify mode: external.
- #define VERIFY_MODE_VALIDATE ((uint8_t)0x03)
  Verify mode: validate.
- #define VERIFY_MODE_INVALIDATE ((uint8_t)0x07)
  Verify mode: invalidate.
- #define VERIFY_MODE_SOURCE_MASK ((uint8_t)0x20)

Verify mode message source mask.
- #define VERIFY_MODE_SOURCE_TEMPKEY ((uint8_t)0x00)
  Verify mode message source is TempKey.
- #define VERIFY_MODE_SOURCE_MSGDIGBUF ((uint8_t)0x20)
  Verify mode message source is the Message Digest Buffer.
- #define VERIFY_MODE_MAC_FLAG ((uint8_t)0x80)
  Verify mode: MAC.
- #define VERIFY_KEY_B283 ((uint16_t)0x0000)
  Verify key type: B283.
- #define VERIFY_KEY_K283 ((uint16_t)0x0001)
  Verify key type: K283.
- #define VERIFY_KEY_P256 ((uint16_t)0x0004)
  Verify key type: P256.
- #define VERIFY_RSP_SIZE ATCA_RSP_SIZE_MIN
  Verify command response packet size.
- #define VERIFY_RSP_SIZE_MAC ATCA_RSP_SIZE_32
  Verify command response packet size with validating MAC.

Definitions for the Write Command

- #define WRITE_ZONE_IDX ATCA_PARAM1_IDX
  Write command index for zone.
- #define WRITE_ADDR_IDX ATCA_PARAM2_IDX
  Write command index for address.
- #define WRITE_VALUE_IDX ATCA_DATA_IDX
  Write command index for data.
- #define WRITE_MAC_VS_IDX (9)
  Write command index for MAC following short data.
- #define WRITE_MAC_VL_IDX (37)
  Write command index for MAC following long data.
- #define WRITE_MAC_SIZE (32)
  Write MAC size.
- #define WRITE_ZONE_MASK ((uint8_t)0xC3)
  Write zone bits 2 to 5 are 0.
- #define WRITE_ZONE_WITH_MAC ((uint8_t)0x40)
  Write zone bit 6: write encrypted with MAC.
- #define WRITE_ZONE_OTP ((uint8_t)1)
  Write zone id OTP.
- #define WRITE_ZONE_DATA ((uint8_t)2)
  Write zone id data.
- #define WRITE_RSP_SIZE ATCA_RSP_SIZE_MIN
  Write command response packet size.

Typedefs

- typedef struct atca_command ∗ ATCACommand

Functions

- ATCA_STATUS initATCACommand (ATCADeviceType device_type, ATCACommand ca_cmd)
  Initializer for ATCACommand.
- ATCACommand newATCACommand (ATCADeviceType device_type)
  constructor for ATCACommand
- void deleteATCACommand (ATCACommand ∗ca_cmd)
  ATCACommand destructor.

- ATCA_STATUS atCheckMAC (ATCACommand ca_cmd, ATCAPacket ∗packet)

  ATCACommand CheckMAC method.
- ATCA_STATUS atCounter (ATCACommand ca_cmd, ATCAPacket ∗packet)

  ATCACommand Counter method.
- ATCA_STATUS atDeriveKey (ATCACommand ca_cmd, ATCAPacket ∗packet, bool has_mac)

  ATCACommand DeriveKey method.
- ATCA_STATUS atECDH (ATCACommand ca_cmd, ATCAPacket ∗packet)

  ATCACommand ECDH method.
- ATCA_STATUS atGenDig (ATCACommand ca_cmd, ATCAPacket ∗packet, bool is_no_mac_key)

  ATCACommand Generate Digest method.
- ATCA_STATUS atGenKey (ATCACommand ca_cmd, ATCAPacket ∗packet)

  ATCACommand Generate Key method.
- ATCA_STATUS atHMAC (ATCACommand ca_cmd, ATCAPacket ∗packet)

  ATCACommand HMAC method.
- ATCA_STATUS atInfo (ATCACommand ca_cmd, ATCAPacket ∗packet)

  ATCACommand Info method.
- ATCA_STATUS atLock (ATCACommand ca_cmd, ATCAPacket ∗packet)

  ATCACommand Lock method.
- ATCA_STATUS atMAC (ATCACommand ca_cmd, ATCAPacket ∗packet)

  ATCACommand MAC method.
- ATCA_STATUS atNonce (ATCACommand ca_cmd, ATCAPacket ∗packet)

  ATCACommand Nonce method.
- ATCA_STATUS atPause (ATCACommand ca_cmd, ATCAPacket ∗packet)

  ATCACommand Pause method.
- ATCA_STATUS atPrivWrite (ATCACommand ca_cmd, ATCAPacket ∗packet)

  ATCACommand PrivWrite method.
- ATCA_STATUS atRandom (ATCACommand ca_cmd, ATCAPacket ∗packet)

  ATCACommand Random method.
- ATCA_STATUS atRead (ATCACommand ca_cmd, ATCAPacket ∗packet)

  ATCACommand Read method.
- ATCA_STATUS atSecureBoot (ATCACommand ca_cmd, ATCAPacket ∗packet)

  ATCACommand SecureBoot method.
- ATCA_STATUS atSHA (ATCACommand ca_cmd, ATCAPacket ∗packet, uint16_t write_context_size)

  ATCACommand SHA method.
- ATCA_STATUS atSign (ATCACommand ca_cmd, ATCAPacket ∗packet)

  ATCACommand Sign method.
- ATCA_STATUS atUpdateExtra (ATCACommand ca_cmd, ATCAPacket ∗packet)

  ATCACommand UpdateExtra method.
- ATCA_STATUS atVerify (ATCACommand ca_cmd, ATCAPacket ∗packet)

  ATCACommand ECDSA Verify method.
- ATCA_STATUS atWrite (ATCACommand ca_cmd, ATCAPacket ∗packet, bool has_mac)

  ATCACommand Write method.
- ATCA_STATUS atAES (ATCACommand ca_cmd, ATCAPacket ∗packet)

  ATCACommand AES method.
- ATCA_STATUS atSelfTest (ATCACommand ca_cmd, ATCAPacket ∗packet)

  ATCACommand AES method.
- ATCA_STATUS atKDF (ATCACommand ca_cmd, ATCAPacket ∗packet)

  ATCACommand KDF method.
- bool atIsSHAFamily (ATCADeviceType device_type)

  determines if a given device type is a SHA device or a superset of a SHA device
- bool atIsECCFamily (ATCADeviceType device_type)

determines if a given device type is an ECC device or a superset of a ECC device

- ATCA_STATUS isATCAError (uint8_t ∗data)

    checks for basic error frame in data

- void atCRC (size_t length, const uint8_t ∗data, uint8_t ∗crc_le)

    Calculates CRC over the given raw data and returns the CRC in little-endian byte order.

- void atCalcCrc (ATCAPacket ∗pkt)

    This function calculates CRC and adds it to the correct offset in the packet data.

- ATCA_STATUS atCheckCrc (const uint8_t ∗response)

    This function checks the consistency of a response.

15.35.1 Detailed Description

Microchip Crypto Auth device command object - this is a command builder only, it does not send the command. The result of a command method is a fully formed packet, ready to send to the ATCAIFace object to dispatch.

This command object supports the ATSHA and ATECC device family. The command list is a superset of all device commands for this family. The command object differentiates the packet contents based on specific device type within the family.

Copyright

15.35.2 Macro Definition Documentation

15.35.2.1 SIGN_COUNT

```
#define SIGN_COUNT ATCA_CMD_SIZE_MIN
```

Sign command packet size.

15.35.2.2 SIGN_KEYID_IDX

```
#define SIGN_KEYID_IDX ATCA_PARAM2_IDX
```

Sign command index for key id.

15.35.2.3 SIGN_MODE_EXTERNAL

```
#define SIGN_MODE_EXTERNAL ((uint8_t)0x80)
```

Sign mode bit 7: external.

15.35.2.4 SIGN_MODE_IDX

```
#define SIGN_MODE_IDX ATCA_PARAM1_IDX
```

Sign command index for mode.

15.35.2.5 SIGN_MODE_INCLUDE_SN

```
#define SIGN_MODE_INCLUDE_SN ((uint8_t)0x40)
```

Sign mode bit 6: include serial number.

15.35.2.6 SIGN_MODE_INTERNAL

```
#define SIGN_MODE_INTERNAL ((uint8_t)0x00)
```

Sign mode 0: internal.

15.35.2.7 SIGN_MODE_INVALIDATE

```
#define SIGN_MODE_INVALIDATE ((uint8_t)0x01)
```

Sign mode bit 1: Signature will be used for Verify(Invalidate)

15.35.2.8 SIGN_MODE_MASK

```
#define SIGN_MODE_MASK ((uint8_t)0xE1)
```

Sign mode bits 1 to 4 are 0.

15.35.2.9 SIGN_MODE_SOURCE_MASK

```
#define SIGN_MODE_SOURCE_MASK ((uint8_t)0x20)
```

Sign mode message source mask.

15.35.2.10 SIGN_MODE_SOURCE_MSGDIGBUF

```
#define SIGN_MODE_SOURCE_MSGDIGBUF ((uint8_t)0x20)
```

Sign mode message source is the Message Digest Buffer.

15.35.2.11 SIGN_MODE_SOURCE_TEMPKEY

```
#define SIGN_MODE_SOURCE_TEMPKEY ((uint8_t)0x00)
```

Sign mode message source is TempKey.

15.35.2.12 SIGN_RSP_SIZE

```
#define SIGN_RSP_SIZE ATCA_RSP_SIZE_MAX
```

Sign command response packet size.

15.35.2.13 UPDATE_COUNT

```
#define UPDATE_COUNT ATCA_CMD_SIZE_MIN
```

UpdateExtra command packet size.

15.35.2.14 UPDATE_MODE_DEC_COUNTER

```
#define UPDATE_MODE_DEC_COUNTER ((uint8_t)0x02)
```

UpdateExtra mode: decrement counter.

15.35.2.15 UPDATE_MODE_IDX

```
#define UPDATE_MODE_IDX ATCA_PARAM1_IDX
```

UpdateExtra command index for mode.

15.35.2.16 UPDATE_MODE_SELECTOR

```
#define UPDATE_MODE_SELECTOR ((uint8_t)0x01)
```

UpdateExtra mode update Selector (config byte 85)

15.35.2.17 UPDATE_MODE_USER_EXTRA

```
#define UPDATE_MODE_USER_EXTRA ((uint8_t)0x00)
```

UpdateExtra mode update UserExtra (config byte 84)

15.35.2.18 UPDATE_MODE_USER_EXTRA_ADD

```
#define UPDATE_MODE_USER_EXTRA_ADD UPDATE_MODE_SELECTOR
```

UpdateExtra mode update UserExtraAdd (config byte 85)

15.35.2.19 UPDATE_RSP_SIZE

```
#define UPDATE_RSP_SIZE ATCA_RSP_SIZE_MIN
```

UpdateExtra command response packet size.

15.35.2.20 UPDATE_VALUE_IDX

```
#define UPDATE_VALUE_IDX ATCA_PARAM2_IDX
```

UpdateExtra command index for new value.

15.35.2.21 VERIFY_256_EXTERNAL_COUNT

```
#define VERIFY_256_EXTERNAL_COUNT (135)
```

Verify command packet size for 256-bit key in external mode.

15.35.2.22 VERIFY_256_KEY_SIZE

```
#define VERIFY_256_KEY_SIZE ( 64)
```

Verify key size for 256-bit key.

15.35.2.23 VERIFY_256_SIGNATURE_SIZE

```
#define VERIFY_256_SIGNATURE_SIZE ( 64)
```

Verify signature size for 256-bit key.

15.35.2.24 VERIFY_256_STORED_COUNT

```
#define VERIFY_256_STORED_COUNT ( 71)
```

Verify command packet size for 256-bit key in stored mode.

15.35.2.25 VERIFY_256_VALIDATE_COUNT

```
#define VERIFY_256_VALIDATE_COUNT ( 90)
```

Verify command packet size for 256-bit key in validate mode.

15.35.2.26 VERIFY_283_EXTERNAL_COUNT

```
#define VERIFY_283_EXTERNAL_COUNT (151)
```

Verify command packet size for 283-bit key in external mode.

15.35.2.27 VERIFY_283_KEY_SIZE

```
#define VERIFY_283_KEY_SIZE ( 72)
```

Verify key size for 283-bit key.

15.35.2.28 VERIFY_283_SIGNATURE_SIZE

```
#define VERIFY_283_SIGNATURE_SIZE ( 72)
```

Verify signature size for 283-bit key.

15.35.2.29 VERIFY_283_STORED_COUNT

```
#define VERIFY_283_STORED_COUNT ( 79)
```

Verify command packet size for 283-bit key in stored mode.

15.35.2.30 VERIFY_283_VALIDATE_COUNT

```
#define VERIFY_283_VALIDATE_COUNT ( 98)
```

Verify command packet size for 283-bit key in validate mode.

15.35.2.31 VERIFY_DATA_IDX

```
#define VERIFY_DATA_IDX ( 5)
```

Verify command index for data.

15.35.2.32 VERIFY_KEY_B283

```
#define VERIFY_KEY_B283 ((uint16_t)0x0000)
```

Verify key type: B283.

15.35.2.33 VERIFY_KEY_K283

```
#define VERIFY_KEY_K283 ((uint16_t)0x0001)
```

Verify key type: K283.

15.35.2.34 VERIFY_KEY_P256

```
#define VERIFY_KEY_P256 ((uint16_t)0x0004)
```

Verify key type: P256.

15.35.2.35 VERIFY_KEYID_IDX

```
#define VERIFY_KEYID_IDX ATCA_PARAM2_IDX
```

Verify command index for key id.

15.35.2.36 VERIFY_MODE_EXTERNAL

```
#define VERIFY_MODE_EXTERNAL ((uint8_t)0x02)
```

Verify mode: external.

15.35.2.37 VERIFY_MODE_IDX

```
#define VERIFY_MODE_IDX ATCA_PARAM1_IDX
```

Verify command index for mode.

15.35.2.38 VERIFY_MODE_INVALIDATE

```
#define VERIFY_MODE_INVALIDATE ((uint8_t)0x07)
```

Verify mode: invalidate.

15.35.2.39 VERIFY_MODE_MAC_FLAG

```
#define VERIFY_MODE_MAC_FLAG ((uint8_t)0x80)
```

Verify mode: MAC.

15.35.2.40 VERIFY_MODE_MASK

```
#define VERIFY_MODE_MASK ((uint8_t)0x03)
```

Verify mode bits 2 to 7 are 0.

15.35.2.41 VERIFY_MODE_SOURCE_MASK

```
#define VERIFY_MODE_SOURCE_MASK ((uint8_t)0x20)
```

Verify mode message source mask.

15.35.2.42 VERIFY_MODE_SOURCE_MSGDIGBUF

```
#define VERIFY_MODE_SOURCE_MSGDIGBUF ((uint8_t)0x20)
```

Verify mode message source is the Message Digest Buffer.

15.35.2.43 VERIFY_MODE_SOURCE_TEMPKEY

```
#define VERIFY_MODE_SOURCE_TEMPKEY ((uint8_t)0x00)
```

Verify mode message source is TempKey.

15.35.2.44 VERIFY_MODE_STORED

```
#define VERIFY_MODE_STORED ((uint8_t)0x00)
```

Verify mode: stored.

15.35.2.45 VERIFY_MODE_VALIDATE

```
#define VERIFY_MODE_VALIDATE ((uint8_t)0x03)
```

Verify mode: validate.

15.35.2.46 VERIFY_MODE_VALIDATE_EXTERNAL

```
#define VERIFY_MODE_VALIDATE_EXTERNAL ((uint8_t)0x01)
```

Verify mode: validate external.

15.35.2.47 VERIFY_OTHER_DATA_SIZE

```
#define VERIFY_OTHER_DATA_SIZE ( 19)
```

Verify size of "other data".

15.35.2.48 VERIFY_RSP_SIZE

```
#define VERIFY_RSP_SIZE ATCA_RSP_SIZE_MIN
```

Verify command response packet size.

15.35.2.49 VERIFY_RSP_SIZE_MAC

```
#define VERIFY_RSP_SIZE_MAC ATCA_RSP_SIZE_32
```

Verify command response packet size with validating MAC.

15.35.2.50 WRITE_ADDR_IDX

```
#define WRITE_ADDR_IDX ATCA_PARAM2_IDX
```

Write command index for address.

15.35.2.51 WRITE_MAC_SIZE

```
#define WRITE_MAC_SIZE (32)
```

Write MAC size.

15.35.2.52 WRITE_MAC_VL_IDX

```
#define WRITE_MAC_VL_IDX (37)
```

Write command index for MAC following long data.

15.35.2.53 WRITE_MAC_VS_IDX

```
#define WRITE_MAC_VS_IDX ( 9)
```

Write command index for MAC following short data.

15.35.2.54 WRITE_RSP_SIZE

```
#define WRITE_RSP_SIZE ATCA_RSP_SIZE_MIN
```

Write command response packet size.

15.35.2.55 WRITE_VALUE_IDX

```
#define WRITE_VALUE_IDX ATCA_DATA_IDX
```

Write command index for data.

15.35.2.56 WRITE_ZONE_DATA

```
#define WRITE_ZONE_DATA ((uint8_t)2)
```

Write zone id data.

15.35.2.57 WRITE_ZONE_IDX

```
#define WRITE_ZONE_IDX ATCA_PARAM1_IDX
```

Write command index for zone.

15.35.2.58 WRITE_ZONE_MASK

```
#define WRITE_ZONE_MASK ((uint8_t)0xC3)
```

Write zone bits 2 to 5 are 0.

15.35.2.59 WRITE_ZONE_OTP

```
#define WRITE_ZONE_OTP ((uint8_t)1)
```

Write zone id OTP.

15.35.2.60 WRITE_ZONE_WITH_MAC

```
#define WRITE_ZONE_WITH_MAC ((uint8_t)0x40)
```

Write zone bit 6: write encrypted with MAC.

15.36 atca_compiler.h File Reference

CryptoAuthLiub is meant to be portable across architectures, even non-Microchip architectures and compiler environments. This file is for isolating compiler specific macros.

15.36.1 Detailed Description

CryptoAuthLiub is meant to be portable across architectures, even non-Microchip architectures and compiler environments. This file is for isolating compiler specific macros.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.37 atca_crypto_sw.h File Reference

Common defines for CryptoAuthLib software crypto wrappers.

```
#include "atca_status.h"
```

15.37.1 Detailed Description

Common defines for CryptoAuthLib software crypto wrappers.

Copyright

15.38 atca_crypto_sw_ecdsa.c File Reference

API wrapper for software ECDSA verify. Currently unimplemented but could be implemented via a 3rd party library such as MicroECC.

```
#include "atca_crypto_sw_ecdsa.h"
```

Functions

- int atcac_sw_ecdsa_verify_p256 (const uint8_t msg[ATCA_ECC_P256_FIELD_SIZE], const uint8_t signature[ATCA_ECC_P256_SIGNATURE_SIZE], const uint8_t public_key[ATCA_ECC_P256_PUBLIC_KEY_SIZE])

    return software generated ECDSA verification result and the function is currently not implemented

15.38.1 Detailed Description

API wrapper for software ECDSA verify. Currently unimplemented but could be implemented via a 3rd party library such as MicroECC.

Copyright

15.39 atca_crypto_sw_ecdsa.h File Reference

```
#include "atca_crypto_sw.h"
#include <stddef.h>
#include <stdint.h>
```

Macros

- #define ATCA_ECC_P256_FIELD_SIZE (256 / 8)
- #define ATCA_ECC_P256_PRIVATE_KEY_SIZE (ATCA_ECC_P256_FIELD_SIZE)
- #define ATCA_ECC_P256_PUBLIC_KEY_SIZE (ATCA_ECC_P256_FIELD_SIZE ∗ 2)
- #define ATCA_ECC_P256_SIGNATURE_SIZE (ATCA_ECC_P256_FIELD_SIZE ∗ 2)

Functions

- int atcac_sw_ecdsa_verify_p256 (const uint8_t msg[ATCA_ECC_P256_FIELD_SIZE], const uint8_t signature[ATCA_ECC_P256_SIGNATURE_SIZE], const uint8_t public_key[ATCA_ECC_P256_PUBLIC_KEY_SIZE])

    return software generated ECDSA verification result and the function is currently not implemented

15.39.1 Detailed Description

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.40 atca_crypto_sw_rand.c File Reference

API wrapper for software random.

```
#include "atca_crypto_sw_rand.h"
```

Functions

- int atcac_sw_random (uint8_t *data, size_t data_size)

    return software generated random number and the function is currently not implemented

15.40.1 Detailed Description

API wrapper for software random.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.41 atca_crypto_sw_rand.h File Reference

```
#include "atca_crypto_sw.h"
#include <stddef.h>
#include <stdint.h>
```

Functions

- int atcac_sw_random (uint8_t *data, size_t data_size)

    return software generated random number and the function is currently not implemented

15.41.1 Detailed Description

Copyright

15.42 atca_crypto_sw_sha1.c File Reference

Wrapper API for SHA 1 routines.

```
#include "atca_crypto_sw_sha1.h"
#include "hashes/sha1_routines.h"
```

Functions

- int atcac_sw_sha1_init (atcac_sha1_ctx ∗ctx)

  Initialize context for performing SHA1 hash in software.
- int atcac_sw_sha1_update (atcac_sha1_ctx ∗ctx, const uint8_t ∗data, size_t data_size)

  Add arbitrary data to a SHA1 hash.
- int atcac_sw_sha1_finish (atcac_sha1_ctx ∗ctx, uint8_t digest[ATCA_SHA1_DIGEST_SIZE])

  Complete the SHA1 hash in software and return the digest.
- int atcac_sw_sha1 (const uint8_t ∗data, size_t data_size, uint8_t digest[ATCA_SHA1_DIGEST_SIZE])

  Perform SHA1 hash of data in software.

15.42.1 Detailed Description

Wrapper API for SHA 1 routines.

Copyright

15.43 atca_crypto_sw_sha1.h File Reference

Wrapper API for SHA 1 routines.

```
#include "atca_crypto_sw.h"
#include <stddef.h>
#include <stdint.h>
```

Data Structures

- struct atcac_sha1_ctx

Macros

- #define ATCA_SHA1_DIGEST_SIZE (20)

Functions

- int atcac_sw_sha1_init (atcac_sha1_ctx ∗ctx)
  Initialize context for performing SHA1 hash in software.
- int atcac_sw_sha1_update (atcac_sha1_ctx ∗ctx, const uint8_t ∗data, size_t data_size)
  Add arbitrary data to a SHA1 hash.
- int atcac_sw_sha1_finish (atcac_sha1_ctx ∗ctx, uint8_t digest[ATCA_SHA1_DIGEST_SIZE])
  Complete the SHA1 hash in software and return the digest.
- int atcac_sw_sha1 (const uint8_t ∗data, size_t data_size, uint8_t digest[ATCA_SHA1_DIGEST_SIZE])
  Perform SHA1 hash of data in software.

15.43.1 Detailed Description

Wrapper API for SHA 1 routines.

Copyright

15.44 atca_crypto_sw_sha2.c File Reference

Wrapper API for software SHA 256 routines.

```
#include "atca_crypto_sw_sha2.h"
#include "hashes/sha2_routines.h"
```

Functions

- int atcac_sw_sha2_256_init (atcac_sha2_256_ctx ∗ctx)
  initializes the SHA256 software
- int atcac_sw_sha2_256_update (atcac_sha2_256_ctx ∗ctx, const uint8_t ∗data, size_t data_size)
  updates the running hash with the next block of data, called iteratively for the entire stream of data to be hashed using the SHA256 software
- int atcac_sw_sha2_256_finish (atcac_sha2_256_ctx ∗ctx, uint8_t digest[ATCA_SHA2_256_DIGEST_SIZE])
  completes the final SHA256 calculation and returns the final digest/hash
- int atcac_sw_sha2_256 (const uint8_t ∗data, size_t data_size, uint8_t digest[ATCA_SHA2_256_DIGEST_SIZE])
  single call convenience function which computes Hash of given data using SHA256 software

15.44.1 Detailed Description

Wrapper API for software SHA 256 routines.

Copyright

15.45 atca_crypto_sw_sha2.h File Reference

Wrapper API for software SHA 256 routines.

```
#include "atca_crypto_sw.h"
#include <stddef.h>
#include <stdint.h>
```

Data Structures

- struct atcac_sha2_256_ctx

Macros

- #define ATCA_SHA2_256_DIGEST_SIZE (32)

Functions

- int atcac_sw_sha2_256_init (atcac_sha2_256_ctx ∗ctx)

  initializes the SHA256 software
- int atcac_sw_sha2_256_update (atcac_sha2_256_ctx ∗ctx, const uint8_t ∗data, size_t data_size)

  updates the running hash with the next block of data, called iteratively for the entire stream of data to be hashed using the SHA256 software
- int atcac_sw_sha2_256_finish (atcac_sha2_256_ctx ∗ctx, uint8_t digest[ATCA_SHA2_256_DIGEST_SIZE])

  completes the final SHA256 calculation and returns the final digest/hash
- int atcac_sw_sha2_256 (const uint8_t ∗data, size_t data_size, uint8_t digest[ATCA_SHA2_256_DIGEST_SIZE])

  single call convenience function which computes Hash of given data using SHA256 software

15.45.1 Detailed Description

Wrapper API for software SHA 256 routines.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.46 atca_device.c File Reference

Microchip CryptoAuth device object.

```
#include <stdlib.h>
#include "atca_device.h"
```

Functions

- ATCADevice newATCADevice (ATCAIfaceCfg ∗cfg)

  constructor for a Microchip CryptoAuth device
- void deleteATCADevice (ATCADevice ∗ca_dev)

  destructor for a device NULLs reference after object is freed
- ATCA_STATUS initATCADevice (ATCAIfaceCfg ∗cfg, ATCADevice ca_dev)

  Initializer for an Microchip CryptoAuth device.
- ATCACommand atGetCommands (ATCADevice dev)

  returns a reference to the ATCACommand object for the device
- ATCAIface atGetIFace (ATCADevice dev)

  returns a reference to the ATCAIface interface object for the device
- ATCA_STATUS releaseATCADevice (ATCADevice ca_dev)

  Release any resources associated with the device.

15.46.1 Detailed Description

Microchip CryptoAuth device object.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.47 atca_device.h File Reference

Microchip Crypto Auth device object.

```
#include "atca_command.h"
#include "atca_iface.h"
```

Data Structures

- struct atca_device

  atca_device is the C object backing ATCADevice. See the atca_device.h file for details on the ATCADevice methods.

Typedefs

- typedef struct atca_device ∗ ATCADevice

Functions

- ATCA_STATUS initATCADevice (ATCAIfaceCfg ∗cfg, ATCADevice ca_dev)

    Initializer for an Microchip CryptoAuth device.
- ATCADevice newATCADevice (ATCAIfaceCfg ∗cfg)

    constructor for a Microchip CryptoAuth device
- ATCA_STATUS releaseATCADevice (ATCADevice ca_dev)

    Release any resources associated with the device.
- void deleteATCADevice (ATCADevice ∗ca_dev)

    destructor for a device NULLs reference after object is freed
- ATCACommand atGetCommands (ATCADevice dev)

    returns a reference to the ATCACommand object for the device
- ATCAIface atGetIFace (ATCADevice dev)

    returns a reference to the ATCAIface interface object for the device

15.47.1 Detailed Description

Microchip Crypto Auth device object.

Copyright

15.48 atca_devtypes.h File Reference

Microchip Crypto Auth.

Enumerations

- enum ATCADeviceType {
  ATSHA204A, ATECC108A, ATECC508A, ATECC608A,
  ATCA_DEV_UNKNOWN = 0x20 }

    The supported Device type in Cryptoauthlib library.

15.48.1 Detailed Description

Microchip Crypto Auth.

Copyright

15.49 atca_execution.c File Reference

Implements an execution handler that executes a given command on a device and returns the results.

```
#include <stdlib.h>
#include <string.h>
#include "atca_command.h"
#include "atca_device.h"
#include "atca_execution.h"
#include "atca_devtypes.h"
#include "hal/atca_hal.h"
```

Macros

- #define ATCA_POLLING_INIT_TIME_MSEC 1
- #define ATCA_POLLING_FREQUENCY_TIME_MSEC 2
- #define ATCA_POLLING_MAX_TIME_MSEC 2500

Functions

- ATCA_STATUS atca_execute_command (ATCAPacket ∗packet, ATCADevice device)

  Wakes up device, sends the packet, waits for command completion, receives response, and puts the device into the idle state.

15.49.1 Detailed Description

Implements an execution handler that executes a given command on a device and returns the results.

This implementation wraps Polling and No polling (simple wait) schemes into a single method and use it across the library. Polling is used by default, however, by defining the ATCA_NO_POLL symbol the code will instead wait an estimated max execution time before requesting the result.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.49.2 Macro Definition Documentation

15.49.2.1 ATCA_POLLING_FREQUENCY_TIME_MSEC

```
#define ATCA_POLLING_FREQUENCY_TIME_MSEC 2
```

15.49.2.2 ATCA_POLLING_INIT_TIME_MSEC

```
#define ATCA_POLLING_INIT_TIME_MSEC 1
```

15.49.2.3 ATCA_POLLING_MAX_TIME_MSEC

```
#define ATCA_POLLING_MAX_TIME_MSEC 2500
```

15.49.3 Function Documentation

15.49.3.1 atca_execute_command()

```
ATCA_STATUS atca_execute_command (
            ATCAPacket * packet,
            ATCADevice device )
```

Wakes up device, sends the packet, waits for command completion, receives response, and puts the device into the idle state.

Parameters

in,out	*packet*	As input, the packet to be sent. As output, the data buffer in the packet structure will contain the response.
in	*device*	CryptoAuthentication device to send the command to.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.50 atca_execution.h File Reference

Defines an execution handler that executes a given command on a device and returns the results.

```
#include "atca_status.h"
#include "atca_command.h"
#include "atca_device.h"
```

Macros

- #define ATCA_UNSUPPORTED_CMD ((uint16_t)0xFFFF)

Functions

- ATCA_STATUS atca_execute_command (ATCAPacket *packet, ATCADevice device)

  Wakes up device, sends the packet, waits for command completion, receives response, and puts the device into the idle state.

15.50.1 Detailed Description

Defines an execution handler that executes a given command on a device and returns the results.

The basic flow is to wake the device, send the command, wait/poll for completion, and finally receives the response from the device and does basic checks before returning to caller.

This handler supports the ATSHA and ATECC device family.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.50.2 Macro Definition Documentation

15.50.2.1 ATCA_UNSUPPORTED_CMD

```
#define ATCA_UNSUPPORTED_CMD ((uint16_t)0xFFFF)
```

15.50.3 Function Documentation

15.50.3.1 atca_execute_command()

```
ATCA_STATUS atca_execute_command (
            ATCAPacket * packet,
            ATCADevice device )
```

Wakes up device, sends the packet, waits for command completion, receives response, and puts the device into the idle state.

Parameters

in,out	*packet*	As input, the packet to be sent. As output, the data buffer in the packet structure will contain the response.
in	*device*	CryptoAuthentication device to send the command to.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.51 atca_hal.c File Reference

low-level HAL - methods used to setup indirection to physical layer interface. this level does the dirty work of abstracting the higher level ATCAIFace methods from the low-level physical interfaces. Its main goal is to keep low-level details from bleeding into the logical interface implemetation.

```
#include "cryptoauthlib.h"
#include "atca_hal.h"
```

Functions

- ATCA_STATUS hal_iface_init (ATCAIfaceCfg ∗cfg, ATCAHAL_t ∗hal)

  Standard HAL API for ATCA to initialize a physical interface.
- ATCA_STATUS hal_iface_release (ATCAIfaceType iface_type, void ∗hal_data)

  releases a physical interface, HAL knows how to interpret hal_data
- ATCA_STATUS hal_check_wake (const uint8_t ∗response, int response_size)

  Utility function for hal_wake to check the reply.

15.51.1 Detailed Description

low-level HAL - methods used to setup indirection to physical layer interface. this level does the dirty work of abstracting the higher level ATCAIFace methods from the low-level physical interfaces. Its main goal is to keep low-level details from bleeding into the logical interface implemetation.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.52 atca_hal.h File Reference

low-level HAL - methods used to setup indirection to physical layer interface

```
#include "atca_status.h"
#include "atca_iface.h"
#include "atca_start_config.h"
#include "atca_start_iface.h"
```

Data Structures

- struct ATCAHAL_t

  an intermediary data structure to allow the HAL layer to point the standard API functions used by the upper layers to the HAL implementation for the interface. This isolates the upper layers and loosely couples the ATCAIface object from the physical implementation.

Functions

- ATCA_STATUS hal_iface_init (ATCAIfaceCfg ∗, ATCAHAL_t ∗hal)

    Standard HAL API for ATCA to initialize a physical interface.
- ATCA_STATUS hal_iface_release (ATCAIfaceType, void ∗hal_data)

    releases a physical interface, HAL knows how to interpret hal_data
- ATCA_STATUS hal_check_wake (const uint8_t ∗response, int response_size)

    Utility function for hal_wake to check the reply.
- void atca_delay_us (uint32_t delay)

    Timer API implemented at the HAL level.
- void atca_delay_10us (uint32_t delay)

    This function delays for a number of tens of microseconds.
- void atca_delay_ms (uint32_t delay)

    This function delays for a number of milliseconds.
- ATCA_STATUS hal_create_mutex (void ∗∗ppMutex, char ∗pName)

    Optional hal interfaces.
- ATCA_STATUS hal_destroy_mutex (void ∗pMutex)
- ATCA_STATUS hal_lock_mutex (void ∗pMutex)
- ATCA_STATUS hal_unlock_mutex (void ∗pMutex)

15.52.1 Detailed Description

low-level HAL - methods used to setup indirection to physical layer interface

Copyright

15.53 atca_helpers.c File Reference

Helpers to support the CryptoAuthLib Basic API methods.

```
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include "cryptoauthlib.h"
#include "atca_helpers.h"
```

Macros

- #define B64_IS_EQUAL (uint8_t)64
- #define B64_IS_INVALID (uint8_t)0xFF

Functions

- ATCA_STATUS atcab_bin2hex (const uint8_t ∗bin, size_t bin_size, char ∗hex, size_t ∗hex_size)

  Convert a binary buffer to a hex string for easy reading.
- ATCA_STATUS atcab_reversal (const uint8_t ∗bin, size_t bin_size, uint8_t ∗dest, size_t ∗dest_size)

  To reverse the input data.
- ATCA_STATUS atcab_bin2hex_ (const uint8_t ∗bin, size_t bin_size, char ∗hex, size_t ∗hex_size, bool is_pretty, bool is_space, bool is_upper)

  Function that converts a binary buffer to a hex string suitable for easy reading.
- ATCA_STATUS atcab_hex2bin_ (const char ∗hex, size_t hex_size, uint8_t ∗bin, size_t ∗bin_size, bool is_space)
- ATCA_STATUS atcab_hex2bin (const char ∗hex, size_t hex_size, uint8_t ∗bin, size_t ∗bin_size)

  Function that converts a hex string to binary buffer.
- bool isDigit (char c)

  Checks to see if a character is an ASCII representation of a digit ((c ge '0') and (c le '9'))
- bool isWhiteSpace (char c)

  Checks to see if a character is whitespace.
- bool isAlpha (char c)

  Checks to see if a character is an ASCII representation of hex ((c >= 'A') and (c <= 'F')) || ((c >= 'a') and (c <= 'f'))
- bool isHexAlpha (char c)

  Checks to see if a character is an ASCII representation of hex ((c >= 'A') and (c <= 'F')) || ((c >= 'a') and (c <= 'f'))
- bool isHex (char c)

  Returns true if this character is a valid hex character or if this is whitespace (The character can be included in a valid hexstring).
- bool isHexDigit (char c)

  Returns true if this character is a valid hex character.
- ATCA_STATUS packHex (const char ∗ascii_hex, size_t ascii_hex_len, char ∗packed_hex, size_t ∗packed_len)

  Remove white space from a ASCII hex string.
- bool isBase64 (char c, const uint8_t ∗rules)

  Returns true if this character is a valid base 64 character or if this is whitespace (A character can be included in a valid base 64 string).
- bool isBase64Digit (char c, const uint8_t ∗rules)

  Returns true if this character is a valid base 64 character.
- uint8_t base64Index (char c, const uint8_t ∗rules)

  Returns the base 64 index of the given character.
- char base64Char (uint8_t id, const uint8_t ∗rules)

  Returns the base 64 character of the given index.
- ATCA_STATUS atcab_base64decode_ (const char ∗encoded, size_t encoded_size, uint8_t ∗data, size_t ∗data_size, const uint8_t ∗rules)

  Decode base64 string to data with ruleset option.
- ATCA_STATUS atcab_base64encode_ (const uint8_t ∗data, size_t data_size, char ∗encoded, size_t ∗encoded_size, const uint8_t ∗rules)

  Encode data as base64 string with ruleset option.
- ATCA_STATUS atcab_base64encode (const uint8_t ∗byte_array, size_t array_len, char ∗encoded, size_t ∗encoded_len)

  Encode data as base64 string.
- ATCA_STATUS atcab_base64decode (const char ∗encoded, size_t encoded_len, uint8_t ∗byte_array, size_t ∗array_len)

  Decode base64 string to data.

Variables

- uint8_t atcab_b64rules_default [4] = { '+', '/', '=', 64 }
- uint8_t atcab_b64rules_mime [4] = { '+', '/', '=', 76 }
- uint8_t atcab_b64rules_urlsafe [4] = { '-', '_', 0, 0 }

15.53.1 Detailed Description

Helpers to support the CryptoAuthLib Basic API methods.

Copyright

15.53.2 Macro Definition Documentation

15.53.2.1 B64_IS_EQUAL

```
#define B64_IS_EQUAL (uint8_t)64
```

15.53.2.2 B64_IS_INVALID

```
#define B64_IS_INVALID (uint8_t)0xFF
```

15.53.3 Function Documentation

15.53.3.1 atcab_base64decode()

```
ATCA_STATUS atcab_base64decode (
            const char * encoded,
            size_t encoded_len,
            uint8_t * byte_array,
            size_t * array_len )
```

Decode base64 string to data.

Parameters

in	*encoded*	Base64 string to be decoded.
in	*encoded_len*	Size of the base64 string in bytes.
out	*byte_array*	Decoded data will be returned here.
in,out	*array_len*	As input, the size of the byte_array buffer. As output, the length of the decoded data.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.53.3.2 atcab_base64decode_()

```
ATCA_STATUS atcab_base64decode_ (
            const char * encoded,
            size_t encoded_size,
            uint8_t * data,
            size_t * data_size,
            const uint8_t * rules )
```

Decode base64 string to data with ruleset option.

Parameters

in	*encoded*	Base64 string to be decoded.
in	*encoded_size*	Size of the base64 string in bytes.
out	*data*	Decoded data will be returned here.
in,out	*data_size*	As input, the size of the byte_array buffer. As output, the length of the decoded data.
in	*rules*	base64 ruleset to use

15.53.3.3 atcab_base64encode()

```
ATCA_STATUS atcab_base64encode (
            const uint8_t * byte_array,
            size_t array_len,
            char * encoded,
            size_t * encoded_len )
```

Encode data as base64 string.

Parameters

in	*byte_array*	Data to be encode in base64.
in	*array_len*	Size of byte_array in bytes.
in	*encoded*	Base64 output is returned here.
in,out	*encoded_len*	As input, the size of the encoded buffer. As output, the length of the encoded base64 character string.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.53.3.4 atcab_base64encode_()

```
ATCA_STATUS atcab_base64encode_ (
            const uint8_t * data,
            size_t data_size,
            char * encoded,
            size_t * encoded_size,
            const uint8_t * rules )
```

Encode data as base64 string with ruleset option.

Parameters

in	*data*	The input byte array that will be converted to base 64 encoded characters
in	*data_size*	The length of the byte array
in	*encoded*	The output converted to base 64 encoded characters.
in,out	*encoded_size*	Input: The size of the encoded buffer, Output: The length of the encoded base 64 character string
in	*rules*	ruleset to use during encoding

15.53.3.5 atcab_bin2hex()

```
ATCA_STATUS atcab_bin2hex (
            const uint8_t * bin,
            size_t bin_size,
            char * hex,
            size_t * hex_size )
```

Convert a binary buffer to a hex string for easy reading.

Parameters

in	*bin*	Input data to convert.
in	*bin_size*	Size of data to convert.
out	*hex*	Buffer that receives hex string.
in,out	*hex_size*	As input, the size of the hex buffer. As output, the size of the output hex.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.53.3.6 atcab_bin2hex_()

```
ATCA_STATUS atcab_bin2hex_ (
            const uint8_t * bin,
```

```
              size_t bin_size,
              char * hex,
              size_t * hex_size,
              bool is_pretty,
              bool is_space,
              bool is_upper )
```

Function that converts a binary buffer to a hex string suitable for easy reading.

Parameters

in	*bin*	Input data to convert.
in	*bin_size*	Size of data to convert.
out	*hex*	Buffer that receives hex string.
in,out	*hex_size*	As input, the size of the hex buffer. As output, the size of the output hex.
in	*is_pretty*	Indicates whether new lines should be added for pretty printing.
in	*is_space*	Convert the output hex with space between it.
in	*is_upper*	Convert the output hex to upper case.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.53.3.7 atcab_hex2bin()

```
ATCA_STATUS atcab_hex2bin (
              const char * hex,
              size_t hex_size,
              uint8_t * bin,
              size_t * bin_size )
```

Function that converts a hex string to binary buffer.

Parameters

in	*hex*	Input buffer to convert
in	*hex_size*	Length of buffer to convert
out	*bin*	Buffer that receives binary
in,out	*bin_size*	As input, the size of the bin buffer. As output, the size of the bin data.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.53.3.8 atcab_hex2bin_()

```
ATCA_STATUS atcab_hex2bin_ (
            const char * hex,
            size_t hex_size,
            uint8_t * bin,
            size_t * bin_size,
            bool is_space )
```

15.53.3.9 atcab_reversal()

```
ATCA_STATUS atcab_reversal (
            const uint8_t * bin,
            size_t bin_size,
            uint8_t * dest,
            size_t * dest_size )
```

To reverse the input data.

Parameters

in	*bin*	Input data to reverse.
in	*bin_size*	Size of data to reverse.
out	*dest*	Buffer to store reversed binary data.
in	*dest_size*	The size of the dest buffer.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.53.3.10 base64Char()

```
char base64Char (
            uint8_t id,
            const uint8_t * rules )
```

Returns the base 64 character of the given index.

Parameters

in	*id*	index to check
in	*rules*	base64 ruleset to use

Returns

the base 64 character of the given index

15.53.3.11 base64Index()

```
uint8_t base64Index (
            char c,
            const uint8_t * rules )
```

Returns the base 64 index of the given character.

Parameters

in	*c*	character to check
in	*rules*	base64 ruleset to use

Returns

the base 64 index of the given character

15.53.3.12 isAlpha()

```
bool isAlpha (
            char c )
```

Checks to see if a character is an ASCII representation of hex ((c >= 'A') and (c <= 'F')) || ((c >= 'a') and (c <= 'f'))

Parameters

in	*c*	character to check

Returns

True if the character is a hex

15.53.3.13 isBase64()

```
bool isBase64 (
            char c,
            const uint8_t * rules )
```

Returns true if this character is a valid base 64 character or if this is whitespace (A character can be included in a valid base 64 string).

Parameters

in	*c*	character to check
in	*rules*	base64 ruleset to use

Returns

True if the character can be included in a valid base 64 string

15.53.3.14 isBase64Digit()

```
bool isBase64Digit (
            char c,
            const uint8_t * rules )
```

Returns true if this character is a valid base 64 character.

Parameters

in	*c*	character to check
in	*rules*	base64 ruleset to use

Returns

True if the character can be included in a valid base 64 string

15.53.3.15 isDigit()

```
bool isDigit (
            char c )
```

Checks to see if a character is an ASCII representation of a digit ((c ge '0') and (c le '9'))

Parameters

in	*c*	character to check

Returns

True if the character is a digit

15.53.3.16 isHex()

```
bool isHex (
            char c )
```

Returns true if this character is a valid hex character or if this is whitespace (The character can be included in a valid hexstring).

Parameters

in	*c*	character to check

Returns

True if the character can be included in a valid hexstring

15.53.3.17 isHexAlpha()

```
bool isHexAlpha (
            char c )
```

Checks to see if a character is an ASCII representation of hex ((c >= 'A') and (c <= 'F')) || ((c >= 'a') and (c <= 'f'))

Parameters

in	*c*	character to check

Returns

True if the character is a hex

15.53.3.18 isHexDigit()

```
bool isHexDigit (
            char c )
```

Returns true if this character is a valid hex character.

Parameters

in	*c*	character to check

Returns

True if the character can be included in a valid hexstring

15.53.3.19 isWhiteSpace()

```
bool isWhiteSpace (
            char c )
```

Checks to see if a character is whitespace.

Parameters

in	*c*	character to check

Returns

True if the character is whitespace

15.53.3.20 packHex()

```
ATCA_STATUS packHex (
            const char * ascii_hex,
            size_t ascii_hex_len,
            char * packed_hex,
            size_t * packed_len )
```

Remove white space from a ASCII hex string.

Parameters

in	*ascii_hex*	Initial hex string to remove white space from
in	*ascii_hex_len*	Length of the initial hex string
in	*packed_hex*	Resulting hex string without white space
in,out	*packed_len*	In: Size to packed_hex buffer Out: Number of bytes in the packed hex string

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.53.4 Variable Documentation

15.53.4.1 atcab_b64rules_default

```
uint8_t atcab_b64rules_default[4] = { '+', '/', '=', 64 }
```

15.53.4.2 atcab_b64rules_mime

```
uint8_t atcab_b64rules_mime[4] = { '+', '/', '=', 76 }
```

15.53.4.3 atcab_b64rules_urlsafe

```
uint8_t atcab_b64rules_urlsafe[4] = { '-', '_', 0, 0 }
```

15.54 atca_helpers.h File Reference

Helpers to support the CryptoAuthLib Basic API methods.

```
#include "cryptoauthlib.h"
```

- uint8_t atcab_b64rules_default [4]
- uint8_t atcab_b64rules_mime [4]
- uint8_t atcab_b64rules_urlsafe [4]
- ATCA_STATUS atcab_printbin (uint8_t ∗binary, size_t bin_len, bool add_space)
- ATCA_STATUS atcab_bin2hex (const uint8_t ∗bin, size_t bin_size, char ∗hex, size_t ∗hex_size)

  Convert a binary buffer to a hex string for easy reading.
- ATCA_STATUS atcab_bin2hex_ (const uint8_t ∗bin, size_t bin_size, char ∗hex, size_t ∗hex_size, bool is_↩ pretty, bool is_space, bool is_upper)

  Function that converts a binary buffer to a hex string suitable for easy reading.
- ATCA_STATUS atcab_hex2bin (const char ∗ascii_hex, size_t ascii_hex_len, uint8_t ∗binary, size_t ∗bin_len)

  Function that converts a hex string to binary buffer.
- ATCA_STATUS atcab_hex2bin_ (const char ∗hex, size_t hex_size, uint8_t ∗bin, size_t ∗bin_size, bool is_↩ space)
- ATCA_STATUS atcab_printbin_sp (uint8_t ∗binary, size_t bin_len)
- ATCA_STATUS atcab_printbin_label (const char ∗label, uint8_t ∗binary, size_t bin_len)
- ATCA_STATUS packHex (const char ∗ascii_hex, size_t ascii_hex_len, char ∗packed_hex, size_t ∗packed↩ _len)

  Remove white space from a ASCII hex string.
- bool isDigit (char c)

  Checks to see if a character is an ASCII representation of a digit ((c ge '0') and (c le '9'))
- bool isWhiteSpace (char c)

  Checks to see if a character is whitespace.
- bool isAlpha (char c)

  Checks to see if a character is an ASCII representation of hex ((c >= 'A') and (c <= 'F')) || ((c >= 'a') and (c <= 'f'))
- bool isHexAlpha (char c)

Checks to see if a character is an ASCII representation of hex ((c >= 'A') and (c <= 'F')) || ((c >= 'a') and (c <= 'f'))
- bool isHex (char c)

  Returns true if this character is a valid hex character or if this is whitespace (The character can be included in a valid hexstring).
- bool isHexDigit (char c)

  Returns true if this character is a valid hex character.
- bool isBase64 (char c, const uint8_t ∗rules)

  Returns true if this character is a valid base 64 character or if this is whitespace (A character can be included in a valid base 64 string).
- bool isBase64Digit (char c, const uint8_t ∗rules)

  Returns true if this character is a valid base 64 character.
- uint8_t base64Index (char c, const uint8_t ∗rules)

  Returns the base 64 index of the given character.
- char base64Char (uint8_t id, const uint8_t ∗rules)

  Returns the base 64 character of the given index.
- ATCA_STATUS atcab_base64decode_ (const char ∗encoded, size_t encoded_size, uint8_t ∗data, size_t ∗data_size, const uint8_t ∗rules)

  Decode base64 string to data with ruleset option.
- ATCA_STATUS atcab_base64decode (const char ∗encoded, size_t encoded_size, uint8_t ∗data, size_↩ t ∗data_size)

  Decode base64 string to data.
- ATCA_STATUS atcab_base64encode_ (const uint8_t ∗data, size_t data_size, char ∗encoded, size_↩ t ∗encoded_size, const uint8_t ∗rules)

  Encode data as base64 string with ruleset option.
- ATCA_STATUS atcab_base64encode (const uint8_t ∗data, size_t data_size, char ∗encoded, size_↩ t ∗encoded_size)

  Encode data as base64 string.
- ATCA_STATUS atcab_reversal (const uint8_t ∗bin, size_t bin_size, uint8_t ∗dest, size_t ∗dest_size)

  To reverse the input data.

15.54.1 Detailed Description

Helpers to support the CryptoAuthLib Basic API methods.

Copyright

15.54.2 Function Documentation

15.54.2.1 atcab_base64decode()

```
ATCA_STATUS atcab_base64decode (
              const char * encoded,
              size_t encoded_len,
              uint8_t * byte_array,
              size_t * array_len )
```

Decode base64 string to data.

Parameters

in	*encoded*	Base64 string to be decoded.
in	*encoded_len*	Size of the base64 string in bytes.
out	*byte_array*	Decoded data will be returned here.
in,out	*array_len*	As input, the size of the byte_array buffer. As output, the length of the decoded data.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.54.2.2 atcab_base64decode_()

```
ATCA_STATUS atcab_base64decode_ (
             const char * encoded,
             size_t encoded_size,
             uint8_t * data,
             size_t * data_size,
             const uint8_t * rules )
```

Decode base64 string to data with ruleset option.

Parameters

in	*encoded*	Base64 string to be decoded.
in	*encoded_size*	Size of the base64 string in bytes.
out	*data*	Decoded data will be returned here.
in,out	*data_size*	As input, the size of the byte_array buffer. As output, the length of the decoded data.
in	*rules*	base64 ruleset to use

15.54.2.3 atcab_base64encode()

```
ATCA_STATUS atcab_base64encode (
             const uint8_t * byte_array,
             size_t array_len,
             char * encoded,
             size_t * encoded_len )
```

Encode data as base64 string.

Parameters

in	*byte_array*	Data to be encode in base64.
in	*array_len*	Size of byte_array in bytes.
in	*encoded*	Base64 output is returned here.
in,out	*encoded_len*	As input, the size of the encoded buffer. As output, the length of the encoded base64 character string.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.54.2.4 atcab_base64encode_()

```
ATCA_STATUS atcab_base64encode_ (
            const uint8_t * data,
            size_t data_size,
            char * encoded,
            size_t * encoded_size,
            const uint8_t * rules )
```

Encode data as base64 string with ruleset option.

Parameters

in	*data*	The input byte array that will be converted to base 64 encoded characters
in	*data_size*	The length of the byte array
in	*encoded*	The output converted to base 64 encoded characters.
in,out	*encoded_size*	Input: The size of the encoded buffer, Output: The length of the encoded base 64 character string
in	*rules*	ruleset to use during encoding

15.54.2.5 atcab_bin2hex()

```
ATCA_STATUS atcab_bin2hex (
            const uint8_t * bin,
            size_t bin_size,
            char * hex,
            size_t * hex_size )
```

Convert a binary buffer to a hex string for easy reading.

Parameters

in	*bin*	Input data to convert.
in	*bin_size*	Size of data to convert.
out	*hex*	Buffer that receives hex string.
in,out	*hex_size*	As input, the size of the hex buffer. As output, the size of the output hex.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.54.2.6 atcab_bin2hex_()

```
ATCA_STATUS atcab_bin2hex_ (
            const uint8_t * bin,
            size_t bin_size,
            char * hex,
            size_t * hex_size,
            bool is_pretty,
            bool is_space,
            bool is_upper )
```

Function that converts a binary buffer to a hex string suitable for easy reading.

Parameters

in	*bin*	Input data to convert.
in	*bin_size*	Size of data to convert.
out	*hex*	Buffer that receives hex string.
in,out	*hex_size*	As input, the size of the hex buffer. As output, the size of the output hex.
in	*is_pretty*	Indicates whether new lines should be added for pretty printing.
in	*is_space*	Convert the output hex with space between it.
in	*is_upper*	Convert the output hex to upper case.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.54.2.7 atcab_hex2bin()

```
ATCA_STATUS atcab_hex2bin (
            const char * hex,
            size_t hex_size,
            uint8_t * bin,
            size_t * bin_size )
```

Function that converts a hex string to binary buffer.

Parameters

in	*hex*	Input buffer to convert
in	*hex_size*	Length of buffer to convert
out	*bin*	Buffer that receives binary
in,out	*bin_size*	As input, the size of the bin buffer. As output, the size of the bin data.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.54.2.8 atcab_hex2bin_()

```
ATCA_STATUS atcab_hex2bin_ (
            const char * hex,
            size_t hex_size,
            uint8_t * bin,
            size_t * bin_size,
            bool is_space )
```

15.54.2.9 atcab_printbin_label()

```
ATCA_STATUS atcab_printbin_label (
            const char * label,
            uint8_t * binary,
            size_t bin_len )
```

15.54.2.10 atcab_printbin_sp()

```
ATCA_STATUS atcab_printbin_sp (
            uint8_t * binary,
            size_t bin_len )
```

15.54.2.11 atcab_reversal()

```
ATCA_STATUS atcab_reversal (
            const uint8_t * bin,
            size_t bin_size,
            uint8_t * dest,
            size_t * dest_size )
```

To reverse the input data.

Parameters

in	*bin*	Input data to reverse.
in	*bin_size*	Size of data to reverse.
out	*dest*	Buffer to store reversed binary data.
in	*dest_size*	The size of the dest buffer.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.54.2.12 base64Char()

```
char base64Char (
            uint8_t id,
            const uint8_t * rules )
```

Returns the base 64 character of the given index.

Parameters

in	*id*	index to check
in	*rules*	base64 ruleset to use

Returns

the base 64 character of the given index

15.54.2.13 base64Index()

```
uint8_t base64Index (
            char c,
            const uint8_t * rules )
```

Returns the base 64 index of the given character.

Parameters

in	*c*	character to check
in	*rules*	base64 ruleset to use

Returns

the base 64 index of the given character

15.54.2.14 isAlpha()

```
bool isAlpha (
            char c )
```

Checks to see if a character is an ASCII representation of hex ((c >= 'A') and (c <= 'F')) || ((c >= 'a') and (c <= 'f'))

Parameters

in	*c*	character to check

Returns

True if the character is a hex

15.54.2.15 isBase64()

```
bool isBase64 (
            char c,
            const uint8_t * rules )
```

Returns true if this character is a valid base 64 character or if this is whitespace (A character can be included in a valid base 64 string).

Parameters

in	*c*	character to check
in	*rules*	base64 ruleset to use

Returns

True if the character can be included in a valid base 64 string

15.54.2.16 isBase64Digit()

```
bool isBase64Digit (
            char c,
            const uint8_t * rules )
```

Returns true if this character is a valid base 64 character.

Parameters

in	*c*	character to check
in	*rules*	base64 ruleset to use

Returns

True if the character can be included in a valid base 64 string

15.54.2.17 isDigit()

```
bool isDigit (
            char c )
```

Checks to see if a character is an ASCII representation of a digit ((c ge '0') and (c le '9'))

Parameters

in	*c*	character to check

Returns

True if the character is a digit

15.54.2.18 isHex()

```
bool isHex (
            char c )
```

Returns true if this character is a valid hex character or if this is whitespace (The character can be included in a valid hexstring).

Parameters

in	*c*	character to check

Returns

True if the character can be included in a valid hexstring

15.54.2.19 isHexAlpha()

```
bool isHexAlpha (
            char c )
```

Checks to see if a character is an ASCII representation of hex ((c >= 'A') and (c <= 'F')) || ((c >= 'a') and (c <= 'f'))

Parameters

in	*c*	character to check

Returns

True if the character is a hex

15.54.2.20 isHexDigit()

```
bool isHexDigit (
            char c )
```

Returns true if this character is a valid hex character.

Parameters

in	*c*	character to check

Returns

True if the character can be included in a valid hexstring

15.54.2.21 isWhiteSpace()

```
bool isWhiteSpace (
            char c )
```

Checks to see if a character is whitespace.

Parameters

in	*c*	character to check

Returns

True if the character is whitespace

15.54.2.22 packHex()

```
ATCA_STATUS packHex (
            const char * ascii_hex,
            size_t ascii_hex_len,
            char * packed_hex,
            size_t * packed_len )
```

Remove white space from a ASCII hex string.

Parameters

in	*ascii_hex*	Initial hex string to remove white space from
in	*ascii_hex_len*	Length of the initial hex string
in	*packed_hex*	Resulting hex string without white space
in,out	*packed_len*	In: Size to packed_hex buffer Out: Number of bytes in the packed hex string

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.54.3 Variable Documentation

15.54.3.1 atcab_b64rules_default

```
uint8_t atcab_b64rules_default[4]
```

15.54.3.2 atcab_b64rules_mime

```
uint8_t atcab_b64rules_mime[4]
```

15.54.3.3 atcab_b64rules_urlsafe

```
uint8_t atcab_b64rules_urlsafe[4]
```

15.55 atca_host.c File Reference

Host side methods to support CryptoAuth computations.

```
#include "atca_host.h"
#include "crypto/atca_crypto_sw_sha2.h"
```

Functions

- uint8_t ∗ atcah_include_data (struct atca_include_data_in_out ∗param)

  This function copies otp and sn data into a command buffer.
- ATCA_STATUS atcah_nonce (struct atca_nonce_in_out ∗param)

  This function calculates host side nonce with the parameters passed.
- ATCA_STATUS atcah_io_decrypt (struct atca_io_decrypt_in_out ∗param)

  Decrypt data that's been encrypted by the IO protection key. The ECDH and KDF commands on the ATECC608A are the only ones that support this operation.
- ATCA_STATUS atcah_verify_mac (atca_verify_mac_in_out_t ∗param)

  Calculate the expected MAC on the host side for the Verify command.
- ATCA_STATUS atcah_secureboot_enc (atca_secureboot_enc_in_out_t ∗param)

  Encrypts the digest for the SecureBoot command when using the encrypted digest / validating mac option.
- ATCA_STATUS atcah_secureboot_mac (atca_secureboot_mac_in_out_t ∗param)

Calculates the expected MAC returned from the SecureBoot command when verification is a success.
- ATCA_STATUS atcah_mac (struct atca_mac_in_out ∗param)

  This function generates an SHA-256 digest (MAC) of a key, challenge, and other information.
- ATCA_STATUS atcah_check_mac (struct atca_check_mac_in_out ∗param)

  This function performs the checkmac operation to generate client response on the host side .
- ATCA_STATUS atcah_hmac (struct atca_hmac_in_out ∗param)

  This function generates an HMAC / SHA-256 hash of a key and other information.
- ATCA_STATUS atcah_gen_dig (struct atca_gen_dig_in_out ∗param)

  This function combines the current TempKey with a stored value.
- ATCA_STATUS atcah_gen_mac (struct atca_gen_dig_in_out ∗param)

  This function generates mac with session key with a plain text.
- ATCA_STATUS atcah_write_auth_mac (struct atca_write_mac_in_out ∗param)

  This function calculates the input MAC for the Write command.
- ATCA_STATUS atcah_privwrite_auth_mac (struct atca_write_mac_in_out ∗param)

  This function calculates the input MAC for the PrivWrite command.
- ATCA_STATUS atcah_derive_key (struct atca_derive_key_in_out ∗param)

  This function derives a key with a key and TempKey.
- ATCA_STATUS atcah_derive_key_mac (struct atca_derive_key_mac_in_out ∗param)

  This function calculates the input MAC for a DeriveKey command.
- ATCA_STATUS atcah_decrypt (struct atca_decrypt_in_out ∗param)

  This function decrypts 32-byte encrypted data received with the Read command.
- ATCA_STATUS atcah_sha256 (int32_t len, const uint8_t ∗message, uint8_t ∗digest)

  This function creates a SHA256 digest on a little-endian system.
- ATCA_STATUS atcah_gen_key_msg (struct atca_gen_key_in_out ∗param)

  Calculate the PubKey digest created by GenKey and saved to TempKey.
- ATCA_STATUS atcah_config_to_sign_internal (ATCADeviceType device_type, struct atca_sign_internal_in_out ∗param, const uint8_t ∗config)

  Populate the slot_config, key_config, and is_slot_locked fields in the atca_sign_internal_in_out structure from the provided config zone.
- ATCA_STATUS atcah_sign_internal_msg (ATCADeviceType device_type, struct atca_sign_internal_in_out ∗param)

  Builds the full message that would be signed by the Sign(Internal) command.
- ATCA_STATUS atcah_encode_counter_match (uint32_t counter_value, uint8_t ∗counter_match_value)

  Builds the counter match value that needs to be stored in a slot.

15.55.1 Detailed Description

Host side methods to support CryptoAuth computations.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.56 atca_host.h File Reference

Definitions and Prototypes for ATCA Utility Functions.

```
#include <stdint.h>
#include "cryptoauthlib.h"
```

Data Structures

- struct atca_temp_key

  Structure to hold TempKey fields.
- struct atca_include_data_in_out

  Input / output parameters for function atca_include_data().
- struct atca_nonce_in_out

  Input/output parameters for function atca_nonce().
- struct atca_io_decrypt_in_out
- struct atca_verify_mac
- struct atca_secureboot_enc_in_out
- struct atca_secureboot_mac_in_out
- struct atca_mac_in_out

  Input/output parameters for function atca_mac().
- struct atca_hmac_in_out

  Input/output parameters for function atca_hmac().
- struct atca_gen_dig_in_out

  Input/output parameters for function atcah_gen_dig().
- struct atca_write_mac_in_out

  Input/output parameters for function atcah_write_auth_mac() and atcah_privwrite_auth_mac().
- struct atca_derive_key_in_out

  Input/output parameters for function atcah_derive_key().
- struct atca_derive_key_mac_in_out

  Input/output parameters for function atcah_derive_key_mac().
- struct atca_decrypt_in_out

  Input/output parameters for function atca_decrypt().
- struct atca_check_mac_in_out

  Input/output parameters for function atcah_check_mac().
- struct atca_verify_in_out

  Input/output parameters for function atcah_verify().
- struct atca_gen_key_in_out

  Input/output parameters for calculating the PubKey digest put into TempKey by the GenKey command with the atcah_gen_key_msg() function.
- struct atca_sign_internal_in_out

  Input/output parameters for calculating the message and digest used by the Sign(internal) command. Used with the atcah_sign_internal_msg() function.

Macros

Definitions for ATECC Message Sizes to Calculate a SHA256 Hash

"||" is the concatenation operator. The number in braces is the length of the hash input value in bytes.

- #define ATCA_MSG_SIZE_NONCE (55)

  RandOut{32} || NumIn{20} || OpCode{1} || Mode{1} || LSB of Param2{1}.
- #define ATCA_MSG_SIZE_MAC (88)

  (Key or TempKey){32} || (Challenge or TempKey){32} || OpCode{1} || Mode{1} || Param2{2} || (OTP0_7 or 0){8} || (OTP8_10 or 0){3} || SN8{1} || (SN4_7 or 0){4} || SN0_1{2} || (SN2_3 or 0){2}
- #define ATCA_MSG_SIZE_HMAC (88)
- #define ATCA_MSG_SIZE_GEN_DIG (96)

  KeyId{32} || OpCode{1} || Param1{1} || Param2{2} || SN8{1} || SN0_1{2} || 0{25} || TempKey{32}.
- #define ATCA_MSG_SIZE_DERIVE_KEY (96)

  KeyId{32} || OpCode{1} || Param1{1} || Param2{2} || SN8{1} || SN0_1{2} || 0{25} || TempKey{32}.
- #define ATCA_MSG_SIZE_DERIVE_KEY_MAC (39)

KeyId{32} || OpCode{1} || Param1{1} || Param2{2} || SN8{1} || SN0_1{2}.
- #define ATCA_MSG_SIZE_ENCRYPT_MAC (96)

  KeyId{32} || OpCode{1} || Param1{1} || Param2{2}|| SN8{1} || SN0_1{2} || 0{25} || TempKey{32}.
- #define ATCA_MSG_SIZE_PRIVWRITE_MAC (96)

  KeyId{32} || OpCode{1} || Param1{1} || Param2{2}|| SN8{1} || SN0_1{2} || 0{21} || PlainText{36}.
- #define ATCA_COMMAND_HEADER_SIZE (4)
- #define ATCA_GENDIG_ZEROS_SIZE (25)
- #define ATCA_WRITE_MAC_ZEROS_SIZE (25)
- #define ATCA_PRIVWRITE_MAC_ZEROS_SIZE (21)
- #define ATCA_PRIVWRITE_PLAIN_TEXT_SIZE (36)
- #define ATCA_DERIVE_KEY_ZEROS_SIZE (25)
- #define HMAC_BLOCK_SIZE (64)
- #define ENCRYPTION_KEY_SIZE (64)

Default Fixed Byte Values of Serial Number (SN[0:1] and SN[8])

- #define ATCA_SN_0_DEF (0x01)
- #define ATCA_SN_1_DEF (0x23)
- #define ATCA_SN_8_DEF (0xEE)

Definition for TempKey Mode

- #define MAC_MODE_USE_TEMPKEY_MASK ((uint8_t)0x03)

  mode mask for MAC command when using TempKey

Typedefs

- typedef struct atca_temp_key atca_temp_key_t

  Structure to hold TempKey fields.
- typedef struct atca_nonce_in_out atca_nonce_in_out_t
- typedef struct atca_io_decrypt_in_out atca_io_decrypt_in_out_t
- typedef struct atca_verify_mac atca_verify_mac_in_out_t
- typedef struct atca_secureboot_enc_in_out atca_secureboot_enc_in_out_t
- typedef struct atca_secureboot_mac_in_out atca_secureboot_mac_in_out_t
- typedef struct atca_mac_in_out atca_mac_in_out_t
- typedef struct atca_gen_dig_in_out atca_gen_dig_in_out_t

  Input/output parameters for function atcah_gen_dig().
- typedef struct atca_write_mac_in_out atca_write_mac_in_out_t

  Input/output parameters for function atcah_write_auth_mac() and atcah_privwrite_auth_mac().
- typedef struct atca_check_mac_in_out atca_check_mac_in_out_t

  Input/output parameters for function atcah_check_mac().
- typedef struct atca_verify_in_out atca_verify_in_out_t
- typedef struct atca_gen_key_in_out atca_gen_key_in_out_t

  Input/output parameters for calculating the PubKey digest put into TempKey by the GenKey command with the atcah_gen_key_msg() function.
- typedef struct atca_sign_internal_in_out atca_sign_internal_in_out_t

  Input/output parameters for calculating the message and digest used by the Sign(internal) command. Used with the atcah_sign_internal_msg() function.

Functions

- ATCA_STATUS atcah_nonce (struct atca_nonce_in_out *param)

  This function calculates host side nonce with the parameters passed.
- ATCA_STATUS atcah_mac (struct atca_mac_in_out *param)

  This function generates an SHA-256 digest (MAC) of a key, challenge, and other information.
- ATCA_STATUS atcah_check_mac (struct atca_check_mac_in_out *param)

  This function performs the checkmac operation to generate client response on the host side .
- ATCA_STATUS atcah_hmac (struct atca_hmac_in_out *param)

  This function generates an HMAC / SHA-256 hash of a key and other information.
- ATCA_STATUS atcah_gen_dig (struct atca_gen_dig_in_out *param)

  This function combines the current TempKey with a stored value.
- ATCA_STATUS atcah_gen_mac (struct atca_gen_dig_in_out *param)

  This function generates mac with session key with a plain text.
- ATCA_STATUS atcah_write_auth_mac (struct atca_write_mac_in_out *param)

  This function calculates the input MAC for the Write command.
- ATCA_STATUS atcah_privwrite_auth_mac (struct atca_write_mac_in_out *param)

  This function calculates the input MAC for the PrivWrite command.
- ATCA_STATUS atcah_derive_key (struct atca_derive_key_in_out *param)

  This function derives a key with a key and TempKey.
- ATCA_STATUS atcah_derive_key_mac (struct atca_derive_key_mac_in_out *param)

  This function calculates the input MAC for a DeriveKey command.
- ATCA_STATUS atcah_decrypt (struct atca_decrypt_in_out *param)

  This function decrypts 32-byte encrypted data received with the Read command.
- ATCA_STATUS atcah_sha256 (int32_t len, const uint8_t *message, uint8_t *digest)

  This function creates a SHA256 digest on a little-endian system.
- uint8_t * atcah_include_data (struct atca_include_data_in_out *param)

  This function copies otp and sn data into a command buffer.
- ATCA_STATUS atcah_gen_key_msg (struct atca_gen_key_in_out *param)

  Calculate the PubKey digest created by GenKey and saved to TempKey.
- ATCA_STATUS atcah_config_to_sign_internal (ATCADeviceType device_type, struct atca_sign_internal_in_out *param, const uint8_t *config)

  Populate the slot_config, key_config, and is_slot_locked fields in the atca_sign_internal_in_out structure from the provided config zone.
- ATCA_STATUS atcah_sign_internal_msg (ATCADeviceType device_type, struct atca_sign_internal_in_out *param)

  Builds the full message that would be signed by the Sign(Internal) command.
- ATCA_STATUS atcah_verify_mac (atca_verify_mac_in_out_t *param)

  Calculate the expected MAC on the host side for the Verify command.
- ATCA_STATUS atcah_secureboot_enc (atca_secureboot_enc_in_out_t *param)

  Encrypts the digest for the SecureBoot command when using the encrypted digest / validating mac option.
- ATCA_STATUS atcah_secureboot_mac (atca_secureboot_mac_in_out_t *param)

  Calculates the expected MAC returned from the SecureBoot command when verification is a success.
- ATCA_STATUS atcah_encode_counter_match (uint32_t counter, uint8_t *counter_match)

  Builds the counter match value that needs to be stored in a slot.
- ATCA_STATUS atcah_io_decrypt (struct atca_io_decrypt_in_out *param)

  Decrypt data that's been encrypted by the IO protection key. The ECDH and KDF commands on the ATECC608A are the only ones that support this operation.

15.56.1 Detailed Description

Definitions and Prototypes for ATCA Utility Functions.

Copyright

15.57 atca_iface.c File Reference

Microchip CryptoAuthLib hardware interface object.

```
#include <stdlib.h>
#include "atca_iface.h"
#include "hal/atca_hal.h"
```

Macros

- #define ATCA_POST_DELAY_MSEC 25

Functions

- ATCA_STATUS _atinit (ATCAIface ca_iface, ATCAHAL_t *hal)
- ATCA_STATUS initATCAIface (ATCAIfaceCfg *cfg, ATCAIface ca_iface)

  Initializer for ATCAIface objects.
- ATCAIface newATCAIface (ATCAIfaceCfg *cfg)

  Constructor for ATCAIface objects.
- ATCA_STATUS atinit (ATCAIface ca_iface)

  Performs the HAL initialization by calling intermediate HAL wrapper function. If using the basic API, the atcab_init() function should be called instead.
- ATCA_STATUS atsend (ATCAIface ca_iface, uint8_t *txdata, int txlength)

  Sends the data to the device by calling intermediate HAL wrapper function.
- ATCA_STATUS atreceive (ATCAIface ca_iface, uint8_t *rxdata, uint16_t *rxlength)

  Receives data from the device by calling intermediate HAL wrapper function.
- ATCA_STATUS atwake (ATCAIface ca_iface)

  Wakes up the device by calling intermediate HAL wrapper function. If using the basic API, the atcab_wakeup() function should be used instead.
- ATCA_STATUS atidle (ATCAIface ca_iface)

  Puts the device into idle state by calling intermediate HAL wrapper function. If using the basic API, the atcab_idle() function should be used instead.
- ATCA_STATUS atsleep (ATCAIface ca_iface)

  Puts the device into sleep state by calling intermediate HAL wrapper function. If using the basic API, the atcab_sleep() function should be used instead.
- ATCAIfaceCfg * atgetifacecfg (ATCAIface ca_iface)

  Returns the logical interface configuration for the device.
- void * atgetifacehaldat (ATCAIface ca_iface)

  Returns the HAL data pointer for the device.
- ATCA_STATUS releaseATCAIface (ATCAIface ca_iface)

  Instruct the HAL driver to release any resources associated with this interface.
- void deleteATCAIface (ATCAIface *ca_iface)

  Instruct the HAL driver to release any resources associated with this interface, then delete the object.

15.57.1 Detailed Description

Microchip CryptoAuthLib hardware interface object.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.58 atca_iface.h File Reference

Microchip Crypto Auth hardware interface object.

```
#include "atca_command.h"
```

Data Structures

- struct ATCAIfaceCfg
- struct atca_iface

  atca_iface is the C object backing ATCAIface. See the atca_iface.h file for details on the ATCAIface methods

Typedefs

- typedef struct atca_iface ∗ ATCAIface

Enumerations

- enum ATCAIfaceType {
  ATCA_I2C_IFACE, ATCA_SWI_IFACE, ATCA_UART_IFACE, ATCA_SPI_IFACE,
  ATCA_HID_IFACE, ATCA_CUSTOM_IFACE, ATCA_UNKNOWN_IFACE }
- enum ATCAKitType { ATCA_KIT_AUTO_IFACE, ATCA_KIT_I2C_IFACE, ATCA_KIT_SWI_IFACE, ATCA_KIT_UNKNOWN_IFACE }

Functions

- ATCA_STATUS initATCAIface (ATCAIfaceCfg ∗cfg, ATCAIface ca_iface)

  Initializer for ATCAIface objects.
- ATCAIface newATCAIface (ATCAIfaceCfg ∗cfg)

  Constructor for ATCAIface objects.
- ATCA_STATUS releaseATCAIface (ATCAIface ca_iface)

  Instruct the HAL driver to release any resources associated with this interface.
- void deleteATCAIface (ATCAIface ∗ca_iface)

  Instruct the HAL driver to release any resources associated with this interface, then delete the object.
- ATCA_STATUS atinit (ATCAIface ca_iface)

  Performs the HAL initialization by calling intermediate HAL wrapper function. If using the basic API, the atcab_init() function should be called instead.
- ATCA_STATUS atpostinit (ATCAIface ca_iface)
- ATCA_STATUS atsend (ATCAIface ca_iface, uint8_t ∗txdata, int txlength)

Sends the data to the device by calling intermediate HAL wrapper function.
- ATCA_STATUS atreceive (ATCAIface ca_iface, uint8_t *rxdata, uint16_t *rxlength)

  Receives data from the device by calling intermediate HAL wrapper function.
- ATCA_STATUS atwake (ATCAIface ca_iface)

  Wakes up the device by calling intermediate HAL wrapper function. If using the basic API, the atcab_wakeup() function should be used instead.
- ATCA_STATUS atidle (ATCAIface ca_iface)

  Puts the device into idle state by calling intermediate HAL wrapper function. If using the basic API, the atcab_idle() function should be used instead.
- ATCA_STATUS atsleep (ATCAIface ca_iface)

  Puts the device into sleep state by calling intermediate HAL wrapper function. If using the basic API, the atcab_sleep() function should be used instead.
- ATCAIfaceCfg * atgetifacecfg (ATCAIface ca_iface)

  Returns the logical interface configuration for the device.
- void * atgetifacehaldat (ATCAIface ca_iface)

  Returns the HAL data pointer for the device.

15.58.1 Detailed Description

Microchip Crypto Auth hardware interface object.

Copyright

15.59 atca_jwt.c File Reference

Utilities to create and verify a JSON Web Token (JWT)

```
#include "cryptoauthlib.h"
#include "basic/atca_helpers.h"
#include "crypto/atca_crypto_sw_sha2.h"
#include "jwt/atca_jwt.h"
#include <stdio.h>
```

Functions

- void atca_jwt_check_payload_start (atca_jwt_t *jwt)

  Check the provided context to see what character needs to be added in order to append a claim.
- ATCA_STATUS atca_jwt_init (atca_jwt_t *jwt, char *buf, uint16_t buflen)

  Initialize a JWT structure.
- ATCA_STATUS atca_jwt_finalize (atca_jwt_t *jwt, uint16_t key_id)

  Close the claims of a token, encode them, then sign the result.
- ATCA_STATUS atca_jwt_add_claim_string (atca_jwt_t *jwt, const char *claim, const char *value)

  Add a string claim to a token.
- ATCA_STATUS atca_jwt_add_claim_numeric (atca_jwt_t *jwt, const char *claim, int32_t value)

  Add a numeric claim to a token.
- ATCA_STATUS atca_jwt_verify (const char *buf, uint16_t buflen, const uint8_t *pubkey)

  Verifies the signature of a jwt using the provided public key.

15.59.1 Detailed Description

Utilities to create and verify a JSON Web Token (JWT)

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.60 atca_jwt.h File Reference

Utilities to create and verify a JSON Web Token (JWT)

```
#include "cryptoauthlib.h"
```

Data Structures

- struct atca_jwt_t

  Structure to hold metadata information about the jwt being built.

Functions

- ATCA_STATUS atca_jwt_init (atca_jwt_t ∗jwt, char ∗buf, uint16_t buflen)

  Initialize a JWT structure.
- ATCA_STATUS atca_jwt_add_claim_string (atca_jwt_t ∗jwt, const char ∗claim, const char ∗value)

  Add a string claim to a token.
- ATCA_STATUS atca_jwt_add_claim_numeric (atca_jwt_t ∗jwt, const char ∗claim, int32_t value)

  Add a numeric claim to a token.
- ATCA_STATUS atca_jwt_finalize (atca_jwt_t ∗jwt, uint16_t key_id)

  Close the claims of a token, encode them, then sign the result.
- void atca_jwt_check_payload_start (atca_jwt_t ∗jwt)

  Check the provided context to see what character needs to be added in order to append a claim.
- ATCA_STATUS atca_jwt_verify (const char ∗buf, uint16_t buflen, const uint8_t ∗pubkey)

  Verifies the signature of a jwt using the provided public key.

15.60.1 Detailed Description

Utilities to create and verify a JSON Web Token (JWT)

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.61 atca_mbedtls_ecdh.c File Reference

```
#include "mbedtls/config.h"
```

15.62 atca_mbedtls_ecdsa.c File Reference

```
#include "mbedtls/config.h"
```

15.63 atca_mbedtls_wrap.c File Reference

Wrapper functions to replace cryptoauthlib software crypto functions with the mbedTLS equivalent.

```
#include "mbedtls/config.h"
#include <stdlib.h>
#include "mbedtls/pk.h"
#include "mbedtls/ecp.h"
#include "mbedtls/x509_crt.h"
#include "cryptoauthlib.h"
#include "atcacert/atcacert_client.h"
#include "atcacert/atcacert_def.h"
```

Macros

- #define mbedtls_calloc calloc
- #define mbedtls_free free

Functions

- int atca_mbedtls_pk_init (mbedtls_pk_context ∗pkey, const uint16_t slotid)

  Initializes an mbedtls pk context for use with EC operations.
- int atca_mbedtls_cert_add (mbedtls_x509_crt ∗cert, const atcacert_def_t ∗cert_def)

  Rebuild a certificate from an atcacert_def_t structure, and then add it to an mbedtls cert chain.

15.63.1 Detailed Description

Wrapper functions to replace cryptoauthlib software crypto functions with the mbedTLS equivalent.

Copyright

15.63.2 Macro Definition Documentation

15.63.2.1 mbedtls_calloc

```
#define mbedtls_calloc calloc
```

15.63.2.2 mbedtls_free

```
#define mbedtls_free free
```

15.63.3 Function Documentation

15.63.3.1 atca_mbedtls_cert_add()

```
int atca_mbedtls_cert_add (
            mbedtls_x509_crt * cert,
            const atcacert_def_t * cert_def )
```

Rebuild a certificate from an atcacert_def_t structure, and then add it to an mbedtls cert chain.

Parameters

in,out	*cert*	mbedtls cert chain. Must have already been initialized
in	*cert_def*	Certificate definition that will be rebuilt and added

Returns

0 on success, otherwise an error code.

15.64 atca_mbedtls_wrap.h File Reference

Functions

- int atca_mbedtls_pk_init (struct mbedtls_pk_context ∗pkey, const uint16_t slotid)
  Initializes an mbedtls pk context for use with EC operations.
- int atca_mbedtls_cert_add (struct mbedtls_x509_crt ∗cert, const struct atcacert_def_s ∗cert_def)
- int atca_mbedtls_ecdh_slot_cb (void)
  ECDH Callback to obtain the "slot" used in ECDH operations from the application.
- int atca_mbedtls_ecdh_ioprot_cb (uint8_t secret[32])
  ECDH Callback to obtain the IO Protection secret from the application.

15.65 atca_start_config.h File Reference

15.66 atca_start_iface.h File Reference

15.67 atca_status.h File Reference

Microchip Crypto Auth status codes.

```
#include <stdint.h>
#include "atca_bool.h"
```

Enumerations

- enum ATCA_STATUS {
ATCA_SUCCESS = 0x00, ATCA_CONFIG_ZONE_LOCKED = 0x01, ATCA_DATA_ZONE_LOCKED = 0x02, ATCA_WAKE_FAILED = 0xD0,
ATCA_CHECKMAC_VERIFY_FAILED = 0xD1, ATCA_PARSE_ERROR = 0xD2, ATCA_STATUS_CRC = 0xD4, ATCA_STATUS_UNKNOWN = 0xD5,
ATCA_STATUS_ECC = 0xD6, ATCA_STATUS_SELFTEST_ERROR = 0xD7, ATCA_FUNC_FAIL = 0xE0, ATCA_GEN_FAIL = 0xE1,
ATCA_BAD_PARAM = 0xE2, ATCA_INVALID_ID = 0xE3, ATCA_INVALID_SIZE = 0xE4, ATCA_RX_CRC_ERROR = 0xE5,
ATCA_RX_FAIL = 0xE6, ATCA_RX_NO_RESPONSE = 0xE7, ATCA_RESYNC_WITH_WAKEUP = 0xE8, ATCA_PARITY_ERROR = 0xE9,
ATCA_TX_TIMEOUT = 0xEA, ATCA_RX_TIMEOUT = 0xEB, ATCA_TOO_MANY_COMM_RETRIES = 0xEC, ATCA_SMALL_BUFFER = 0xED,
ATCA_COMM_FAIL = 0xF0, ATCA_TIMEOUT = 0xF1, ATCA_BAD_OPCODE = 0xF2, ATCA_WAKE_SUCCESS = 0xF3,
ATCA_EXECUTION_ERROR = 0xF4, ATCA_UNIMPLEMENTED = 0xF5, ATCA_ASSERT_FAILURE = 0xF6, ATCA_TX_FAIL = 0xF7,
ATCA_NOT_LOCKED = 0xF8, ATCA_NO_DEVICES = 0xF9, ATCA_HEALTH_TEST_ERROR = 0xFA, ATCA_ALLOC_FAILURE = 0xFB }

15.67.1 Detailed Description

Microchip Crypto Auth status codes.

Copyright

15.67.2 Enumeration Type Documentation

15.67.2.1 ATCA_STATUS

`enum ATCA_STATUS`

Enumerator

ATCA_SUCCESS	Function succeeded.
ATCA_CONFIG_ZONE_LOCKED	
ATCA_DATA_ZONE_LOCKED	
ATCA_WAKE_FAILED	response status byte indicates CheckMac failure (status byte = 0x01)
ATCA_CHECKMAC_VERIFY_FAILED	response status byte indicates CheckMac failure (status byte = 0x01)
ATCA_PARSE_ERROR	response status byte indicates parsing error (status byte = 0x03)
ATCA_STATUS_CRC	response status byte indicates DEVICE did not receive data properly (status byte = 0xFF)
ATCA_STATUS_UNKNOWN	response status byte is unknown

Enumerator

ATCA_STATUS_ECC	response status byte is ECC fault (status byte = 0x05)
ATCA_STATUS_SELFTEST_ERROR	response status byte is Self Test Error, chip in failure mode (status byte = 0x07)
ATCA_FUNC_FAIL	Function could not execute due to incorrect condition / state.
ATCA_GEN_FAIL	unspecified error
ATCA_BAD_PARAM	bad argument (out of range, null pointer, etc.)
ATCA_INVALID_ID	invalid device id, id not set
ATCA_INVALID_SIZE	Count value is out of range or greater than buffer size.
ATCA_RX_CRC_ERROR	CRC error in data received from device.
ATCA_RX_FAIL	Timed out while waiting for response. Number of bytes received is $>$ 0.
ATCA_RX_NO_RESPONSE	Not an error while the Command layer is polling for a command response.
ATCA_RESYNC_WITH_WAKEUP	Re-synchronization succeeded, but only after generating a Wake-up.
ATCA_PARITY_ERROR	for protocols needing parity
ATCA_TX_TIMEOUT	for Microchip PHY protocol, timeout on transmission waiting for master
ATCA_RX_TIMEOUT	for Microchip PHY protocol, timeout on receipt waiting for master
ATCA_TOO_MANY_COMM_RETRIES	Device did not respond too many times during a transmission. Could indicate no device present.
ATCA_SMALL_BUFFER	Supplied buffer is too small for data required.
ATCA_COMM_FAIL	Communication with device failed. Same as in hardware dependent modules.
ATCA_TIMEOUT	Timed out while waiting for response. Number of bytes received is 0.
ATCA_BAD_OPCODE	opcode is not supported by the device
ATCA_WAKE_SUCCESS	received proper wake token
ATCA_EXECUTION_ERROR	chip was in a state where it could not execute the command, response status byte indicates command execution error (status byte = 0x0F)
ATCA_UNIMPLEMENTED	Function or some element of it hasn't been implemented yet.
ATCA_ASSERT_FAILURE	Code failed run-time consistency check.
ATCA_TX_FAIL	Failed to write.
ATCA_NOT_LOCKED	required zone was not locked
ATCA_NO_DEVICES	For protocols that support device discovery (kit protocol), no devices were found.
ATCA_HEALTH_TEST_ERROR	random number generator health test error
ATCA_ALLOC_FAILURE	Couldn't allocate required memory.

15.68 atcacert.h File Reference

Declarations common to all atcacert code.

```
#include <stddef.h>
#include <stdint.h>
```

Macros

- #define FALSE (0)
- #define TRUE (1)
- #define ATCACERT_E_SUCCESS 0

  Operation completed successfully.
- #define ATCACERT_E_ERROR 1

  General error.
- #define ATCACERT_E_BAD_PARAMS 2

  Invalid/bad parameter passed to function.
- #define ATCACERT_E_BUFFER_TOO_SMALL 3

  Supplied buffer for output is too small to hold the result.
- #define ATCACERT_E_DECODING_ERROR 4

  Data being decoded/parsed has an invalid format.
- #define ATCACERT_E_INVALID_DATE 5

  Date is invalid.
- #define ATCACERT_E_UNIMPLEMENTED 6

  Function is unimplemented for the current configuration.
- #define ATCACERT_E_UNEXPECTED_ELEM_SIZE 7

  A certificate element size was not what was expected.
- #define ATCACERT_E_ELEM_MISSING 8

  The certificate element isn't defined for the certificate definition.
- #define ATCACERT_E_ELEM_OUT_OF_BOUNDS 9

  Certificate element is out of bounds for the given certificate.
- #define ATCACERT_E_BAD_CERT 10

  Certificate structure is bad in some way.
- #define ATCACERT_E_WRONG_CERT_DEF 11
- #define ATCACERT_E_VERIFY_FAILED 12

  Certificate or challenge/response verification failed.
- #define ATCACERT_E_INVALID_TRANSFORM 13

  Invalid transform passed to function.

15.68.1 Detailed Description

Declarations common to all atcacert code.

These are common definitions used by all the atcacert code.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.69 atcacert_client.c File Reference

Client side cert i/o methods. These declarations deal with the client-side, the node being authenticated, of the authentication process. It is assumed the client has an ECC CryptoAuthentication device (e.g. ATECC508A) and the certificates are stored on that device.

```
#include <stdlib.h>
#include "atcacert_client.h"
#include "atcacert_pem.h"
#include "cryptoauthlib.h"
#include "basic/atca_basic.h"
```

Functions

- int atcacert_get_response (uint8_t device_private_key_slot, const uint8_t challenge[32], uint8_t response[64])

  Calculates the response to a challenge sent from the host.
- int atcacert_read_device_loc (const atcacert_device_loc_t *device_loc, uint8_t *data)

  Read the data from a device location.
- int atcacert_read_cert (const atcacert_def_t *cert_def, const uint8_t ca_public_key[64], uint8_t *cert, size_t *cert_size)

  Reads the certificate specified by the certificate definition from the ATECC508A device.
- int atcacert_write_cert (const atcacert_def_t *cert_def, const uint8_t *cert, size_t cert_size)

  Take a full certificate and write it to the ATECC508A device according to the certificate definition.
- int atcacert_create_csr_pem (const atcacert_def_t *csr_def, char *csr, size_t *csr_size)

  Creates a CSR specified by the CSR definition from the ATECC508A device. This process involves reading the dynamic CSR data from the device and combining it with the template found in the CSR definition, then signing it. Return the CSR int der format.
- int atcacert_create_csr (const atcacert_def_t *csr_def, uint8_t *csr, size_t *csr_size)

  Creates a CSR specified by the CSR definition from the ATECC508A device. This process involves reading the dynamic CSR data from the device and combining it with the template found in the CSR definition, then signing it. Return the CSR int der format.

15.69.1 Detailed Description

Client side cert i/o methods. These declarations deal with the client-side, the node being authenticated, of the authentication process. It is assumed the client has an ECC CryptoAuthentication device (e.g. ATECC508A) and the certificates are stored on that device.

Copyright

15.70 atcacert_client.h File Reference

Client side cert i/o methods. These declarations deal with the client-side, the node being authenticated, of the authentication process. It is assumed the client has an ECC CryptoAuthentication device (e.g. ATECC508A) and the certificates are stored on that device.

```
#include <stddef.h>
#include <stdint.h>
#include "atcacert_def.h"
```

Functions

- int atcacert_read_device_loc (const atcacert_device_loc_t *device_loc, uint8_t *data)

  Read the data from a device location.
- int atcacert_read_cert (const atcacert_def_t *cert_def, const uint8_t ca_public_key[64], uint8_t *cert, size_t *cert_size)

  Reads the certificate specified by the certificate definition from the ATECC508A device.
- int atcacert_write_cert (const atcacert_def_t *cert_def, const uint8_t *cert, size_t cert_size)

  Take a full certificate and write it to the ATECC508A device according to the certificate definition.
- int atcacert_create_csr (const atcacert_def_t *csr_def, uint8_t *csr, size_t *csr_size)

  Creates a CSR specified by the CSR definition from the ATECC508A device. This process involves reading the dynamic CSR data from the device and combining it with the template found in the CSR definition, then signing it. Return the CSR int der format.
- int atcacert_create_csr_pem (const atcacert_def_t *csr_def, char *csr, size_t *csr_size)

  Creates a CSR specified by the CSR definition from the ATECC508A device. This process involves reading the dynamic CSR data from the device and combining it with the template found in the CSR definition, then signing it. Return the CSR int der format.
- int atcacert_get_response (uint8_t device_private_key_slot, const uint8_t challenge[32], uint8_t response[64])

  Calculates the response to a challenge sent from the host.

15.70.1 Detailed Description

Client side cert i/o methods. These declarations deal with the client-side, the node being authenticated, of the authentication process. It is assumed the client has an ECC CryptoAuthentication device (e.g. ATECC508A) and the certificates are stored on that device.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.71 atcacert_date.c File Reference

Date handling with regard to certificates.

```
#include <string.h>
#include "atcacert_date.h"
```

Functions

- int atcacert_date_enc (atcacert_date_format_t format, const atcacert_tm_utc_t *timestamp, uint8_t *formatted_date, size_t *formatted_date_size)

  Format a timestamp according to the format type.
- int atcacert_date_dec (atcacert_date_format_t format, const uint8_t *formatted_date, size_t formatted_date_size, atcacert_tm_utc_t *timestamp)

  Parse a formatted timestamp according to the specified format.
- int atcacert_date_get_max_date (atcacert_date_format_t format, atcacert_tm_utc_t *timestamp)

  Return the maximum date available for the given format.
- int atcacert_date_enc_iso8601_sep (const atcacert_tm_utc_t *timestamp, uint8_t formatted_date[DATEFMT_ISO8601_SEP_S

- int atcacert_date_dec_iso8601_sep (const uint8_t formatted_date[DATEFMT_ISO8601_SEP_SIZE], atcacert_tm_utc_t ∗timestamp)
- int atcacert_date_enc_rfc5280_utc (const atcacert_tm_utc_t ∗timestamp, uint8_t formatted_date[DATEFMT_RFC5280_UTC_S
- int atcacert_date_dec_rfc5280_utc (const uint8_t formatted_date[DATEFMT_RFC5280_UTC_SIZE], atcacert_tm_utc_t ∗timestamp)
- int atcacert_date_enc_rfc5280_gen (const atcacert_tm_utc_t ∗timestamp, uint8_t formatted_date[DATEFMT_RFC5280_GEN_
- int atcacert_date_dec_rfc5280_gen (const uint8_t formatted_date[DATEFMT_RFC5280_GEN_SIZE], atcacert_tm_utc_t ∗timestamp)
- int atcacert_date_enc_posix_uint32_be (const atcacert_tm_utc_t ∗timestamp, uint8_t formatted_←date[DATEFMT_POSIX_UINT32_BE_SIZE])
- int atcacert_date_dec_posix_uint32_be (const uint8_t formatted_date[DATEFMT_POSIX_UINT32_BE_SIZE], atcacert_tm_utc_t ∗timestamp)
- int atcacert_date_enc_posix_uint32_le (const atcacert_tm_utc_t ∗timestamp, uint8_t formatted_←date[DATEFMT_POSIX_UINT32_LE_SIZE])
- int atcacert_date_dec_posix_uint32_le (const uint8_t formatted_date[DATEFMT_POSIX_UINT32_LE_SIZE], atcacert_tm_utc_t ∗timestamp)
- int atcacert_date_enc_compcert (const atcacert_tm_utc_t ∗issue_date, uint8_t expire_years, uint8_t enc_←dates[3])

  Encode the issue and expire dates in the format used by the compressed certificate.
- int atcacert_date_dec_compcert (const uint8_t enc_dates[3], atcacert_date_format_t expire_date_format, atcacert_tm_utc_t ∗issue_date, atcacert_tm_utc_t ∗expire_date)

  Decode the issue and expire dates from the format used by the compressed certificate.

Variables

- const size_t ATCACERT_DATE_FORMAT_SIZES [ATCACERT_DATE_FORMAT_SIZES_COUNT]

15.71.1 Detailed Description

Date handling with regard to certificates.

Copyright

15.72 atcacert_date.h File Reference

Declarations for date handling with regard to certificates.

```
#include <stddef.h>
#include "atcacert.h"
```

Data Structures

- struct atcacert_tm_utc_s

Macros

- #define DATEFMT_ISO8601_SEP_SIZE (20)
- #define DATEFMT_RFC5280_UTC_SIZE (13)
- #define DATEFMT_POSIX_UINT32_BE_SIZE (4)
- #define DATEFMT_POSIX_UINT32_LE_SIZE (4)
- #define DATEFMT_RFC5280_GEN_SIZE (15)
- #define DATEFMT_MAX_SIZE DATEFMT_ISO8601_SEP_SIZE
- #define ATCACERT_DATE_FORMAT_SIZES_COUNT 5

Typedefs

- typedef struct atcacert_tm_utc_s atcacert_tm_utc_t
- typedef enum atcacert_date_format_e atcacert_date_format_t

Enumerations

- enum atcacert_date_format_e {
DATEFMT_ISO8601_SEP, DATEFMT_RFC5280_UTC, DATEFMT_POSIX_UINT32_BE, DATEFMT_POSIX_UINT32_LE,
DATEFMT_RFC5280_GEN }

Functions

- int atcacert_date_enc (atcacert_date_format_t format, const atcacert_tm_utc_t ∗timestamp, uint8_t ∗formatted_date, size_t ∗formatted_date_size)

  Format a timestamp according to the format type.
- int atcacert_date_dec (atcacert_date_format_t format, const uint8_t ∗formatted_date, size_t formatted_date_size, atcacert_tm_utc_t ∗timestamp)

  Parse a formatted timestamp according to the specified format.
- int atcacert_date_enc_compcert (const atcacert_tm_utc_t ∗issue_date, uint8_t expire_years, uint8_t enc_dates[3])

  Encode the issue and expire dates in the format used by the compressed certificate.
- int atcacert_date_dec_compcert (const uint8_t enc_dates[3], atcacert_date_format_t expire_date_format, atcacert_tm_utc_t ∗issue_date, atcacert_tm_utc_t ∗expire_date)

  Decode the issue and expire dates from the format used by the compressed certificate.
- int atcacert_date_get_max_date (atcacert_date_format_t format, atcacert_tm_utc_t ∗timestamp)

  Return the maximum date available for the given format.
- int atcacert_date_enc_iso8601_sep (const atcacert_tm_utc_t ∗timestamp, uint8_t formatted_date[DATEFMT_ISO8601_SEP_S
- int atcacert_date_dec_iso8601_sep (const uint8_t formatted_date[DATEFMT_ISO8601_SEP_SIZE], atcacert_tm_utc_t ∗timestamp)
- int atcacert_date_enc_rfc5280_utc (const atcacert_tm_utc_t ∗timestamp, uint8_t formatted_date[DATEFMT_RFC5280_UTC_S
- int atcacert_date_dec_rfc5280_utc (const uint8_t formatted_date[DATEFMT_RFC5280_UTC_SIZE], atcacert_tm_utc_t ∗timestamp)
- int atcacert_date_enc_rfc5280_gen (const atcacert_tm_utc_t ∗timestamp, uint8_t formatted_date[DATEFMT_RFC5280_GEN_
- int atcacert_date_dec_rfc5280_gen (const uint8_t formatted_date[DATEFMT_RFC5280_GEN_SIZE], atcacert_tm_utc_t ∗timestamp)
- int atcacert_date_enc_posix_uint32_be (const atcacert_tm_utc_t ∗timestamp, uint8_t formatted_date[DATEFMT_POSIX_UINT32_BE_SIZE])
- int atcacert_date_dec_posix_uint32_be (const uint8_t formatted_date[DATEFMT_POSIX_UINT32_BE_SIZE], atcacert_tm_utc_t ∗timestamp)
- int atcacert_date_enc_posix_uint32_le (const atcacert_tm_utc_t ∗timestamp, uint8_t formatted_date[DATEFMT_POSIX_UINT32_LE_SIZE])
- int atcacert_date_dec_posix_uint32_le (const uint8_t formatted_date[DATEFMT_POSIX_UINT32_LE_SIZE], atcacert_tm_utc_t ∗timestamp)

Variables

- const size_t ATCACERT_DATE_FORMAT_SIZES [ATCACERT_DATE_FORMAT_SIZES_COUNT]

15.72.1 Detailed Description

Declarations for date handling with regard to certificates.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.73 atcacert_def.c File Reference

Main certificate definition implementation.

```
#include "atcacert_def.h"
#include "crypto/atca_crypto_sw_sha1.h"
#include "crypto/atca_crypto_sw_sha2.h"
#include "atcacert_der.h"
#include "atcacert_date.h"
#include <string.h>
#include "basic/atca_helpers.h"
```

Macros

- #define ATCACERT_MIN(x, y) ((x) < (y) ? (x) : (y))
- #define ATCACERT_MAX(x, y) ((x) >= (y) ? (x) : (y))

Functions

- int atcacert_merge_device_loc (atcacert_device_loc_t *device_locs, size_t *device_locs_count, size_t device_locs_max_count, const atcacert_device_loc_t *device_loc, size_t block_size)

  Merge a new device location into a list of device locations. If the new location overlaps with an existing location, the existing one will be modified to encompass both. Otherwise the new location is appended to the end of the list.
- int atcacert_get_device_locs (const atcacert_def_t *cert_def, atcacert_device_loc_t *device_locs, size_t *device_locs_count, size_t device_locs_max_count, size_t block_size)

  Add all the device locations required to rebuild the specified certificate (cert_def) to a device locations list.
- int atcacert_cert_build_start (atcacert_build_state_t *build_state, const atcacert_def_t *cert_def, uint8_t *cert, size_t *cert_size, const uint8_t ca_public_key[64])

  Starts the certificate rebuilding process.
- int atcacert_cert_build_process (atcacert_build_state_t *build_state, const atcacert_device_loc_t *device_loc, const uint8_t *device_data)

  Process information read from the ATECC device. If it contains information for the certificate, it will be incorporated into the certificate.
- int atcacert_cert_build_finish (atcacert_build_state_t *build_state)

  Completes any final certificate processing required after all data from the device has been incorporated.

- int atcacert_is_device_loc_overlap (const atcacert_device_loc_t ∗device_loc1, const atcacert_device_loc_t ∗device_loc2)

  Determines if the two device locations overlap.
- int atcacert_get_device_data (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, const atcacert_device_loc_t ∗device_loc, uint8_t ∗device_data)

  Gets the dynamic data that would be saved to the specified device location. This function is primarily used to break down a full certificate into the dynamic components to be saved to a device.
- int atcacert_set_subj_public_key (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t cert_size, const uint8_t subj_public_key[64])

  Sets the subject public key and subject key ID in a certificate.
- int atcacert_get_subj_public_key (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_t subj_public_key[64])

  Gets the subject public key from a certificate.
- int atcacert_get_subj_key_id (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_t subj_key_id[20])

  Gets the subject key ID from a certificate.
- int atcacert_set_signature (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t ∗cert_size, size_t max_cert_size, const uint8_t signature[64])

  Sets the signature in a certificate. This may alter the size of the X.509 certificates.
- int atcacert_get_signature (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_t signature[64])

  Gets the signature from a certificate.
- int atcacert_set_issue_date (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t cert_size, const atcacert_tm_utc_t ∗timestamp)

  Sets the issue date (notBefore) in a certificate. Will be formatted according to the date format specified in the certificate definition.
- int atcacert_get_issue_date (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, atcacert_tm_utc_t ∗timestamp)

  Gets the issue date from a certificate. Will be parsed according to the date format specified in the certificate definition.
- int atcacert_set_expire_date (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t cert_size, const atcacert_tm_utc_t ∗timestamp)

  Sets the expire date (notAfter) in a certificate. Will be formatted according to the date format specified in the certificate definition.
- int atcacert_get_expire_date (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, atcacert_tm_utc_t ∗timestamp)

  Gets the expire date from a certificate. Will be parsed according to the date format specified in the certificate definition.
- int atcacert_set_signer_id (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t cert_size, const uint8_t signer_id[2])

  Sets the signer ID in a certificate. Will be formatted as 4 upper-case hex digits.
- int atcacert_get_signer_id (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_t signer_id[2])

  Gets the signer ID from a certificate. Will be parsed as 4 upper-case hex digits.
- int atcacert_set_cert_sn (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t ∗cert_size, size_t max_cert_size, const uint8_t ∗cert_sn, size_t cert_sn_size)

  Sets the certificate serial number in a certificate.
- int atcacert_gen_cert_sn (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t cert_size, const uint8_t device_sn[9])

  Sets the certificate serial number by generating it from other information in the certificate using the scheme specified by sn_source in cert_def. See the.
- int atcacert_get_cert_sn (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_t ∗cert_sn, size_t ∗cert_sn_size)

  Gets the certificate serial number from a certificate.
- int atcacert_set_auth_key_id (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t cert_size, const uint8_t auth_public_key[64])

Sets the authority key ID in a certificate. Note that this takes the actual public key creates a key ID from it.

- int atcacert_set_auth_key_id_raw (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t cert_size, const uint8_t ∗auth_key_id)

  Sets the authority key ID in a certificate.
- int atcacert_get_auth_key_id (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_t auth_key_id[20])

  Gets the authority key ID from a certificate.
- int atcacert_set_comp_cert (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t ∗cert_size, size_t max_↩ cert_size, const uint8_t comp_cert[72])

  Sets the signature, issue date, expire date, and signer ID found in the compressed certificate. This also checks fields common between the cert_def and the compressed certificate to make sure they match.
- int atcacert_get_comp_cert (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_↩ t comp_cert[72])

  Generate the compressed certificate for the given certificate.
- int atcacert_get_tbs (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, const uint8_t ∗∗tbs, size_t ∗tbs_size)

  Get a pointer to the TBS data in a certificate.
- int atcacert_get_tbs_digest (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_↩ t tbs_digest[32])

  Get the SHA256 digest of certificate's TBS data.
- int atcacert_set_cert_element (const atcacert_def_t ∗cert_def, const atcacert_cert_loc_t ∗cert_loc, uint8_t ∗cert, size_t cert_size, const uint8_t ∗data, size_t data_size)

  Sets an element in a certificate. The data_size must match the size in cert_loc.
- int atcacert_get_cert_element (const atcacert_def_t ∗cert_def, const atcacert_cert_loc_t ∗cert_loc, const uint8_t ∗cert, size_t cert_size, uint8_t ∗data, size_t data_size)

  Gets an element from a certificate.
- int atcacert_get_key_id (const uint8_t public_key[64], uint8_t key_id[20])

  Calculates the key ID for a given public ECC P256 key.
- void atcacert_public_key_add_padding (const uint8_t raw_key[64], uint8_t padded_key[72])

  Takes a raw P256 ECC public key and converts it to the padded version used by ATECC devices. Input and output buffers can point to the same location to do an in-place transform.
- void atcacert_public_key_remove_padding (const uint8_t padded_key[72], uint8_t raw_key[64])

  Takes a padded public key used by ATECC devices and converts it to a raw P256 ECC public key. Input and output buffers can point to the same location to do an in-place transform.
- int atcacert_transform_data (atcacert_transform_t transform, const uint8_t ∗data, size_t data_size, uint8_t ∗destination, size_t ∗destination_size)

  Apply the specified transform to the specified data.
- int atcacert_max_cert_size (const atcacert_def_t ∗cert_def, size_t ∗max_cert_size)

  Return the maximum possible certificate size in bytes for a given cert def. Certificate can be variable size, so this gives an appropriate buffer size when reading the certificates.

15.73.1 Detailed Description

Main certificate definition implementation.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.73.2 Macro Definition Documentation

15.73.2.1 ATCACERT_MAX

```
#define ATCACERT_MAX(
              x,
              y ) ((x) >= (y) ? (x) : (y))
```

15.73.2.2 ATCACERT_MIN

```
#define ATCACERT_MIN(
              x,
              y ) ((x) < (y) ? (x) : (y))
```

15.74 atcacert_def.h File Reference

Declarations for certificates related to ECC CryptoAuthentication devices. These are the definitions required to define a certificate and its various elements with regards to the CryptoAuthentication ECC devices.

```
#include <stddef.h>
#include <stdint.h>
#include "atca_compiler.h"
#include "atcacert.h"
#include "atcacert_date.h"
#include "basic/atca_helpers.h"
```

Data Structures

- struct atcacert_device_loc_s
- struct atcacert_cert_loc_s
- struct atcacert_cert_element_s
- struct atcacert_def_s
- struct atcacert_build_state_s

Macros

- #define ATCA_MAX_TRANSFORMS 2

Typedefs

- typedef enum atcacert_cert_type_e atcacert_cert_type_t
- typedef enum atcacert_cert_sn_src_e atcacert_cert_sn_src_t
- typedef enum atcacert_device_zone_e atcacert_device_zone_t
- typedef enum atcacert_transform_e atcacert_transform_t

  How to transform the data from the device to the certificate.
- typedef enum atcacert_std_cert_element_e atcacert_std_cert_element_t
- typedef struct atcacert_device_loc_s atcacert_device_loc_t
- typedef struct atcacert_cert_loc_s atcacert_cert_loc_t
- typedef struct atcacert_cert_element_s atcacert_cert_element_t
- typedef struct atcacert_def_s atcacert_def_t
- typedef struct atcacert_build_state_s atcacert_build_state_t

Enumerations

- enum atcacert_cert_type_e { CERTTYPE_X509, CERTTYPE_CUSTOM }
- enum atcacert_cert_sn_src_e {
SNSRC_STORED = 0x0, SNSRC_STORED_DYNAMIC = 0x7, SNSRC_DEVICE_SN = 0x8, SNSRC_SIGNER_ID = 0x9,
SNSRC_PUB_KEY_HASH = 0xA, SNSRC_DEVICE_SN_HASH = 0xB, SNSRC_PUB_KEY_HASH_POS = 0xC, SNSRC_DEVICE_SN_HASH_POS = 0xD,
SNSRC_PUB_KEY_HASH_RAW = 0xE, SNSRC_DEVICE_SN_HASH_RAW = 0xF }
- enum atcacert_device_zone_e { DEVZONE_CONFIG = 0x00, DEVZONE_OTP = 0x01, DEVZONE_DATA = 0x02, DEVZONE_NONE = 0x07 }
- enum atcacert_transform_e {
TF_NONE, TF_REVERSE, TF_BIN2HEX_UC, TF_BIN2HEX_LC,
TF_HEX2BIN_UC, TF_HEX2BIN_LC, TF_BIN2HEX_SPACE_UC, TF_BIN2HEX_SPACE_LC,
TF_HEX2BIN_SPACE_UC, TF_HEX2BIN_SPACE_LC }

  How to transform the data from the device to the certificate.
- enum atcacert_std_cert_element_e {
STDCERT_PUBLIC_KEY, STDCERT_SIGNATURE, STDCERT_ISSUE_DATE, STDCERT_EXPIRE_DATE,
STDCERT_SIGNER_ID, STDCERT_CERT_SN, STDCERT_AUTH_KEY_ID, STDCERT_SUBJ_KEY_ID,
STDCERT_NUM_ELEMENTS }

Functions

- int atcacert_get_device_locs (const atcacert_def_t ∗cert_def, atcacert_device_loc_t ∗device_locs, size_↩ t ∗device_locs_count, size_t device_locs_max_count, size_t block_size)

  Add all the device locations required to rebuild the specified certificate (cert_def) to a device locations list.
- int atcacert_cert_build_start (atcacert_build_state_t ∗build_state, const atcacert_def_t ∗cert_def, uint8_↩ t ∗cert, size_t ∗cert_size, const uint8_t ca_public_key[64])

  Starts the certificate rebuilding process.
- int atcacert_cert_build_process (atcacert_build_state_t ∗build_state, const atcacert_device_loc_t ∗device↩ _loc, const uint8_t ∗device_data)

  Process information read from the ATECC device. If it contains information for the certificate, it will be incorporated into the certificate.
- int atcacert_cert_build_finish (atcacert_build_state_t ∗build_state)

  Completes any final certificate processing required after all data from the device has been incorporated.
- int atcacert_get_device_data (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, const atcacert_device_loc_t ∗device_loc, uint8_t ∗device_data)

  Gets the dynamic data that would be saved to the specified device location. This function is primarily used to break down a full certificate into the dynamic components to be saved to a device.
- int atcacert_set_subj_public_key (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t cert_size, const uint8_t subj_public_key[64])

  Sets the subject public key and subject key ID in a certificate.
- int atcacert_get_subj_public_key (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_t subj_public_key[64])

  Gets the subject public key from a certificate.
- int atcacert_get_subj_key_id (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_t subj_key_id[20])

  Gets the subject key ID from a certificate.
- int atcacert_set_signature (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t ∗cert_size, size_t max_↩ cert_size, const uint8_t signature[64])

  Sets the signature in a certificate. This may alter the size of the X.509 certificates.
- int atcacert_get_signature (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_↩ t signature[64])

  Gets the signature from a certificate.

- int atcacert_set_issue_date (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t cert_size, const atcacert_tm_utc_t ∗timestamp)

  Sets the issue date (notBefore) in a certificate. Will be formatted according to the date format specified in the certificate definition.
- int atcacert_get_issue_date (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, atcacert_tm_utc_t ∗timestamp)

  Gets the issue date from a certificate. Will be parsed according to the date format specified in the certificate definition.
- int atcacert_set_expire_date (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t cert_size, const atcacert_tm_utc_t ∗timestamp)

  Sets the expire date (notAfter) in a certificate. Will be formatted according to the date format specified in the certificate definition.
- int atcacert_get_expire_date (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, atcacert_tm_utc_t ∗timestamp)

  Gets the expire date from a certificate. Will be parsed according to the date format specified in the certificate definition.
- int atcacert_set_signer_id (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t cert_size, const uint8_t signer_id[2])

  Sets the signer ID in a certificate. Will be formatted as 4 upper-case hex digits.
- int atcacert_get_signer_id (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_t signer_id[2])

  Gets the signer ID from a certificate. Will be parsed as 4 upper-case hex digits.
- int atcacert_set_cert_sn (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t ∗cert_size, size_t max_cert_size, const uint8_t ∗cert_sn, size_t cert_sn_size)

  Sets the certificate serial number in a certificate.
- int atcacert_gen_cert_sn (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t cert_size, const uint8_t device_sn[9])

  Sets the certificate serial number by generating it from other information in the certificate using the scheme specified by sn_source in cert_def. See the.
- int atcacert_get_cert_sn (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_t ∗cert_sn, size_t ∗cert_sn_size)

  Gets the certificate serial number from a certificate.
- int atcacert_set_auth_key_id (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t cert_size, const uint8_t auth_public_key[64])

  Sets the authority key ID in a certificate. Note that this takes the actual public key creates a key ID from it.
- int atcacert_set_auth_key_id_raw (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t cert_size, const uint8_t ∗auth_key_id)

  Sets the authority key ID in a certificate.
- int atcacert_get_auth_key_id (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_t auth_key_id[20])

  Gets the authority key ID from a certificate.
- int atcacert_set_comp_cert (const atcacert_def_t ∗cert_def, uint8_t ∗cert, size_t ∗cert_size, size_t max_cert_size, const uint8_t comp_cert[72])

  Sets the signature, issue date, expire date, and signer ID found in the compressed certificate. This also checks fields common between the cert_def and the compressed certificate to make sure they match.
- int atcacert_get_comp_cert (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_t comp_cert[72])

  Generate the compressed certificate for the given certificate.
- int atcacert_get_tbs (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, const uint8_t ∗∗tbs, size_t ∗tbs_size)

  Get a pointer to the TBS data in a certificate.
- int atcacert_get_tbs_digest (const atcacert_def_t ∗cert_def, const uint8_t ∗cert, size_t cert_size, uint8_t tbs_digest[32])

  Get the SHA256 digest of certificate's TBS data.
- int atcacert_set_cert_element (const atcacert_def_t ∗cert_def, const atcacert_cert_loc_t ∗cert_loc, uint8_t ∗cert, size_t cert_size, const uint8_t ∗data, size_t data_size)

Sets an element in a certificate. The data_size must match the size in cert_loc.

- int atcacert_get_cert_element (const atcacert_def_t ∗cert_def, const atcacert_cert_loc_t ∗cert_loc, const uint8_t ∗cert, size_t cert_size, uint8_t ∗data, size_t data_size)

  Gets an element from a certificate.
- int atcacert_get_key_id (const uint8_t public_key[64], uint8_t key_id[20])

  Calculates the key ID for a given public ECC P256 key.
- int atcacert_merge_device_loc (atcacert_device_loc_t ∗device_locs, size_t ∗device_locs_count, size_←t device_locs_max_count, const atcacert_device_loc_t ∗device_loc, size_t block_size)

  Merge a new device location into a list of device locations. If the new location overlaps with an existing location, the existing one will be modified to encompass both. Otherwise the new location is appended to the end of the list.
- int atcacert_is_device_loc_overlap (const atcacert_device_loc_t ∗device_loc1, const atcacert_device_loc_t ∗device_loc2)

  Determines if the two device locations overlap.
- void atcacert_public_key_add_padding (const uint8_t raw_key[64], uint8_t padded_key[72])

  Takes a raw P256 ECC public key and converts it to the padded version used by ATECC devices. Input and output buffers can point to the same location to do an in-place transform.
- void atcacert_public_key_remove_padding (const uint8_t padded_key[72], uint8_t raw_key[64])

  Takes a padded public key used by ATECC devices and converts it to a raw P256 ECC public key. Input and output buffers can point to the same location to do an in-place transform.
- int atcacert_transform_data (atcacert_transform_t transform, const uint8_t ∗data, size_t data_size, uint8_t ∗destination, size_t ∗destination_size)

  Apply the specified transform to the specified data.
- int atcacert_max_cert_size (const atcacert_def_t ∗cert_def, size_t ∗max_cert_size)

  Return the maximum possible certificate size in bytes for a given cert def. Certificate can be variable size, so this gives an appropriate buffer size when reading the certificates.

15.74.1 Detailed Description

Declarations for certificates related to ECC CryptoAuthentication devices. These are the definitions required to define a certificate and its various elements with regards to the CryptoAuthentication ECC devices.

Only the dynamic elements of a certificate (the parts of the certificate that change from device to device) are stored on the ATECC device. The definitions here describe the form of the certificate, and where the dynamic elements can be found both on the ATECC device itself and in the certificate template.

This also defines utility functions for working with the certificates and their definitions.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.74.2 Macro Definition Documentation

15.74.2.1 ATCA_MAX_TRANSFORMS

```
#define ATCA_MAX_TRANSFORMS 2
```

15.75 atcacert_der.c File Reference

functions required to work with DER encoded data related to X.509 certificates.

```
#include "atcacert_der.h"
#include <string.h>
```

Functions

- int atcacert_der_enc_length (uint32_t length, uint8_t ∗der_length, size_t ∗der_length_size)

  Encode a length in DER format.
- int atcacert_der_dec_length (const uint8_t ∗der_length, size_t ∗der_length_size, uint32_t ∗length)

  Decode a DER format length.
- int atcacert_der_adjust_length (uint8_t ∗der_length, size_t ∗der_length_size, int delta_length, uint32_t ∗new_length)
- int atcacert_der_enc_integer (const uint8_t ∗int_data, size_t int_data_size, uint8_t is_unsigned, uint8_t ∗der_int, size_t ∗der_int_size)

  Encode an ASN.1 integer in DER format, including tag and length fields.
- int atcacert_der_dec_integer (const uint8_t ∗der_int, size_t ∗der_int_size, uint8_t ∗int_data, size_t ∗int_data_size)

  Decode an ASN.1 DER encoded integer.
- int atcacert_der_enc_ecdsa_sig_value (const uint8_t raw_sig[64], uint8_t ∗der_sig, size_t ∗der_sig_size)

  Formats a raw ECDSA P256 signature in the DER encoding found in X.509 certificates.
- int atcacert_der_dec_ecdsa_sig_value (const uint8_t ∗der_sig, size_t ∗der_sig_size, uint8_t raw_sig[64])

  Parses an ECDSA P256 signature in the DER encoding as found in X.509 certificates.

15.75.1 Detailed Description

functions required to work with DER encoded data related to X.509 certificates.

Copyright

15.76 atcacert_der.h File Reference

function declarations required to work with DER encoded data related to X.509 certificates.

```
#include <stddef.h>
#include <stdint.h>
#include "atcacert.h"
```

Functions

- int atcacert_der_enc_length (uint32_t length, uint8_t *der_length, size_t *der_length_size)

  Encode a length in DER format.
- int atcacert_der_dec_length (const uint8_t *der_length, size_t *der_length_size, uint32_t *length)

  Decode a DER format length.
- int atcacert_der_adjust_length (uint8_t *der_length, size_t *der_length_size, int delta_length, uint32_t *new_length)
- int atcacert_der_enc_integer (const uint8_t *int_data, size_t int_data_size, uint8_t is_unsigned, uint8_t *der_int, size_t *der_int_size)

  Encode an ASN.1 integer in DER format, including tag and length fields.
- int atcacert_der_dec_integer (const uint8_t *der_int, size_t *der_int_size, uint8_t *int_data, size_t *int_data_size)

  Decode an ASN.1 DER encoded integer.
- int atcacert_der_enc_ecdsa_sig_value (const uint8_t raw_sig[64], uint8_t *der_sig, size_t *der_sig_size)

  Formats a raw ECDSA P256 signature in the DER encoding found in X.509 certificates.
- int atcacert_der_dec_ecdsa_sig_value (const uint8_t *der_sig, size_t *der_sig_size, uint8_t raw_sig[64])

  Parses an ECDSA P256 signature in the DER encoding as found in X.509 certificates.

15.76.1 Detailed Description

function declarations required to work with DER encoded data related to X.509 certificates.

Copyright

15.77 atcacert_host_hw.c File Reference

host side methods using CryptoAuth hardware

```
#include "atcacert_host_hw.h"
#include "basic/atca_basic.h"
#include "crypto/atca_crypto_sw_sha2.h"
```

Functions

- int atcacert_verify_cert_hw (const atcacert_def_t *cert_def, const uint8_t *cert, size_t cert_size, const uint8_t ca_public_key[64])

  Verify a certificate against its certificate authority's public key using the host's ATECC device for crypto functions.
- int atcacert_gen_challenge_hw (uint8_t challenge[32])

  Generate a random challenge to be sent to the client using the RNG on the host's ATECC device.
- int atcacert_verify_response_hw (const uint8_t device_public_key[64], const uint8_t challenge[32], const uint8_t response[64])

  Verify a client's response to a challenge using the host's ATECC device for crypto functions.

15.77.1 Detailed Description

host side methods using CryptoAuth hardware

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.78 atcacert_host_hw.h File Reference

host side methods using CryptoAuth hardware

```
#include <stddef.h>
#include <stdint.h>
#include "atcacert_def.h"
```

Functions

- int atcacert_verify_cert_hw (const atcacert_def_t *cert_def, const uint8_t *cert, size_t cert_size, const uint8_t ca_public_key[64])

  Verify a certificate against its certificate authority's public key using the host's ATECC device for crypto functions.
- int atcacert_gen_challenge_hw (uint8_t challenge[32])

  Generate a random challenge to be sent to the client using the RNG on the host's ATECC device.
- int atcacert_verify_response_hw (const uint8_t device_public_key[64], const uint8_t challenge[32], const uint8_t response[64])

  Verify a client's response to a challenge using the host's ATECC device for crypto functions.

15.78.1 Detailed Description

host side methods using CryptoAuth hardware

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.79 atcacert_host_sw.c File Reference

host side methods using software implementations

```
#include "atcacert_host_sw.h"
#include "crypto/atca_crypto_sw_sha2.h"
#include "crypto/atca_crypto_sw_ecdsa.h"
#include "crypto/atca_crypto_sw_rand.h"
```

Functions

- int atcacert_verify_cert_sw (const atcacert_def_t *cert_def, const uint8_t *cert, size_t cert_size, const uint8_t ca_public_key[64])

  Verify a certificate against its certificate authority's public key using software crypto functions.The function is currently not implemented.
- int atcacert_gen_challenge_sw (uint8_t challenge[32])

  Generate a random challenge to be sent to the client using a software PRNG.The function is currently not implemented.
- int atcacert_verify_response_sw (const uint8_t device_public_key[64], const uint8_t challenge[32], const uint8_t response[64])

  Verify a client's response to a challenge using software crypto functions.The function is currently not implemented.

15.79.1 Detailed Description

host side methods using software implementations

Copyright

15.80 atcacert_host_sw.h File Reference

Host side methods using software implementations. host-side, the one authenticating a client, of the authentication process. Crypto functions are performed using a software library.

```
#include <stddef.h>
#include <stdint.h>
#include "atcacert_def.h"
```

Functions

- int atcacert_verify_cert_sw (const atcacert_def_t *cert_def, const uint8_t *cert, size_t cert_size, const uint8_t ca_public_key[64])

  Verify a certificate against its certificate authority's public key using software crypto functions.The function is currently not implemented.
- int atcacert_gen_challenge_sw (uint8_t challenge[32])

  Generate a random challenge to be sent to the client using a software PRNG.The function is currently not implemented.
- int atcacert_verify_response_sw (const uint8_t device_public_key[64], const uint8_t challenge[32], const uint8_t response[64])

  Verify a client's response to a challenge using software crypto functions.The function is currently not implemented.

15.80.1 Detailed Description

Host side methods using software implementations. host-side, the one authenticating a client, of the authentication process. Crypto functions are performed using a software library.

Copyright

15.81 atcacert_pem.c File Reference

```
#include "atcacert.h"
#include "atcacert_pem.h"
#include "../basic/atca_helpers.h"
```

Functions

- int atcacert_encode_pem (const uint8_t ∗der, size_t der_size, char ∗pem, size_t ∗pem_size, const char ∗header, const char ∗footer)

  Encode a DER data in PEM format.
- int atcacert_decode_pem (const char ∗pem, size_t pem_size, uint8_t ∗der, size_t ∗der_size, const char ∗header, const char ∗footer)

  Decode PEM data into DER format.
- int atcacert_encode_pem_cert (const uint8_t ∗der_cert, size_t der_cert_size, char ∗pem_cert, size_t ∗pem_cert_size)

  Encode a DER certificate in PEM format.
- int atcacert_encode_pem_csr (const uint8_t ∗der_csr, size_t der_csr_size, char ∗pem_csr, size_t ∗pem_csr_size)

  Encode a DER CSR in PEM format.
- int atcacert_decode_pem_cert (const char ∗pem_cert, size_t pem_cert_size, uint8_t ∗der_cert, size_t ∗der_cert_size)

  Decode a PEM certificate into DER format.
- int atcacert_decode_pem_csr (const char ∗pem_csr, size_t pem_csr_size, uint8_t ∗der_csr, size_t ∗der_csr_size)

  Extract the CSR certificate bytes from a PEM encoded CSR certificate.

15.81.1 Function Documentation

15.81.1.1 atcacert_decode_pem()

```
int atcacert_decode_pem (
            const char * pem,
            size_t pem_size,
            uint8_t * der,
            size_t * der_size,
            const char * header,
            const char * footer )
```

Decode PEM data into DER format.

Parameters

in	*pem*	PEM data to decode to DER.
in	*pem_size*	PEM data size in bytes.
out	*der*	DER data is returned here.
in,out	*der_size*	As input, the size of the der buffer. As output, the size of the DER data.
in	*header*	Header to find the beginning of the PEM data.
in	*footer*	Footer to find the end of the PEM data.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.81.1.2 atcacert_decode_pem_cert()

```
int atcacert_decode_pem_cert (
            const char * pem_cert,
            size_t pem_cert_size,
            uint8_t * der_cert,
            size_t * der_cert_size )
```

Decode a PEM certificate into DER format.

Parameters

in	*pem_cert*	PEM certificate to decode to DER.
in	*pem_cert_size*	PEM certificate size in bytes.
out	*der_cert*	DER certificate is returned here.
in,out	*der_cert_size*	As input, the size of the der_cert buffer. As output, the size of the DER certificate.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.81.1.3 atcacert_decode_pem_csr()

```
int atcacert_decode_pem_csr (
            const char * pem_csr,
            size_t pem_csr_size,
            uint8_t * der_csr,
            size_t * der_csr_size )
```

Extract the CSR certificate bytes from a PEM encoded CSR certificate.

Parameters

in	*pem_csr*	PEM CSR to decode to DER.
in	*pem_csr_size*	PEM CSR size in bytes.
out	*der_csr*	DER CSR is returned here.
in,out	*der_csr_size*	As input, the size of the der_csr buffer. As output, the size of the DER CSR.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.81.1.4 atcacert_encode_pem()

```
int atcacert_encode_pem (
            const uint8_t * der,
            size_t der_size,
            char * pem,
            size_t * pem_size,
            const char * header,
            const char * footer )
```

Encode a DER data in PEM format.

Parameters

in	*der*	DER data to be encoded as PEM.
out	*der_size*	DER data size in bytes.
out	*pem*	PEM encoded data is returned here.
in,out	*pem_size*	As input, the size of the pem buffer. As output, the size of the PEM data.
in	*header*	Header to place at the beginning of the PEM data.
in	*footer*	Footer to place at the end of the PEM data.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.81.1.5 atcacert_encode_pem_cert()

```
int atcacert_encode_pem_cert (
            const uint8_t * der_cert,
            size_t der_cert_size,
            char * pem_cert,
            size_t * pem_cert_size )
```

Encode a DER certificate in PEM format.

Parameters

in	*der_cert*	DER certificate to be encoded as PEM.
out	*der_cert_size*	DER certificate size in bytes.
out	*pem_cert*	PEM encoded certificate is returned here.
in,out	*pem_cert_size*	As input, the size of the pem_cert buffer. As output, the size of the PEM certificate.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.81.1.6 atcacert_encode_pem_csr()

```
int atcacert_encode_pem_csr (
            const uint8_t * der_csr,
            size_t der_csr_size,
            char * pem_csr,
            size_t * pem_csr_size )
```

Encode a DER CSR in PEM format.

Parameters

in	*der_csr*	DER CSR to be encoded as PEM.
out	*der_csr_size*	DER CSR size in bytes.
out	*pem_csr*	PEM encoded CSR is returned here.
in,out	*pem_csr_size*	As input, the size of the pem_csr buffer. As output, the size of the PEM CSR.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.82 atcacert_pem.h File Reference

Functions for converting between DER and PEM formats.

```
#include <stdint.h>
```

Macros

- #define PEM_CERT_BEGIN "-----BEGIN CERTIFICATE-----"
- #define PEM_CERT_END "-----END CERTIFICATE-----"
- #define PEM_CSR_BEGIN "-----BEGIN CERTIFICATE REQUEST-----"
- #define PEM_CSR_END "-----END CERTIFICATE REQUEST-----"

Functions

- int atcacert_encode_pem (const uint8_t ∗der, size_t der_size, char ∗pem, size_t ∗pem_size, const char ∗header, const char ∗footer)

  Encode a DER data in PEM format.
- int atcacert_decode_pem (const char ∗pem, size_t pem_size, uint8_t ∗der, size_t ∗der_size, const char ∗header, const char ∗footer)

  Decode PEM data into DER format.
- int atcacert_encode_pem_cert (const uint8_t ∗der_cert, size_t der_cert_size, char ∗pem_cert, size_t ∗pem_cert_size)

  Encode a DER certificate in PEM format.
- int atcacert_decode_pem_cert (const char ∗pem_cert, size_t pem_cert_size, uint8_t ∗der_cert, size_t ∗der_cert_size)

  Decode a PEM certificate into DER format.
- int atcacert_encode_pem_csr (const uint8_t ∗der_csr, size_t der_csr_size, char ∗pem_csr, size_t ∗pem_csr_size)

  Encode a DER CSR in PEM format.
- int atcacert_decode_pem_csr (const char ∗pem_csr, size_t pem_csr_size, uint8_t ∗der_csr, size_t ∗der_csr_size)

  Extract the CSR certificate bytes from a PEM encoded CSR certificate.

15.82.1 Detailed Description

Functions for converting between DER and PEM formats.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.82.2 Macro Definition Documentation

15.82.2.1 PEM_CERT_BEGIN

```
#define PEM_CERT_BEGIN "-----BEGIN CERTIFICATE-----"
```

15.82.2.2 PEM_CERT_END

```
#define PEM_CERT_END "-----END CERTIFICATE-----"
```

15.82.2.3 PEM_CSR_BEGIN

```
#define PEM_CSR_BEGIN "-----BEGIN CERTIFICATE REQUEST-----"
```

15.82.2.4 PEM_CSR_END

```
#define PEM_CSR_END "-----END CERTIFICATE REQUEST-----"
```

15.82.3 Function Documentation

15.82.3.1 atcacert_decode_pem()

```
int atcacert_decode_pem (
            const char * pem,
            size_t pem_size,
            uint8_t * der,
            size_t * der_size,
            const char * header,
            const char * footer )
```

Decode PEM data into DER format.

Parameters

in	*pem*	PEM data to decode to DER.
in	*pem_size*	PEM data size in bytes.
out	*der*	DER data is returned here.
in,out	*der_size*	As input, the size of the der buffer. As output, the size of the DER data.
in	*header*	Header to find the beginning of the PEM data.
in	*footer*	Footer to find the end of the PEM data.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.82.3.2 atcacert_decode_pem_cert()

```
int atcacert_decode_pem_cert (
            const char * pem_cert,
            size_t pem_cert_size,
            uint8_t * der_cert,
            size_t * der_cert_size )
```

Decode a PEM certificate into DER format.

Parameters

in	*pem_cert*	PEM certificate to decode to DER.
in	*pem_cert_size*	PEM certificate size in bytes.
out	*der_cert*	DER certificate is returned here.
in,out	*der_cert_size*	As input, the size of the der_cert buffer. As output, the size of the DER certificate.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.82.3.3 atcacert_decode_pem_csr()

```
int atcacert_decode_pem_csr (
            const char * pem_csr,
            size_t pem_csr_size,
            uint8_t * der_csr,
            size_t * der_csr_size )
```

Extract the CSR certificate bytes from a PEM encoded CSR certificate.

Parameters

in	*pem_csr*	PEM CSR to decode to DER.
in	*pem_csr_size*	PEM CSR size in bytes.
out	*der_csr*	DER CSR is returned here.
in,out	*der_csr_size*	As input, the size of the der_csr buffer. As output, the size of the DER CSR.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.82.3.4 atcacert_encode_pem()

```
int atcacert_encode_pem (
            const uint8_t * der,
            size_t der_size,
            char * pem,
            size_t * pem_size,
            const char * header,
            const char * footer )
```

Encode a DER data in PEM format.

Parameters

in	*der*	DER data to be encoded as PEM.
out	*der_size*	DER data size in bytes.
out	*pem*	PEM encoded data is returned here.
in,out	*pem_size*	As input, the size of the pem buffer. As output, the size of the PEM data.
in	*header*	Header to place at the beginning of the PEM data.
in	*footer*	Footer to place at the end of the PEM data.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.82.3.5 atcacert_encode_pem_cert()

```
int atcacert_encode_pem_cert (
            const uint8_t * der_cert,
            size_t der_cert_size,
            char * pem_cert,
            size_t * pem_cert_size )
```

Encode a DER certificate in PEM format.

Parameters

in	*der_cert*	DER certificate to be encoded as PEM.
out	*der_cert_size*	DER certificate size in bytes.
out	*pem_cert*	PEM encoded certificate is returned here.
in,out	*pem_cert_size*	As input, the size of the pem_cert buffer. As output, the size of the PEM certificate.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.82.3.6 atcacert_encode_pem_csr()

```
int atcacert_encode_pem_csr (
            const uint8_t * der_csr,
            size_t der_csr_size,
            char * pem_csr,
            size_t * pem_csr_size )
```

Encode a DER CSR in PEM format.

Parameters

in	*der_csr*	DER CSR to be encoded as PEM.
out	*der_csr_size*	DER CSR size in bytes.
out	*pem_csr*	PEM encoded CSR is returned here.
in,out	*pem_csr_size*	As input, the size of the pem_csr buffer. As output, the size of the PEM CSR.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.83 cryptoauthlib.h File Reference

Single aggregation point for all CryptoAuthLib header files.

```
#include <stddef.h>
#include <string.h>
#include "hal/atca_hal.h"
#include "atca_status.h"
#include "atca_device.h"
#include "atca_command.h"
#include "atca_cfgs.h"
#include "basic/atca_basic.h"
#include "basic/atca_helpers.h"
```

Macros

- #define BREAK(status, message) { break; }
- #define RETURN(status, message) { return status; }
- #define PRINT(message) { break; }
- #define DBGOUT(message) { break; }

15.83.1 Detailed Description

Single aggregation point for all CryptoAuthLib header files.

Copyright

15.83.2 Macro Definition Documentation

15.83.2.1 BREAK

```
#define BREAK(
              status,
              message ) { break; }
```

15.83.2.2 DBGOUT

```
#define DBGOUT(
              message ) { break; }
```

15.83.2.3 PRINT

```
#define PRINT(
              message ) { break; }
```

15.83.2.4 RETURN

```
#define RETURN(
              status,
              message ) { return status; }
```

15.84 hal_all_platforms_kit_hidapi.c File Reference

HAL for kit protocol over HID for any platform.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "hidapi.h"
#include "atca_hal.h"
#include "hal_all_platforms_kit_hidapi.h"
#include "hal/kit_protocol.h"
```

Functions

- ATCA_STATUS hal_kit_hid_discover_buses (int i2c_buses[ ], int max_buses)

  discover cdc buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge
- ATCA_STATUS hal_kit_hid_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

  discover any CryptoAuth devices on a given logical bus number
- ATCA_STATUS hal_kit_hid_init (void ∗hal, ATCAIfaceCfg ∗cfg)

  HAL implementation of Kit USB HID init.
- ATCA_STATUS hal_kit_hid_post_init (ATCAIface iface)

  HAL implementation of Kit HID post init.
- ATCA_STATUS kit_phy_send (ATCAIface iface, uint8_t ∗txdata, int txlength)

  HAL implementation of send over USB HID.
- ATCA_STATUS kit_phy_receive (ATCAIface iface, uint8_t ∗rxdata, int ∗rxsize)

  HAL implementation of kit protocol send over USB HID.
- ATCA_STATUS kit_phy_num_found (int8_t ∗num_found)

  Number of USB HID devices found.
- ATCA_STATUS hal_kit_hid_send (ATCAIface iface, uint8_t ∗txdata, int txlength)

  HAL implementation of kit protocol send over USB HID.
- ATCA_STATUS hal_kit_hid_receive (ATCAIface iface, uint8_t ∗rxdata, uint16_t ∗rxsize)

  HAL implementation of send over USB HID.
- ATCA_STATUS hal_kit_hid_wake (ATCAIface iface)

  Call the wake for kit protocol.
- ATCA_STATUS hal_kit_hid_idle (ATCAIface iface)

  Call the idle for kit protocol.
- ATCA_STATUS hal_kit_hid_sleep (ATCAIface iface)

  Call the sleep for kit protocol.
- ATCA_STATUS hal_kit_hid_release (void ∗hal_data)

  Close the physical port for HID.

Variables

- atcahid_t _gHid

15.84.1 Detailed Description

HAL for kit protocol over HID for any platform.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.85 hal_all_platforms_kit_hidapi.h File Reference

HAL for kit protocol over HID for any platform.

```
#include "hidapi.h"
```

Data Structures

- struct atcahid

Macros

- #define HID_DEVICES_MAX 10
- #define HID_PACKET_MAX 512

Typedefs

- typedef struct atcahid atcahid_t

15.85.1 Detailed Description

HAL for kit protocol over HID for any platform.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.86 hal_at90usb1287_i2c_asf.c File Reference

ATCA Hardware abstraction layer for AT90USB1287 I2C over ASF drivers.

```
#include <asf.h>
#include <string.h>
#include <stdio.h>
#include "atca_hal.h"
#include "hal_at90usb1287_i2c_asf.h"
#include "atca_device.h"
#include "atca_execution.h"
```

Functions

- ATCA_STATUS hal_i2c_discover_buses (int i2c_buses[ ], int max_buses)

    discover i2c buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the prior knowledge
- ATCA_STATUS hal_i2c_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

    discover any CryptoAuth devices on a given logical bus number
- ATCA_STATUS hal_i2c_init (void ∗hal, ATCAIfaceCfg ∗cfg)

    initialize an I2C interface using given config
- ATCA_STATUS hal_i2c_post_init (ATCAIface iface)

    HAL implementation of I2C post init.
- ATCA_STATUS hal_i2c_send (ATCAIface iface, uint8_t ∗txdata, int txlength)

    HAL implementation of I2C send over ASF.
- ATCA_STATUS hal_i2c_receive (ATCAIface iface, uint8_t ∗rxdata, uint16_t ∗rxlength)

    HAL implementation of I2C receive function for ASF I2C.
- void change_i2c_speed (ATCAIface iface, uint32_t speed)

    method to change the bus speed of I2C
- ATCA_STATUS hal_i2c_wake (ATCAIface iface)

    wake up CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_idle (ATCAIface iface)

    idle CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_sleep (ATCAIface iface)

    sleep CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_release (void ∗hal_data)

    manages reference count on given bus and releases resource if no more refences exist

15.86.1 Detailed Description

ATCA Hardware abstraction layer for AT90USB1287 I2C over ASF drivers.

Prerequisite: Add I2C Master Polled/Interrupt support to application in Atmel Studio this HAL implementation assumes you've included the ASF I2C libraries in your project, otherwise, the HAL layer will not compile because the ASF I2C drivers are a dependency

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.87 hal_at90usb1287_i2c_asf.h File Reference

ATCA Hardware abstraction layer for AT90USB1287 I2C over ASF drivers.

```
#include <asf.h>
#include "twi_megarf.h"
```

Data Structures

- struct atcaI2Cmaster

    this is the hal_data for ATCA HAL created using ASF

Macros

- #define MAX_I2C_BUSES 1

Typedefs

- typedef struct atcaI2Cmaster ATCAI2CMaster_t

  this is the hal_data for ATCA HAL created using ASF

Functions

- void change_i2c_speed (ATCAIface iface, uint32_t speed)

  method to change the bus speed of I2C

15.87.1 Detailed Description

ATCA Hardware abstraction layer for AT90USB1287 I2C over ASF drivers.

Prerequisite: add I2C Master Polled support to application in Atmel Studio

Copyright

15.88 hal_at90usb1287_timer_asf.c File Reference

ATCA Hardware abstraction layer for AT90USB1287 timer/delay over ASF drivers.

```
#include <asf.h>
#include <delay.h>
#include "atca_hal.h"
```

Functions

- void atca_delay_us (uint32_t delay)

  Timer API implemented at the HAL level.
- void atca_delay_10us (uint32_t delay)

  This function delays for a number of tens of microseconds.
- void atca_delay_ms (uint32_t delay)

  This function delays for a number of milliseconds.

15.88.1 Detailed Description

ATCA Hardware abstraction layer for AT90USB1287 timer/delay over ASF drivers.

Copyright

15.89 hal_esp32_i2c.c File Reference

```
#include <stdio.h>
#include <string.h>
#include <driver/i2c.h>
#include "hal/atca_hal.h"
#include "esp_err.h"
#include "esp_log.h"
```

Data Structures

- struct atcaI2Cmaster

  this is the hal_data for ATCA HAL created using ASF

Macros

- #define SDA_PIN 16
- #define SCL_PIN 17
- #define ACK_CHECK_EN 0x1
- #define ACK_CHECK_DIS 0x0
- #define ACK_VAL 0x0
- #define NACK_VAL 0x1
- #define LOG_LOCAL_LEVEL ESP_LOG_INFO
- #define MAX_I2C_BUSES 2

Typedefs

- typedef struct atcaI2Cmaster ATCAI2CMaster_t

Functions

- void hal_i2c_change_baud (ATCAIface iface, uint32_t speed)
- ATCA_STATUS hal_i2c_init (void ∗hal, ATCAIfaceCfg ∗cfg)
- ATCA_STATUS hal_i2c_post_init (ATCAIface iface)
- ATCA_STATUS hal_i2c_send (ATCAIface iface, uint8_t ∗txdata, int txlength)
- ATCA_STATUS hal_i2c_receive (ATCAIface iface, uint8_t ∗rxdata, uint16_t ∗rxlength)
- ATCA_STATUS hal_i2c_release (void ∗hal_data)
- ATCA_STATUS hal_i2c_wake (ATCAIface iface)
- ATCA_STATUS hal_i2c_idle (ATCAIface iface)
- ATCA_STATUS hal_i2c_sleep (ATCAIface iface)
- ATCA_STATUS hal_i2c_discover_buses (int i2c_buses[ ], int max_buses)
- ATCA_STATUS hal_i2c_discover_devices (int bus_num, ATCAIfaceCfg ∗cfg, int ∗found)

Variables

- ATCAI2CMaster_t ∗ i2c_hal_data [MAX_I2C_BUSES]
- int i2c_bus_ref_ct = 0
- i2c_config_t conf
- const char ∗ TAG = "HAL_I2C"

15.89.1 Macro Definition Documentation

15.89.1.1 ACK_CHECK_DIS

```
#define ACK_CHECK_DIS 0x0
```

I2C master will not check ack from slave

15.89.1.2 ACK_CHECK_EN

```
#define ACK_CHECK_EN 0x1
```

I2C master will check ack from slave

15.89.1.3 ACK_VAL

```
#define ACK_VAL 0x0
```

I2C ack value

15.89.1.4 LOG_LOCAL_LEVEL

```
#define LOG_LOCAL_LEVEL ESP_LOG_INFO
```

15.89.1.5 MAX_I2C_BUSES

```
#define MAX_I2C_BUSES 2
```

15.89.1.6 NACK_VAL

```
#define NACK_VAL 0x1
```

I2C nack value

15.89.1.7 SCL_PIN

```
#define SCL_PIN 17
```

15.89.1.8 SDA_PIN

```
#define SDA_PIN 16
```

15.89.2 Typedef Documentation

15.89.2.1 ATCAI2CMaster_t

```
typedef struct atcaI2Cmaster ATCAI2CMaster_t
```

15.89.3 Function Documentation

15.89.3.1 hal_i2c_change_baud()

```
void hal_i2c_change_baud (
            ATCAIface iface,
            uint32_t speed )
```

15.89.3.2 hal_i2c_discover_buses()

```
ATCA_STATUS hal_i2c_discover_buses (
            int i2c_buses[ ],
            int max_buses )
```

15.89.3.3 hal_i2c_discover_devices()

```
ATCA_STATUS hal_i2c_discover_devices (
            int bus_num,
            ATCAIfaceCfg * cfg,
            int * found )
```

15.89.3.4 hal_i2c_idle()

```
ATCA_STATUS hal_i2c_idle (
            ATCAIface iface )
```

15.89.3.5 hal_i2c_init()

```
ATCA_STATUS hal_i2c_init (
            void * hal,
            ATCAIfaceCfg * cfg )
```

15.89.3.6 hal_i2c_post_init()

```
ATCA_STATUS hal_i2c_post_init (
            ATCAIface iface )
```

15.89.3.7 hal_i2c_receive()

```
ATCA_STATUS hal_i2c_receive (
            ATCAIface iface,
            uint8_t * rxdata,
            uint16_t * rxlength )
```

15.89.3.8 hal_i2c_release()

```
ATCA_STATUS hal_i2c_release (
            void * hal_data )
```

15.89.3.9 hal_i2c_send()

```
ATCA_STATUS hal_i2c_send (
            ATCAIface iface,
            uint8_t * txdata,
            int txlength )
```

15.89.3.10 hal_i2c_sleep()

```
ATCA_STATUS hal_i2c_sleep (
            ATCAIface iface )
```

15.89.3.11 hal_i2c_wake()

```
ATCA_STATUS hal_i2c_wake (
            ATCAIface iface )
```

15.89.4 Variable Documentation

15.89.4.1 conf

```
i2c_config_t conf
```

15.89.4.2 i2c_bus_ref_ct

```
int i2c_bus_ref_ct = 0
```

15.89.4.3 i2c_hal_data

```
ATCAI2CMaster_t* i2c_hal_data[MAX_I2C_BUSES]
```

15.89.4.4 TAG

```
const char* TAG = "HAL_I2C"
```

15.90 hal_esp32_timer.c File Reference

```
#include "atca_hal.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
```

Functions

- void ets_delay_us (uint32_t)
- void atca_delay_ms (uint32_t msec)

15.90.1 Function Documentation

15.90.1.1 atca_delay_ms()

```
void atca_delay_ms (
            uint32_t msec )
```

15.90.1.2 ets_delay_us()

```
void ets_delay_us (
            uint32_t  )
```

15.91 hal_freertos.c File Reference

FreeRTOS Hardware/OS Abstration Layer.

```
#include "atca_hal.h"
#include "FreeRTOS.h"
#include "semphr.h"
#include "task.h"
```

Macros

- #define ATCA_MUTEX_TIMEOUT portMAX_DELAY

Functions

- ATCA_STATUS hal_create_mutex (void **ppMutex, char *pName)

  Optional hal interfaces.
- ATCA_STATUS hal_destroy_mutex (void *pMutex)
- ATCA_STATUS hal_lock_mutex (void *pMutex)
- ATCA_STATUS hal_unlock_mutex (void *pMutex)

15.91.1 Detailed Description

FreeRTOS Hardware/OS Abstration Layer.

Copyright

15.91.2 Macro Definition Documentation

15.91.2.1 ATCA_MUTEX_TIMEOUT

```
#define ATCA_MUTEX_TIMEOUT portMAX_DELAY
```

15.92 hal_i2c_bitbang.c File Reference

ATCA Hardware abstraction layer for I2C bit banging.

```
#include <asf.h>
#include <string.h>
#include <stdio.h>
#include "atca_hal.h"
#include "atca_device.h"
#include "hal_i2c_bitbang.h"
#include "atca_execution.h"
```

Functions

- ATCA_STATUS hal_i2c_discover_buses (int i2c_buses[ ], int max_buses)

  discover i2c buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the prior knowledge
- ATCA_STATUS hal_i2c_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

  discover any CryptoAuth devices on a given logical bus number
- ATCA_STATUS hal_i2c_init (void ∗hal, ATCAIfaceCfg ∗cfg)

  initialize an I2C interface using given config
- ATCA_STATUS hal_i2c_post_init (ATCAIface iface)

  HAL implementation of I2C post init.
- ATCA_STATUS hal_i2c_send (ATCAIface iface, uint8_t ∗txdata, int txlength)

  HAL implementation of I2C send over ASF.
- ATCA_STATUS hal_i2c_receive (ATCAIface iface, uint8_t ∗rxdata, uint16_t ∗rxlength)

  HAL implementation of I2C receive function for ASF I2C.
- ATCA_STATUS hal_i2c_wake (ATCAIface iface)

  wake up CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_idle (ATCAIface iface)

  idle CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_sleep (ATCAIface iface)

  sleep CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_release (void ∗hal_data)

  manages reference count on given bus and releases resource if no more refences exist

15.92.1 Detailed Description

ATCA Hardware abstraction layer for I2C bit banging.

Copyright

15.93 hal_i2c_bitbang.h File Reference

ATCA Hardware abstraction layer for I2C bit banging.

```
#include "i2c_bitbang_at88ck9000.h"
```

Data Structures

- struct atcaI2Cmaster

  this is the hal_data for ATCA HAL created using ASF

Typedefs

- typedef struct atcaI2Cmaster ATCAI2CMaster_t

  This is the hal_data for ATCA HAL.

Enumerations

- enum i2c_read_write_flag { I2C_WRITE = (uint8_t)0x00, I2C_READ = (uint8_t)0x01 }

  This enumeration lists flags for I2C read or write addressing.

15.93.1 Detailed Description

ATCA Hardware abstraction layer for I2C bit banging.

Copyright

15.94 hal_i2c_start.c File Reference

ATCA Hardware abstraction layer for SAMD21 I2C over START drivers.

```
#include <string.h>
#include <stdio.h>
#include <atmel_start.h>
#include <hal_gpio.h>
#include <hal_delay.h>
#include "atca_hal.h"
#include "atca_device.h"
#include "hal_i2c_start.h"
#include "peripheral_clk_config.h"
#include "atca_execution.h"
#include "atca_start_config.h"
#include "atca_start_iface.h"
```

Functions

- ATCA_STATUS hal_i2c_discover_buses (int i2c_buses[ ], int max_buses)

  discover i2c buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the prior knowledge
- ATCA_STATUS hal_i2c_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

  discover any CryptoAuth devices on a given logical bus number
- ATCA_STATUS hal_i2c_init (void ∗hal, ATCAIfaceCfg ∗cfg)

  initialize an I2C interface using given config
- ATCA_STATUS hal_i2c_post_init (ATCAIface iface)

  HAL implementation of I2C post init.
- ATCA_STATUS hal_i2c_send (ATCAIface iface, uint8_t ∗txdata, int txlength)

  HAL implementation of I2C send over ASF.
- ATCA_STATUS hal_i2c_receive (ATCAIface iface, uint8_t ∗rxdata, uint16_t ∗rxlength)

  HAL implementation of I2C receive function for ASF I2C.
- void change_i2c_speed (ATCAIface iface, uint32_t speed)

  method to change the bus speed of I2C
- ATCA_STATUS hal_i2c_wake (ATCAIface iface)

  wake up CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_idle (ATCAIface iface)

  idle CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_sleep (ATCAIface iface)

  sleep CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_release (void ∗hal_data)

  manages reference count on given bus and releases resource if no more refences exist

15.94.1 Detailed Description

ATCA Hardware abstraction layer for SAMD21 I2C over START drivers.

This code is structured in two parts. Part 1 is the connection of the ATCA HAL API to the physical I2C implementation. Part 2 is the START I2C primitives to set up the interface.

Prerequisite: add SERCOM I2C Master Polled support to application in Atmel Studio

Copyright

15.95 hal_i2c_start.h File Reference

ATCA Hardware abstraction layer for SAMD21 I2C over START drivers.

```
#include "atmel_start.h"
#include <stdlib.h>
```

Data Structures

- struct atcaI2Cmaster
  this is the hal_data for ATCA HAL created using ASF

Macros

- #define MAX_I2C_BUSES 6

Typedefs

- typedef struct atcaI2Cmaster ATCAI2CMaster_t
  this is the hal_data for ATCA HAL for Atmel START SERCOM

Functions

- void change_i2c_speed (ATCAIface iface, uint32_t speed)
  method to change the bus speed of I2C

15.95.1 Detailed Description

ATCA Hardware abstraction layer for SAMD21 I2C over START drivers.

Prerequisite: add SERCOM I2C Master Polled support to application in Atmel Studio

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.96 hal_linux_i2c_userspace.c File Reference

ATCA Hardware abstraction layer for Linux using I2C.

```
#include <linux/i2c-dev.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include "atca_hal.h"
#include "hal_linux_i2c_userspace.h"
```

Functions

- ATCA_STATUS hal_i2c_discover_buses (int i2c_buses[ ], int max_buses)

    discover i2c buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the prior knowledge
- ATCA_STATUS hal_i2c_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

    discover any CryptoAuth devices on a given logical bus number
- ATCA_STATUS hal_i2c_init (void ∗hal, ATCAIfaceCfg ∗cfg)

    initialize an I2C interface using given config
- ATCA_STATUS hal_i2c_post_init (ATCAIface iface)

    HAL implementation of I2C post init.
- ATCA_STATUS hal_i2c_send (ATCAIface iface, uint8_t ∗txdata, int txlength)

    HAL implementation of I2C send over ASF.
- ATCA_STATUS hal_i2c_receive (ATCAIface iface, uint8_t ∗rxdata, uint16_t ∗rxlength)

    HAL implementation of I2C receive function for ASF I2C.
- void change_i2c_speed (ATCAIface iface, uint32_t speed)

    method to change the bus speed of I2C
- ATCA_STATUS hal_i2c_wake (ATCAIface iface)

    wake up CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_idle (ATCAIface iface)

    idle CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_sleep (ATCAIface iface)

    sleep CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_release (void ∗hal_data)

    manages reference count on given bus and releases resource if no more refences exist

15.96.1 Detailed Description

ATCA Hardware abstraction layer for Linux using I2C.

Copyright

15.97 hal_linux_i2c_userspace.h File Reference

ATCA Hardware abstraction layer for Linux using I2C.

Data Structures

- struct atcaI2Cmaster

    this is the hal_data for ATCA HAL created using ASF

Macros

- #define MAX_I2C_BUSES 2

Typedefs

- typedef struct atcaI2Cmaster ATCAI2CMaster_t

15.97.1 Detailed Description

ATCA Hardware abstraction layer for Linux using I2C.

Copyright

15.98 hal_linux_kit_cdc.c File Reference

ATCA Hardware abstraction layer for Linux using kit protocol over a USB CDC device.

```
#include <stdio.h>
#include <string.h>
#include <termios.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include "atca_hal.h"
#include "kit_phy.h"
#include "hal_linux_kit_cdc.h"
#include "kit_protocol.h"
```

Macros

- #define max(a, b) (((a) > (b)) ? (a) : (b))
- #define min(a, b) (((a) < (b)) ? (a) : (b))

Functions

- ATCA_STATUS hal_cdc_discover_buses (int cdc_buses[ ], int max_buses)

  discover cdc buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge.This function is currently not implemented.
- ATCA_STATUS hal_cdc_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

  discover any CryptoAuth devices on a given logical bus number
- ATCA_STATUS hal_kit_cdc_init (void ∗hal, ATCAIfaceCfg ∗cfg)

  HAL implementation of Kit USB CDC init.
- ATCA_STATUS hal_kit_cdc_post_init (ATCAIface iface)

  HAL implementation of Kit USB CDC post init.
- ATCA_STATUS kit_phy_send (ATCAIface iface, const char ∗txdata, int txlength)

  HAL implementation of kit protocol send .It is called by the top layer.
- ATCA_STATUS kit_phy_receive (ATCAIface iface, char ∗rxdata, int ∗rxsize)

  HAL implementation of kit protocol receive data.It is called by the top layer.

- ATCA_STATUS hal_kit_phy_num_found (int8_t ∗num_found)

    Number of USB CDC devices found.
- ATCA_STATUS hal_kit_cdc_send (ATCAIface iface, uint8_t ∗txdata, int txlength)

    HAL implementation of kit protocol send over USB CDC.
- ATCA_STATUS hal_kit_cdc_receive (ATCAIface iface, uint8_t ∗rxdata, uint16_t ∗rxsize)

    HAL implementation of kit protocol receive over USB CDC.
- ATCA_STATUS hal_kit_cdc_wake (ATCAIface iface)

    Call the wake for kit protocol over USB CDC.
- ATCA_STATUS hal_kit_cdc_idle (ATCAIface iface)

    Call the idle for kit protocol over USB CDC.
- ATCA_STATUS hal_kit_cdc_sleep (ATCAIface iface)

    Call the sleep for kit protocol over USB CDC.
- ATCA_STATUS hal_kit_cdc_release (void ∗hal_data)

    Close the physical port for CDC over USB CDC.
- ATCA_STATUS hal_kit_cdc_discover_buses (int cdc_buses[ ], int max_buses)

    discover cdc buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge.This function is currently not implemented.
- ATCA_STATUS hal_kit_cdc_discover_devices (int bus_num, ATCAIfaceCfg ∗cfg, int ∗found)

    discover any CryptoAuth devices on a given logical bus number

Variables

- atcacdc_t _gCdc
- char ∗ dev = "/dev/ttyACM0"
- int speed = B115200

15.98.1 Detailed Description

ATCA Hardware abstraction layer for Linux using kit protocol over a USB CDC device.

Copyright

15.99 hal_linux_kit_cdc.h File Reference

ATCA Hardware abstraction layer for Linux using kit protocol over a USB CDC device.

Data Structures

- struct cdc_device
- struct atcacdc

Macros

- #define CDC_DEVICES_MAX 10
- #define CDC_BUFFER_MAX 1024
- #define INVALID_HANDLE_VALUE ((int)(-1))

Typedefs

- typedef int HANDLE
- typedef struct cdc_device cdc_device_t
- typedef struct atcacdc atcacdc_t

15.99.1 Detailed Description

ATCA Hardware abstraction layer for Linux using kit protocol over a USB CDC device.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.100 hal_linux_kit_hid.c File Reference

ATCA Hardware abstraction layer for Linux using kit protocol over a USB HID device.

```
#include <libudev.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include "atca_hal.h"
#include "hal_linux_kit_hid.h"
#include "hal/kit_protocol.h"
```

Functions

- ATCA_STATUS hal_kit_hid_discover_buses (int i2c_buses[ ], int max_buses)

  discover cdc buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge
- ATCA_STATUS hal_kit_hid_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

  discover any CryptoAuth devices on a given logical bus number
- ATCA_STATUS hal_kit_hid_init (void ∗hal, ATCAIfaceCfg ∗cfg)

  HAL implementation of Kit USB HID init.
- ATCA_STATUS hal_kit_hid_post_init (ATCAIface iface)

  HAL implementation of Kit HID post init.
- ATCA_STATUS kit_phy_send (ATCAIface iface, uint8_t ∗txdata, int txlength)

  HAL implementation of send over USB HID.
- ATCA_STATUS kit_phy_receive (ATCAIface iface, uint8_t ∗rxdata, int ∗rxsize)

  HAL implementation of kit protocol send over USB HID.
- ATCA_STATUS kit_phy_num_found (int8_t ∗num_found)

  Number of USB HID devices found.
- ATCA_STATUS hal_kit_hid_send (ATCAIface iface, uint8_t ∗txdata, int txlength)

  HAL implementation of kit protocol send over USB HID.
- ATCA_STATUS hal_kit_hid_receive (ATCAIface iface, uint8_t ∗rxdata, uint16_t ∗rxsize)

  HAL implementation of send over USB HID.
- ATCA_STATUS hal_kit_hid_wake (ATCAIface iface)

  Call the wake for kit protocol.
- ATCA_STATUS hal_kit_hid_idle (ATCAIface iface)

  Call the idle for kit protocol.
- ATCA_STATUS hal_kit_hid_sleep (ATCAIface iface)

  Call the sleep for kit protocol.
- ATCA_STATUS hal_kit_hid_release (void ∗hal_data)

  Close the physical port for HID.

Variables

- atcahid_t _gHid

15.100.1 Detailed Description

ATCA Hardware abstraction layer for Linux using kit protocol over a USB HID device.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.101 hal_linux_kit_hid.h File Reference

ATCA Hardware abstraction layer for Linux using kit protocol over a USB HID device.

Data Structures

- struct hid_device
- struct atcahid

Macros

- #define HID_DEVICES_MAX 10
- #define HID_PACKET_MAX 512

Typedefs

- typedef struct hid_device hid_device_t
- typedef struct atcahid atcahid_t

15.101.1 Detailed Description

ATCA Hardware abstraction layer for Linux using kit protocol over a USB HID device.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.102 hal_linux_timer.c File Reference

Timer Utility Functions for Linux.

```
#include <stdint.h>
#include <unistd.h>
#include "atca_hal.h"
```

Functions

- void atca_delay_us (uint32_t delay)

  Timer API implemented at the HAL level.
- void atca_delay_10us (uint32_t delay)

  This function delays for a number of tens of microseconds.
- void atca_delay_ms (uint32_t delay)

  This function delays for a number of milliseconds.

15.102.1 Detailed Description

Timer Utility Functions for Linux.

Copyright

15.103 hal_pic32mx695f512h_i2c.c File Reference

ATCA Hardware abstraction layer for PIC32MX695F512H I2C over plib drivers.

```
#include <plib.h>
#include <stdio.h>
#include <string.h>
#include "hal/atca_hal.h"
#include "hal/hal_pic32mx695f512h_i2c.h"
```

Functions

- void i2c_write (I2C_MODULE i2c_id, uint8_t address, uint8_t ∗data, int len)
- ATCA_STATUS i2c_read (I2C_MODULE i2c_id, uint8_t address, uint8_t ∗data, uint16_t len)
- ATCA_STATUS hal_i2c_discover_buses (int i2c_buses[ ], int max_buses)

  discover i2c buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the prior knowledge
- ATCA_STATUS hal_i2c_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

  discover any CryptoAuth devices on a given logical bus number
- ATCA_STATUS hal_i2c_init (void ∗hal, ATCAIfaceCfg ∗cfg)

  initialize an I2C interface using given config
- ATCA_STATUS hal_i2c_post_init (ATCAIface iface)

  HAL implementation of I2C post init.
- ATCA_STATUS hal_i2c_send (ATCAIface iface, uint8_t ∗txdata, int txlength)

  HAL implementation of I2C send over ASF.
- ATCA_STATUS hal_i2c_receive (ATCAIface iface, uint8_t ∗rxdata, uint16_t ∗rxlength)

  HAL implementation of I2C receive function for ASF I2C.
- void change_i2c_speed (ATCAIface iface, uint32_t speed)

  method to change the bus speed of I2C
- ATCA_STATUS hal_i2c_wake (ATCAIface iface)

  wake up CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_idle (ATCAIface iface)

  idle CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_sleep (ATCAIface iface)

  sleep CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_release (void ∗hal_data)

  manages reference count on given bus and releases resource if no more refences exist

15.103.1 Detailed Description

ATCA Hardware abstraction layer for PIC32MX695F512H I2C over plib drivers.

This code is structured in two parts. Part 1 is the connection of the ATCA HAL API to the physical I2C implementation. Part 2 is the xxx I2C primitives to set up the interface.

Prerequisite:

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.104 hal_pic32mx695f512h_i2c.h File Reference

ATCA Hardware abstraction layer for PIC32MX695F512H I2C over xxx drivers.

Data Structures

- struct atcaI2Cmaster

    this is the hal_data for ATCA HAL created using ASF

Macros

- #define GetSystemClock() (80000000ul)
- #define GetPeripheralClock() (GetSystemClock() / (1 << OSCCONbits.PBDIV))
- #define GetInstructionClock() (GetSystemClock())
- #define MAX_I2C_BUSES 4

Typedefs

- typedef struct atcaI2Cmaster ATCAI2CMaster_t

    this is the hal_data for ATCA HAL

Functions

- void i2c_write (I2C_MODULE i2c_id, uint8_t address, uint8_t *data, int len)
- ATCA_STATUS i2c_read (I2C_MODULE i2c_id, uint8_t address, uint8_t *data, uint16_t len)
- void change_i2c_speed (ATCAIface iface, uint32_t speed)

    method to change the bus speed of I2C

15.104.1 Detailed Description

ATCA Hardware abstraction layer for PIC32MX695F512H I2C over xxx drivers.

This code is structured in two parts. Part 1 is the connection of the ATCA HAL API to the physical I2C implementation. Part 2 is the xxx I2C primitives to set up the interface.

Prerequisite:

Copyright

15.105 hal_pic32mx695f512h_timer.c File Reference

ATCA Hardware abstraction layer for PIC32MX695F512H timer/delay routine.

```
#include <plib.h>
#include "hal/atca_hal.h"
```

Macros

- #define CPU_CLOCK (80000000UL)
- #define us_SCALE ((CPU_CLOCK / 2) / 1000000)

Functions

- void delay_us (UINT32 delay)
- void atca_delay_us (uint32_t delay)

  Timer API implemented at the HAL level.
- void atca_delay_10us (uint32_t delay)

  This function delays for a number of tens of microseconds.
- void atca_delay_ms (uint32_t delay)

  This function delays for a number of milliseconds.

15.105.1 Detailed Description

ATCA Hardware abstraction layer for PIC32MX695F512H timer/delay routine.

Copyright

15.106 hal_pic32mz2048efm_i2c.c File Reference

ATCA Hardware abstraction layer for PIC32MZ2048.

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "atca_hal.h"
#include "atca_device.h"
#include "hal/hal_pic32mz2048efm_i2c.h"
#include "driver/i2c/drv_i2c.h"
#include "system_definitions.h"
#include "driver/i2c/src/drv_i2c_local.h"
```

Functions

- ATCA_STATUS hal_i2c_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int *found)

  discover any CryptoAuth devices on a given logical bus number
- ATCA_STATUS hal_i2c_discover_buses (int i2c_buses[ ], int max_buses)

  discover i2c buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge
- ATCA_STATUS hal_i2c_init (void *hal, ATCAIfaceCfg *cfg)

  initialize an I2C interface using given config
- ATCA_STATUS hal_i2c_post_init (ATCAIface iface)

  HAL implementation of I2C post init.
- ATCA_STATUS hal_i2c_send (ATCAIface iface, uint8_t *txdata, int txlength)
- ATCA_STATUS hal_i2c_receive (ATCAIface iface, uint8_t *rxdata, uint16_t *rxlength)

  HAL implementation of I2C receive function.
- ATCA_STATUS hal_i2c_wake (ATCAIface iface)

  wake up CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_idle (ATCAIface iface)

  idle CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_sleep (ATCAIface iface)

  sleep CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_release (void *hal_data)

  manages reference count on given bus and releases resource if no more refences exist

Variables

- DRV_HANDLE drvI2CMasterHandle
- DRV_HANDLE drvI2CMasterHandle1
- DRV_I2C_BUFFER_HANDLE write_bufHandle
- DRV_I2C_BUFFER_HANDLE read_bufHandle
- uint32_t Debug_count = 0

  HAL implementation of I2C send over ASF.
- uint32_t bytes_transferred = 0

15.106.1 Detailed Description

ATCA Hardware abstraction layer for PIC32MZ2048.

Copyright

15.106.2 Function Documentation

15.106.2.1 hal_i2c_discover_buses()

```
ATCA_STATUS hal_i2c_discover_buses (
             int i2c_buses[],
             int max_buses )
```

discover i2c buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge

Parameters

in	*i2c_buses*	an array of logical bus numbers
in	*max_buses*	maximum number of buses the app wants to attempt to discover

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.106.2.2 hal_i2c_discover_devices()

```
ATCA_STATUS hal_i2c_discover_devices (
             int bus_num,
             ATCAIfaceCfg cfg[],
             int * found )
```

discover any CryptoAuth devices on a given logical bus number

Parameters

in	*bus_num*	logical bus number on which to look for CryptoAuth devices
out	*cfg*	pointer to head of an array of interface config structures which get filled in by this method
out	*found*	number of devices found on this bus

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.106.2.3 hal_i2c_idle()

```
ATCA_STATUS hal_i2c_idle (
            ATCAIface iface )
```

idle CryptoAuth device using I2C bus

Parameters

in	*iface*	interface to logical device to idle

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.106.2.4 hal_i2c_init()

```
ATCA_STATUS hal_i2c_init (
            void * hal,
            ATCAIfaceCfg * cfg )
```

initialize an I2C interface using given config

Parameters

in	*hal*	opaque ptr to HAL data
in	*cfg*	interface configuration

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.106.2.5 hal_i2c_post_init()

```
ATCA_STATUS hal_i2c_post_init (
            ATCAIface iface )
```

HAL implementation of I2C post init.

Parameters

in	*iface*	instance

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.106.2.6 hal_i2c_receive()

```
ATCA_STATUS hal_i2c_receive (
            ATCAIface iface,
            uint8_t * rxdata,
            uint16_t * rxlength )
```

HAL implementation of I2C receive function.

Parameters

in	*iface*	Device to interact with.
out	*rxdata*	Data received will be returned here.
in,out	*rxlength*	As input, the size of the rxdata buffer. As output, the number of bytes received.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.106.2.7 hal_i2c_release()

```
ATCA_STATUS hal_i2c_release (
            void * hal_data )
```

manages reference count on given bus and releases resource if no more refences exist

Parameters

in	*hal_data*	opaque pointer to hal data structure - known only to the HAL implementation

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.106.2.8 hal_i2c_send()

```
ATCA_STATUS hal_i2c_send (
            ATCAIface iface,
            uint8_t * txdata,
            int txlength )
```

15.106.2.9 hal_i2c_sleep()

```
ATCA_STATUS hal_i2c_sleep (
            ATCAIface iface )
```

sleep CryptoAuth device using I2C bus

Parameters

in	*iface*	interface to logical device to sleep

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.106.2.10 hal_i2c_wake()

```
ATCA_STATUS hal_i2c_wake (
            ATCAIface iface )
```

wake up CryptoAuth device using I2C bus

Parameters

in	*iface*	interface to logical device to wakeup

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.106.3 Variable Documentation

15.106.3.1 bytes_transferred

```
uint32_t bytes_transferred = 0
```

15.106.3.2 Debug_count

```
uint32_t Debug_count = 0
```

HAL implementation of I2C send over ASF.

Parameters

in	*iface*	instance
in	*txdata*	pointer to space to bytes to send
in	*txlength*	number of bytes to send

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.106.3.3 drvI2CMasterHandle

```
DRV_HANDLE drvI2CMasterHandle
```

15.106.3.4 drvI2CMasterHandle1

```
DRV_HANDLE drvI2CMasterHandle1
```

15.106.3.5 read_bufHandle

```
DRV_I2C_BUFFER_HANDLE read_bufHandle
```

15.106.3.6 write_bufHandle

```
DRV_I2C_BUFFER_HANDLE write_bufHandle
```

15.107 hal_pic32mz2048efm_i2c.h File Reference

ATCA Hardware abstraction layer for PIC32MZ2048.

Data Structures

- struct atcaI2Cmaster

  this is the hal_data for ATCA HAL created using ASF
- struct DRV_I2C_Object

Macros

- #define HARMONY_I2C_DRIVER 1
- #define MAX_I2C_BUSES 3

Typedefs

- typedef struct atcaI2Cmaster ATCAI2CMaster_t

  this is the hal_data for ATCA HAL

15.107.1 Detailed Description

ATCA Hardware abstraction layer for PIC32MZ2048.

Copyright

15.108 hal_pic32mz2048efm_timer.c File Reference

ATCA Hardware abstraction layer for PIC32MZ2048.

```
#include <stdint.h>
```

Macros

- #define GetSystemClock() (200000000UL)/∗ Fcy = 200MHz ∗/
- #define us_SCALE (GetSystemClock() / 2000000)

Functions

- void delay_us (uint32_t delay)
- void atca_delay_us (uint32_t delay)

  Timer API implemented at the HAL level.
- void atca_delay_10us (uint32_t delay)

  This function delays for a number of tens of microseconds.
- void atca_delay_ms (uint32_t delay)

  This function delays for a number of milliseconds.

15.108.1 Detailed Description

ATCA Hardware abstraction layer for PIC32MZ2048.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.109 hal_sam4s_i2c_asf.c File Reference

ATCA Hardware abstraction layer for SAM4S I2C over ASF drivers.

```
#include <asf.h>
#include <string.h>
#include <stdio.h>
#include "atca_hal.h"
#include "hal_sam4s_i2c_asf.h"
#include "atca_device.h"
#include "atca_execution.h"
```

Functions

- ATCA_STATUS hal_i2c_discover_buses (int i2c_buses[ ], int max_buses)
  discover i2c buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the prior knowledge
- ATCA_STATUS hal_i2c_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)
  discover any CryptoAuth devices on a given logical bus number
- ATCA_STATUS hal_i2c_init (void ∗hal, ATCAIfaceCfg ∗cfg)
  initialize an I2C interface using given config
- ATCA_STATUS hal_i2c_post_init (ATCAIface iface)
  HAL implementation of I2C post init.
- ATCA_STATUS hal_i2c_send (ATCAIface iface, uint8_t ∗txdata, int txlength)
  HAL implementation of I2C send over ASF.
- ATCA_STATUS hal_i2c_receive (ATCAIface iface, uint8_t ∗rxdata, uint16_t ∗rxlength)
  HAL implementation of I2C receive function for ASF I2C.
- void change_i2c_speed (ATCAIface iface, uint32_t speed)
  method to change the bus speed of I2C
- ATCA_STATUS hal_i2c_wake (ATCAIface iface)
  wake up CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_idle (ATCAIface iface)
  idle CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_sleep (ATCAIface iface)
  sleep CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_release (void ∗hal_data)
  manages reference count on given bus and releases resource if no more refences exist

15.109.1 Detailed Description

ATCA Hardware abstraction layer for SAM4S I2C over ASF drivers.

This code is structured in two parts. Part 1 is the connection of the ATCA HAL API to the physical I2C implementation. Part 2 is the ASF I2C primitives to set up the interface.

Prerequisite: add "TWI - Two-Wire Interface (Common API) (service)" module to application in Atmel Studio

Copyright

15.110 hal_sam4s_i2c_asf.h File Reference

ATCA Hardware abstraction layer for SAM4S I2C over ASF drivers.

```
#include <asf.h>
```

Data Structures

- struct atcaI2Cmaster

  this is the hal_data for ATCA HAL created using ASF

Macros

- #define MAX_I2C_BUSES 2

Typedefs

- typedef struct atcaI2Cmaster ATCAI2CMaster_t

  this is the hal_data for ATCA HAL

Functions

- void change_i2c_speed (ATCAIface iface, uint32_t speed)

  method to change the bus speed of I2C

15.110.1 Detailed Description

ATCA Hardware abstraction layer for SAM4S I2C over ASF drivers.

Prerequisite: add "TWI - Two-Wire Interface (Common API) (service)" module to application in Atmel Studio

Copyright

15.111 hal_sam4s_timer_asf.c File Reference

ATCA Hardware abstraction layer for SAM4S I2C over ASF drivers.

```
#include <asf.h>
#include <delay.h>
#include "atca_hal.h"
```

Functions

- void atca_delay_us (uint32_t delay)

  Timer API implemented at the HAL level.
- void atca_delay_10us (uint32_t delay)

  This function delays for a number of tens of microseconds.
- void atca_delay_ms (uint32_t delay)

  This function delays for a number of milliseconds.

15.111.1 Detailed Description

ATCA Hardware abstraction layer for SAM4S I2C over ASF drivers.

Prerequisite: add "Delay routines (service)" module to application in Atmel Studio

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.112 hal_samb11_i2c_asf.c File Reference

ATCA Hardware abstraction layer for SAMB11 I2C over ASF drivers.

```
#include <asf.h>
#include <string.h>
#include <stdio.h>
#include "atca_hal.h"
#include "hal_samb11_i2c_asf.h"
#include "atca_device.h"
#include "atca_execution.h"
```

Functions

- ATCA_STATUS hal_i2c_discover_buses (int i2c_buses[ ], int max_buses)

  discover i2c buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the prior knowledge
- ATCA_STATUS hal_i2c_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

  discover any CryptoAuth devices on a given logical bus number
- ATCA_STATUS hal_i2c_init (void ∗hal, ATCAIfaceCfg ∗cfg)

  initialize an I2C interface using given config
- ATCA_STATUS hal_i2c_post_init (ATCAIface iface)

  HAL implementation of I2C post init.
- ATCA_STATUS hal_i2c_send (ATCAIface iface, uint8_t ∗txdata, int txlength)

  HAL implementation of I2C send over ASF.
- ATCA_STATUS hal_i2c_receive (ATCAIface iface, uint8_t ∗rxdata, uint16_t ∗rxlength)

  HAL implementation of I2C receive function for ASF I2C.
- ATCA_STATUS hal_i2c_wake (ATCAIface iface)

  wake up CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_idle (ATCAIface iface)

  idle CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_sleep (ATCAIface iface)

  sleep CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_release (void ∗hal_data)

  manages reference count on given bus and releases resource if no more refences exist

15.112.1 Detailed Description

ATCA Hardware abstraction layer for SAMB11 I2C over ASF drivers.

This code is structured in two parts. Part 1 is the connection of the ATCA HAL API to the physical I2C implementation. Part 2 is the ASF I2C primitives to set up the interface.

Prerequisite: add I2C Master Polled support to application in Atmel Studio

Copyright

15.113 hal_samb11_i2c_asf.h File Reference

ATCA Hardware abstraction layer for SAMB11 I2C over ASF drivers.

```
#include <asf.h>
```

Data Structures

- struct atcaI2Cmaster

  this is the hal_data for ATCA HAL created using ASF

Macros

- #define MAX_I2C_BUSES 2

Typedefs

- typedef struct atcaI2Cmaster ATCAI2CMaster_t

  this is the hal_data for ATCA HAL for ASF

15.113.1 Detailed Description

ATCA Hardware abstraction layer for SAMB11 I2C over ASF drivers.

Prerequisite: add SERCOM I2C Master Polled support to application in Atmel Studio

Copyright

15.114 hal_samb11_timer_asf.c File Reference

ATCA Hardware abstraction layer for SAMB11 timer/delay over ASF drivers.

```
#include <asf.h>
#include <delay.h>
#include "atca_hal.h"
```

Functions

- void atca_delay_us (uint32_t delay)

  Timer API implemented at the HAL level.
- void atca_delay_10us (uint32_t delay)

  This function delays for a number of tens of microseconds.
- void atca_delay_ms (uint32_t delay)

  This function delays for a number of milliseconds.

15.114.1 Detailed Description

ATCA Hardware abstraction layer for SAMB11 timer/delay over ASF drivers.

Copyright

15.115 hal_samd21_i2c_asf.c File Reference

ATCA Hardware abstraction layer for SAMD21 I2C over ASF drivers.

```
#include <asf.h>
#include <string.h>
#include <stdio.h>
#include "atca_hal.h"
#include "hal_samd21_i2c_asf.h"
#include "atca_device.h"
#include "atca_execution.h"
#include "atca_status.h"
```

Functions

- ATCA_STATUS hal_i2c_discover_buses (int i2c_buses[ ], int max_buses)

    discover i2c buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the prior knowledge
- ATCA_STATUS hal_i2c_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

    discover any CryptoAuth devices on a given logical bus number
- ATCA_STATUS hal_i2c_init (void ∗hal, ATCAIfaceCfg ∗cfg)

    initialize an I2C interface using given config
- ATCA_STATUS hal_i2c_post_init (ATCAIface iface)

    HAL implementation of I2C post init.
- ATCA_STATUS hal_i2c_send (ATCAIface iface, uint8_t ∗txdata, int txlength)

    HAL implementation of I2C send over ASF.
- ATCA_STATUS hal_i2c_receive (ATCAIface iface, uint8_t ∗rxdata, uint16_t ∗rxlength)

    HAL implementation of I2C receive function for ASF I2C.
- void change_i2c_speed (ATCAIface iface, uint32_t speed)

    method to change the bus speed of I2C
- ATCA_STATUS hal_i2c_wake (ATCAIface iface)

    wake up CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_idle (ATCAIface iface)

    idle CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_sleep (ATCAIface iface)

    sleep CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_release (void ∗hal_data)

    manages reference count on given bus and releases resource if no more refences exist

15.115.1 Detailed Description

ATCA Hardware abstraction layer for SAMD21 I2C over ASF drivers.

This code is structured in two parts. Part 1 is the connection of the ATCA HAL API to the physical I2C implementation. Part 2 is the ASF I2C primitives to set up the interface.

Prerequisite: add SERCOM I2C Master Polled support to application in Atmel Studio

Copyright

15.116 hal_samd21_i2c_asf.h File Reference

ATCA Hardware abstraction layer for SAMD21 I2C over ASF drivers.

```
#include <asf.h>
```

Data Structures

- struct atcaI2Cmaster

  this is the hal_data for ATCA HAL created using ASF

Macros

- #define MAX_I2C_BUSES 6

Typedefs

- typedef struct atcaI2Cmaster ATCAI2CMaster_t

  this is the hal_data for ATCA HAL for ASF SERCOM

Functions

- void change_i2c_speed (ATCAIface iface, uint32_t speed)

  method to change the bus speed of I2C

15.116.1 Detailed Description

ATCA Hardware abstraction layer for SAMD21 I2C over ASF drivers.

Prerequisite: add SERCOM I2C Master Polled support to application in Atmel Studio

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.117 hal_samd21_timer_asf.c File Reference

ATCA Hardware abstraction layer for SAMD21 timer/delay over ASF drivers.

```
#include <asf.h>
#include <delay.h>
#include "atca_hal.h"
```

Functions

- void atca_delay_us (uint32_t delay)

  Timer API implemented at the HAL level.
- void atca_delay_10us (uint32_t delay)

  This function delays for a number of tens of microseconds.
- void atca_delay_ms (uint32_t delay)

  This function delays for a number of milliseconds.

15.117.1 Detailed Description

ATCA Hardware abstraction layer for SAMD21 timer/delay over ASF drivers.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.118 hal_samg55_i2c_asf.c File Reference

ATCA Hardware abstraction layer for SAMG55 I2C over ASF drivers.

```
#include <asf.h>
#include <string.h>
#include <stdio.h>
#include "atca_hal.h"
#include "atca_device.h"
#include "hal_samg55_i2c_asf.h"
#include "atca_execution.h"
```

Functions

- ATCA_STATUS hal_i2c_discover_buses (int i2c_buses[ ], int max_buses)

  discover i2c buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the prior knowledge
- ATCA_STATUS hal_i2c_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

  discover any CryptoAuth devices on a given logical bus number
- ATCA_STATUS hal_i2c_init (void ∗hal, ATCAIfaceCfg ∗cfg)

  initialize an I2C interface using given config
- ATCA_STATUS hal_i2c_post_init (ATCAIface iface)

  HAL implementation of I2C post init.
- ATCA_STATUS hal_i2c_send (ATCAIface iface, uint8_t ∗txdata, int txlength)

  HAL implementation of I2C send over ASF.
- ATCA_STATUS hal_i2c_receive (ATCAIface iface, uint8_t ∗rxdata, uint16_t ∗rxlength)

  HAL implementation of I2C receive function for ASF I2C.
- void change_i2c_speed (ATCAIface iface, uint32_t speed)

  method to change the bus speed of I2C
- ATCA_STATUS hal_i2c_wake (ATCAIface iface)

  wake up CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_idle (ATCAIface iface)

  idle CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_sleep (ATCAIface iface)

  sleep CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_release (void ∗hal_data)

  manages reference count on given bus and releases resource if no more refences exist

15.118.1 Detailed Description

ATCA Hardware abstraction layer for SAMG55 I2C over ASF drivers.

This code is structured in two parts. Part 1 is the connection of the ATCA HAL API to the physical I2C implementation. Part 2 is the ASF I2C primitives to set up the interface.

Prerequisite: add "TWI - Two-Wire Interface (Common API) (service)" module to application in Atmel Studio

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.119 hal_samg55_i2c_asf.h File Reference

ATCA Hardware abstraction layer for SAMG55 I2C over ASF drivers.

```
#include <asf.h>
```

Data Structures

- struct atcaI2Cmaster
  this is the hal_data for ATCA HAL created using ASF

Macros

- #define MAX_I2C_BUSES 2

Typedefs

- typedef struct atcaI2Cmaster ATCAI2CMaster_t
  this is the hal_data for ATCA HAL

Functions

- void change_i2c_speed (ATCAIface iface, uint32_t speed)
  method to change the bus speed of I2C

15.119.1 Detailed Description

ATCA Hardware abstraction layer for SAMG55 I2C over ASF drivers.

Prerequisite: add "TWI - Two-Wire Interface (Common API) (service)" module to application in Atmel Studio

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.120 hal_samg55_timer_asf.c File Reference

Prerequisite: add "Delay routines (service)" module to application in Atmel Studio.

```
#include <asf.h>
#include <delay.h>
#include "atca_hal.h"
```

Functions

- void atca_delay_us (uint32_t delay)

    Timer API implemented at the HAL level.
- void atca_delay_10us (uint32_t delay)

    This function delays for a number of tens of microseconds.
- void atca_delay_ms (uint32_t delay)

    This function delays for a number of milliseconds.

15.120.1 Detailed Description

Prerequisite: add "Delay routines (service)" module to application in Atmel Studio.

Copyright

15.121 hal_samv71_i2c_asf.c File Reference

ATCA Hardware abstraction layer for SAMV71 I2C over ASF drivers.

```
#include <asf.h>
#include <string.h>
#include <stdio.h>
#include "atca_hal.h"
#include "hal_samv71_i2c_asf.h"
#include "atca_device.h"
#include "atca_execution.h"
```

Functions

- ATCA_STATUS hal_i2c_discover_buses (int i2c_buses[ ], int max_buses)

  discover i2c buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the prior knowledge
- ATCA_STATUS hal_i2c_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

  discover any CryptoAuth devices on a given logical bus number
- ATCA_STATUS hal_i2c_init (void ∗hal, ATCAIfaceCfg ∗cfg)

  initialize an I2C interface using given config
- ATCA_STATUS hal_i2c_post_init (ATCAIface iface)

  HAL implementation of I2C post init.
- ATCA_STATUS hal_i2c_send (ATCAIface iface, uint8_t ∗txdata, int txlength)

  HAL implementation of I2C send over ASF.
- ATCA_STATUS hal_i2c_receive (ATCAIface iface, uint8_t ∗rxdata, uint16_t ∗rxlength)

  HAL implementation of I2C receive function for ASF I2C.
- void change_i2c_speed (ATCAIface iface, uint32_t speed)

  method to change the bus speed of I2C
- ATCA_STATUS hal_i2c_wake (ATCAIface iface)

  wake up CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_idle (ATCAIface iface)

  idle CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_sleep (ATCAIface iface)

  sleep CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_release (void ∗hal_data)

  manages reference count on given bus and releases resource if no more refences exist

15.121.1 Detailed Description

ATCA Hardware abstraction layer for SAMV71 I2C over ASF drivers.

This code is structured in two parts. Part 1 is the connection of the ATCA HAL API to the physical I2C implementation. Part 2 is the ASF I2C primitives to set up the interface.

Prerequisite: add SERCOM I2C Master Polled support to application in Atmel Studio

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.122 hal_samv71_i2c_asf.h File Reference

ATCA Hardware abstraction layer for SAMV71 I2C over ASF drivers.

```
#include <asf.h>
```

Data Structures

- struct atcaI2Cmaster

  this is the hal_data for ATCA HAL created using ASF

Macros

- #define MAX_I2C_BUSES 3

Typedefs

- typedef struct atcaI2Cmaster ATCAI2CMaster_t

  this is the hal_data for ATCA HAL for ASF SERCOM

Functions

- void change_i2c_speed (ATCAIface iface, uint32_t speed)

  method to change the bus speed of I2C

15.122.1 Detailed Description

ATCA Hardware abstraction layer for SAMV71 I2C over ASF drivers.

Prerequisite: add SERCOM I2C Master Polled support to application in Atmel Studio

Copyright

15.123 hal_samv71_timer_asf.c File Reference

ATCA Hardware abstraction layer for SAMD21 timer/delay over ASF drivers.

```
#include <asf.h>
#include <delay.h>
#include "atca_hal.h"
```

Functions

- void atca_delay_us (uint32_t delay)

  Timer API implemented at the HAL level.
- void atca_delay_10us (uint32_t delay)

  This function delays for a number of tens of microseconds.
- void atca_delay_ms (uint32_t delay)

  This function delays for a number of milliseconds.

15.123.1 Detailed Description

ATCA Hardware abstraction layer for SAMD21 timer/delay over ASF drivers.

Copyright

15.124 hal_swi_bitbang.c File Reference

ATCA Hardware abstraction layer for SWI bit banging.

```
#include <asf.h>
#include <string.h>
#include <stdio.h>
#include "atca_hal.h"
#include "hal_swi_bitbang.h"
#include "atca_device.h"
```

Functions

- ATCA_STATUS hal_swi_discover_buses (int swi_buses[ ], int max_buses)

  discover swi buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application.This function is currently not supported. of the a-priori knowledge
- ATCA_STATUS hal_swi_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

  discover any CryptoAuth devices on a given logical bus number.This function is curently not supported.
- ATCA_STATUS hal_swi_init (void ∗hal, ATCAIfaceCfg ∗cfg)

  hal_swi_init manages requests to initialize a physical interface. It manages use counts so when an interface has released the physical layer, it will disable the interface for some other use. You can have multiple ATCAIFace instances using the same bus, and you can have multiple ATCAIFace instances on multiple swi buses, so hal_swi_init manages these things and ATCAIFace is abstracted from the physical details.
- ATCA_STATUS hal_swi_post_init (ATCAIface iface)

  HAL implementation of SWI post init.
- ATCA_STATUS hal_swi_send (ATCAIface iface, uint8_t ∗txdata, int txlength)

  Send byte(s) via SWI.
- ATCA_STATUS hal_swi_receive (ATCAIface iface, uint8_t ∗rxdata, uint16_t ∗rxlength)

  Receive byte(s) via SWI.
- ATCA_STATUS hal_swi_wake (ATCAIface iface)

  Send Wake flag via SWI.
- ATCA_STATUS hal_swi_idle (ATCAIface iface)

  Send Idle flag via SWI.
- ATCA_STATUS hal_swi_sleep (ATCAIface iface)

  Send Sleep flag via SWI.
- ATCA_STATUS hal_swi_release (void ∗hal_data)

  Manages reference count on given bus and releases resource if no more reference(s) exist.

15.124.1 Detailed Description

ATCA Hardware abstraction layer for SWI bit banging.

Copyright

15.125 hal_swi_bitbang.h File Reference

ATCA Hardware abstraction layer for SWI bit banging.

```
#include "swi_bitbang_at88ck9000.h"
```

Data Structures

- struct atcaSWImaster

  This is the hal_data for ATCA HAL.

Typedefs

- typedef struct atcaSWImaster ATCASWIMaster_t

  This is the hal_data for ATCA HAL.

Enumerations

- enum swi_flag { SWI_FLAG_CMD = (uint8_t)0x77, SWI_FLAG_TX = (uint8_t)0x88, SWI_FLAG_IDLE = (uint8_t)0xBB, SWI_FLAG_SLEEP = (uint8_t)0xCC }

  This enumeration lists flags for SWI.

15.125.1 Detailed Description

ATCA Hardware abstraction layer for SWI bit banging.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.126 hal_swi_uart.c File Reference

ATCA Hardware abstraction layer for SWI over UART drivers.

```
#include <string.h>
#include <stdio.h>
#include "atca_hal.h"
#include "hal_swi_uart.h"
#include "atca_device.h"
#include "atca_execution.h"
```

Functions

- ATCA_STATUS hal_swi_discover_buses (int swi_buses[ ], int max_buses)

  discover swi buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application.This function is currently not supported. of the a-priori knowledge
- ATCA_STATUS hal_swi_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

  discover any CryptoAuth devices on a given logical bus number.This function is curently not supported.
- ATCA_STATUS hal_swi_init (void ∗hal, ATCAIfaceCfg ∗cfg)

  hal_swi_init manages requests to initialize a physical interface. It manages use counts so when an interface has released the physical layer, it will disable the interface for some other use. You can have multiple ATCAIFace instances using the same bus, and you can have multiple ATCAIFace instances on multiple swi buses, so hal_swi_init manages these things and ATCAIFace is abstracted from the physical details.
- ATCA_STATUS hal_swi_post_init (ATCAIface iface)

  HAL implementation of SWI post init.
- ATCA_STATUS hal_swi_send_flag (ATCAIface iface, uint8_t data)

  HAL implementation of SWI send one byte over UART.
- ATCA_STATUS hal_swi_send (ATCAIface iface, uint8_t ∗txdata, int txlength)

  Send byte(s) via SWI.
- ATCA_STATUS hal_swi_receive (ATCAIface iface, uint8_t ∗rxdata, uint16_t ∗rxlength)

  Receive byte(s) via SWI.
- ATCA_STATUS hal_swi_wake (ATCAIface iface)

  Send Wake flag via SWI.
- ATCA_STATUS hal_swi_idle (ATCAIface iface)

  Send Idle flag via SWI.
- ATCA_STATUS hal_swi_sleep (ATCAIface iface)

  Send Sleep flag via SWI.
- ATCA_STATUS hal_swi_release (void ∗hal_data)

  Manages reference count on given bus and releases resource if no more reference(s) exist.

15.126.1 Detailed Description

ATCA Hardware abstraction layer for SWI over UART drivers.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.127 hal_swi_uart.h File Reference

ATCA Hardware abstraction layer for SWI over UART drivers.

Macros

- #define SWI_WAKE_TOKEN ((uint8_t)0x00)

  flag preceding a command
- #define SWI_FLAG_CMD ((uint8_t)0x77)

  flag preceding a command
- #define SWI_FLAG_TX ((uint8_t)0x88)

  flag requesting a response
- #define SWI_FLAG_IDLE ((uint8_t)0xBB)

  flag requesting to go into Idle mode
- #define SWI_FLAG_SLEEP ((uint8_t)0xCC)

  flag requesting to go into Sleep mode

Functions

- ATCA_STATUS hal_swi_send_flag (ATCAIface iface, uint8_t data)
  HAL implementation of SWI send one byte over UART.

15.127.1 Detailed Description

ATCA Hardware abstraction layer for SWI over UART drivers.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.128 hal_timer_start.c File Reference

ATCA Hardware abstraction layer for SAMD21 I2C over START drivers.

```
#include <hal_delay.h>
#include "atca_hal.h"
```

Functions

- void atca_delay_us (uint32_t delay)
  Timer API implemented at the HAL level.
- void atca_delay_10us (uint32_t delay)
  This function delays for a number of tens of microseconds.
- void atca_delay_ms (uint32_t delay)
  This function delays for a number of milliseconds.

15.128.1 Detailed Description

ATCA Hardware abstraction layer for SAMD21 I2C over START drivers.

Prerequisite: add SERCOM I2C Master Polled support to application in Atmel Studio

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.129 hal_uc3_i2c_asf.c File Reference

ATCA Hardware abstraction layer for SAMV71 I2C over ASF drivers.

```
#include <asf.h>
#include <string.h>
#include <stdio.h>
#include "atca_hal.h"
#include "atca_device.h"
#include "atca_execution.h"
#include "hal_uc3_i2c_asf.h"
```

Functions

- ATCA_STATUS hal_i2c_discover_buses (int i2c_buses[ ], int max_buses)

  discover i2c buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the prior knowledge
- ATCA_STATUS hal_i2c_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

  discover any CryptoAuth devices on a given logical bus number
- ATCA_STATUS hal_i2c_init (void ∗hal, ATCAIfaceCfg ∗cfg)

  initialize an I2C interface using given config
- ATCA_STATUS hal_i2c_post_init (ATCAIface iface)

  HAL implementation of I2C post init.
- ATCA_STATUS hal_i2c_send (ATCAIface iface, uint8_t ∗txdata, int txlength)

  HAL implementation of I2C send over ASF.
- ATCA_STATUS hal_i2c_receive (ATCAIface iface, uint8_t ∗rxdata, uint16_t ∗rxlength)

  HAL implementation of I2C receive function for ASF I2C.
- void change_i2c_speed (ATCAIface iface, uint32_t speed)

  method to change the bus speed of I2C
- ATCA_STATUS hal_i2c_wake (ATCAIface iface)

  wake up CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_idle (ATCAIface iface)

  idle CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_sleep (ATCAIface iface)

  sleep CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_release (void ∗hal_data)

  manages reference count on given bus and releases resource if no more refences exist

15.129.1 Detailed Description

ATCA Hardware abstraction layer for SAMV71 I2C over ASF drivers.

This code is structured in two parts. Part 1 is the connection of the ATCA HAL API to the physical I2C implementation. Part 2 is the ASF I2C primitives to set up the interface.

Prerequisite: add SERCOM I2C Master Polled support to application in Atmel Studio

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.130 hal_uc3_i2c_asf.h File Reference

ATCA Hardware abstraction layer for SAMV71 I2C over ASF drivers.

```
#include <asf.h>
#include "twi.h"
```

Data Structures

- struct atcaI2Cmaster

  this is the hal_data for ATCA HAL created using ASF

Macros

- #define MAX_I2C_BUSES 3

Typedefs

- typedef struct atcaI2Cmaster ATCAI2CMaster_t

  this is the hal_data for ATCA HAL for ASF SERCOM

Functions

- void change_i2c_speed (ATCAIface iface, uint32_t speed)

  method to change the bus speed of I2C

15.130.1 Detailed Description

ATCA Hardware abstraction layer for SAMV71 I2C over ASF drivers.

Prerequisite: add SERCOM I2C Master Polled support to application in Atmel Studio

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.131 hal_uc3_timer_asf.c File Reference

ATCA Hardware abstraction layer for SAM4S I2C over ASF drivers.

```
#include <asf.h>
#include <delay.h>
#include "atca_hal.h"
```

Functions

- void atca_delay_us (uint32_t delay)

  Timer API implemented at the HAL level.
- void atca_delay_10us (uint32_t delay)

  This function delays for a number of tens of microseconds.
- void atca_delay_ms (uint32_t delay)

  This function delays for a number of milliseconds.

15.131.1 Detailed Description

ATCA Hardware abstraction layer for SAM4S I2C over ASF drivers.

Prerequisite: add "Delay routines (service)" module to application in Atmel Studio

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.132 hal_win_kit_cdc.c File Reference

ATCA Hardware abstraction layer for Windows using kit protocol over a USB CDC device.

```
#include "atca_hal.h"
#include "kit_phy.h"
#include "hal_win_kit_cdc.h"
#include "kit_protocol.h"
#include <SetupAPI.h>
#include <stdlib.h>
#include <tchar.h>
#include <stdio.h>
```

Functions

- ATCA_STATUS hal_kit_cdc_init (void ∗hal, ATCAIfaceCfg ∗cfg)

  HAL implementation of Kit USB CDC init.
- ATCA_STATUS hal_cdc_discover_buses (int i2c_buses[ ], int max_buses)

  discover all CDC kits available.This function is currently not implemented. this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge
- ATCA_STATUS hal_cdc_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

  discover any CryptoAuth devices on a given logical bus number.This function is currently not implemented.
- ATCA_STATUS hal_kit_cdc_post_init (ATCAIface iface)

  HAL implementation of Kit USB CDC post init.
- ATCA_STATUS kit_phy_send (ATCAIface iface, const char ∗txdata, int txlength)

  HAL implementation of kit protocol send .It is called by the top layer.
- ATCA_STATUS kit_phy_receive (ATCAIface iface, char ∗rxdata, int ∗rxsize)

  HAL implementation of kit protocol receive data.It is called by the top layer.
- ATCA_STATUS hal_kit_phy_num_found (int8_t ∗num_found)

  Number of USB CDC devices found.
- ATCA_STATUS hal_kit_cdc_send (ATCAIface iface, uint8_t ∗txdata, int txlength)

  HAL implementation of kit protocol send over USB CDC.
- ATCA_STATUS hal_kit_cdc_receive (ATCAIface iface, uint8_t ∗rxdata, uint16_t ∗rxsize)

  HAL implementation of kit protocol receive over USB CDC.
- ATCA_STATUS hal_kit_cdc_wake (ATCAIface iface)

  Call the wake for kit protocol over USB CDC.
- ATCA_STATUS hal_kit_cdc_idle (ATCAIface iface)

  Call the idle for kit protocol over USB CDC.
- ATCA_STATUS hal_kit_cdc_sleep (ATCAIface iface)

Call the sleep for kit protocol over USB CDC.

- ATCA_STATUS hal_kit_cdc_release (void *hal_data)

  Close the physical port for CDC.

- ATCA_STATUS hal_kit_cdc_discover_buses (int cdc_buses[ ], int max_buses)

  discover cdc buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge.This function is currently not implemented.

- ATCA_STATUS hal_kit_cdc_discover_devices (int bus_num, ATCAIfaceCfg *cfg, int *found)

  discover any CryptoAuth devices on a given logical bus number

Variables

- atcacdc_t _gCdc

15.132.1 Detailed Description

ATCA Hardware abstraction layer for Windows using kit protocol over a USB CDC device.

Copyright

15.132.2 Function Documentation

15.132.2.1 hal_cdc_discover_buses()

```
ATCA_STATUS hal_cdc_discover_buses (
            int i2c_buses[ ],
            int max_buses )
```

discover all CDC kits available.This function is currently not implemented. this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge

Parameters

in	*i2c_buses*	- an array of logical bus numbers
in	*max_buses*	- maximum number of buses the app wants to attempt to discover

Returns

ATCA_UNIMPLEMENTED

15.132.2.2 hal_cdc_discover_devices()

```
ATCA_STATUS hal_cdc_discover_devices (
            int bus_num,
            ATCAIfaceCfg cfg[ ],
            int * found )
```

discover any CryptoAuth devices on a given logical bus number.This function is currently not implemented.

Parameters

in	*bus_num*	- logical bus number on which to look for CryptoAuth devices
out	*cfg[ ]*	- pointer to head of an array of interface config structures which get filled in by this method
out	*∗found*	- number of devices found on this bus

Returns

ATCA_UNIMPLEMENTED

15.132.2.3 hal_kit_cdc_discover_buses()

```
ATCA_STATUS hal_kit_cdc_discover_buses (
            int cdc_buses[ ],
            int max_buses )
```

discover cdc buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge.This function is currently not implemented.

Parameters

in	*cdc_buses*	- an array of logical bus numbers
in	*max_buses*	- maximum number of buses the app wants to attempt to discover

Returns

ATCA_UNIMPLEMENTED

15.132.2.4 hal_kit_cdc_discover_devices()

```
ATCA_STATUS hal_kit_cdc_discover_devices (
            int bus_num,
            ATCAIfaceCfg * cfg,
            int * found )
```

discover any CryptoAuth devices on a given logical bus number

Parameters

in	*bus_num*	- logical bus number on which to look for CryptoAuth devices
out	*cfg[ ]*	- pointer to head of an array of interface config structures which get filled in by this method
out	∗*found*	- number of devices found on this bus

Returns

ATCA_UNIMPLEMENTED

15.132.2.5 hal_kit_cdc_idle()

```
ATCA_STATUS hal_kit_cdc_idle (
            ATCAIface iface )
```

Call the idle for kit protocol over USB CDC.

Parameters

in	*iface*	ATCAIface instance that is the interface object to send the bytes over

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.132.2.6 hal_kit_cdc_init()

```
ATCA_STATUS hal_kit_cdc_init (
            void * hal,
            ATCAIfaceCfg * cfg )
```

HAL implementation of Kit USB CDC init.

Parameters

in	*hal*	pointer to HAL specific data that is maintained by this HAL
in	*cfg*	pointer to HAL specific configuration data that is used to initialize this HAL

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.132.2.7 hal_kit_cdc_post_init()

```
ATCA_STATUS hal_kit_cdc_post_init (
            ATCAIface iface )
```

HAL implementation of Kit USB CDC post init.

Parameters

in	*iface*	instance

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.132.2.8 hal_kit_cdc_receive()

```
ATCA_STATUS hal_kit_cdc_receive (
            ATCAIface iface,
            uint8_t * rxdata,
            uint16_t * rxsize )
```

HAL implementation of kit protocol receive over USB CDC.

Parameters

in	*iface*	Device to interact with.
out	*rxdata*	Data received will be returned here.
in,out	*rxsize*	As input, the size of the rxdata buffer. As output, the number of bytes received.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.132.2.9 hal_kit_cdc_release()

```
ATCA_STATUS hal_kit_cdc_release (
            void * hal_data )
```

Close the physical port for CDC.

Parameters

in	*hal_data*	The hardware abstraction data specific to this HAL

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.132.2.10 hal_kit_cdc_send()

```
ATCA_STATUS hal_kit_cdc_send (
            ATCAIface iface,
            uint8_t * txdata,
            int txlength )
```

HAL implementation of kit protocol send over USB CDC.

Parameters

in	*iface*	instance
in	*txdata*	pointer to bytes to send
in	*txlength*	number of bytes to send

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.132.2.11 hal_kit_cdc_sleep()

```
ATCA_STATUS hal_kit_cdc_sleep (
            ATCAIface iface )
```

Call the sleep for kit protocol over USB CDC.

Parameters

in	*iface*	ATCAIface instance that is the interface object to send the bytes over

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.132.2.12 hal_kit_cdc_wake()

```
ATCA_STATUS hal_kit_cdc_wake (
            ATCAIface iface )
```

Call the wake for kit protocol over USB CDC.

Parameters

in	*iface*	ATCAIface instance that is the interface object to send the bytes over

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.132.2.13 hal_kit_phy_num_found()

```
ATCA_STATUS hal_kit_phy_num_found (
            int8_t * num_found )
```

Number of USB CDC devices found.

Parameters

out	*num_found*	Number of USB CDC devices found returned here

Returns

ATCA_SUCCESS

15.132.2.14 kit_phy_receive()

```
ATCA_STATUS kit_phy_receive (
            ATCAIface iface,
            char * rxdata,
            int * rxsize )
```

HAL implementation of kit protocol receive data.It is called by the top layer.

Parameters

in	*iface*	instance
out	*rxdata*	pointer to space to receive the data
in,out	*rxsize*	ptr to expected number of receive bytes to request

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.132.2.15 kit_phy_send()

```
ATCA_STATUS kit_phy_send (
            ATCAIface iface,
            const char * txdata,
            int txlength )
```

HAL implementation of kit protocol send .It is called by the top layer.

Parameters

in	*iface*	instance
in	*txdata*	pointer to bytes to send
in	*txlength*	number of bytes to send

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.132.3 Variable Documentation

15.132.3.1 _gCdc

```
atcacdc_t _gCdc
```

15.133 hal_win_kit_cdc.h File Reference

ATCA Hardware abstraction layer for Windows using kit protocol over a USB CDC device.

```
#include <Windows.h>
```

Data Structures

- struct cdc_device
- struct atcacdc

Macros

- #define CDC_DEVICES_MAX 10
- #define CDC_BUFFER_MAX 1024

Typedefs

- typedef struct cdc_device cdc_device_t
- typedef struct atcacdc atcacdc_t

15.133.1 Detailed Description

ATCA Hardware abstraction layer for Windows using kit protocol over a USB CDC device.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.133.2 Macro Definition Documentation

15.133.2.1 CDC_BUFFER_MAX

```
#define CDC_BUFFER_MAX 1024
```

15.133.2.2 CDC_DEVICES_MAX

```
#define CDC_DEVICES_MAX 10
```

15.133.3 Typedef Documentation

15.133.3.1 atcacdc_t

```
typedef struct atcacdc atcacdc_t
```

15.133.3.2 cdc_device_t

```
typedef struct cdc_device cdc_device_t
```

15.134 hal_win_kit_hid.c File Reference

ATCA Hardware abstraction layer for Windows using kit protocol over a USB HID device.

```
#include "atca_hal.h"
#include "hal_win_kit_hid.h"
#include "kit_protocol.h"
#include "kit_phy.h"
#include <SetupAPI.h>
#include <stdio.h>
#include <stdlib.h>
#include <tchar.h>
```

Macros

- #define HID_GUID { 0x4d1e55b2, 0xf16f, 0x11cf, 0x88, 0xcb, 0x00, 0x11, 0x11, 0x00, 0x00, 0x30 }

Functions

- ATCA_STATUS hal_kit_hid_init (void ∗hal, ATCAIfaceCfg ∗cfg)

  HAL implementation of Kit USB HID init.
- ATCA_STATUS hal_kit_hid_discover_buses (int i2c_buses[ ], int max_buses)

  discover cdc buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge
- ATCA_STATUS hal_kit_hid_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

  discover any CryptoAuth devices on a given logical bus number
- ATCA_STATUS hal_kit_hid_post_init (ATCAIface iface)

  HAL implementation of Kit HID post init.
- ATCA_STATUS kit_phy_send (ATCAIface iface, const char ∗txdata, int txlength)

  HAL implementation of kit protocol send .It is called by the top layer.
- ATCA_STATUS kit_phy_receive (ATCAIface iface, char ∗rxdata, int ∗rxsize)

  HAL implementation of kit protocol receive data.It is called by the top layer.
- ATCA_STATUS kit_phy_num_found (int8_t ∗num_found)

  Number of USB HID devices found.
- ATCA_STATUS hal_kit_hid_send (ATCAIface iface, uint8_t ∗txdata, int txlength)

  HAL implementation of kit protocol send over USB HID.
- ATCA_STATUS hal_kit_hid_receive (ATCAIface iface, uint8_t ∗rxdata, uint16_t ∗rxsize)

  HAL implementation of send over USB HID.
- ATCA_STATUS hal_kit_hid_wake (ATCAIface iface)

  Call the wake for kit protocol.
- ATCA_STATUS hal_kit_hid_idle (ATCAIface iface)

  Call the idle for kit protocol.
- ATCA_STATUS hal_kit_hid_sleep (ATCAIface iface)

  Call the sleep for kit protocol.
- ATCA_STATUS hal_kit_hid_release (void ∗hal_data)

  Close the physical port for HID.

Variables

- atcahid_t _gHid

15.134.1 Detailed Description

ATCA Hardware abstraction layer for Windows using kit protocol over a USB HID device.

Copyright

15.135 hal_win_kit_hid.h File Reference

ATCA Hardware abstraction layer for Windows using kit protocol over a USB HID device.

```
#include <Windows.h>
```

Data Structures

- struct hid_device
- struct atcahid

Macros

- #define HID_DEVICES_MAX 10
- #define HID_PACKET_MAX 512

Typedefs

- typedef struct hid_device hid_device_t
- typedef struct atcahid atcahid_t

15.135.1 Detailed Description

ATCA Hardware abstraction layer for Windows using kit protocol over a USB HID device.

Copyright

15.136 hal_win_timer.c File Reference

ATCA Hardware abstraction layer for windows timer functions.

```
#include <windows.h>
#include <math.h>
#include "atca_hal.h"
```

Functions

- void atca_delay_us (uint32_t delay)

  Timer API implemented at the HAL level.
- void atca_delay_10us (uint32_t delay)

  This function delays for a number of tens of microseconds.
- void atca_delay_ms (uint32_t delay)

  This function delays for a number of milliseconds.

15.136.1 Detailed Description

ATCA Hardware abstraction layer for windows timer functions.

Copyright

15.137 hal_xmega_a3bu_i2c_asf.c File Reference

ATCA Hardware abstraction layer for XMEGA-A3BU I2C over ASF drivers.

```
#include <asf.h>
#include <string.h>
#include <stdio.h>
#include "atca_hal.h"
#include "hal_xmega_a3bu_i2c_asf.h"
#include "atca_device.h"
#include "atca_execution.h"
```

Functions

- ATCA_STATUS hal_i2c_discover_buses (int i2c_buses[ ], int max_buses)

  discover i2c buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the prior knowledge
- ATCA_STATUS hal_i2c_discover_devices (int bus_num, ATCAIfaceCfg cfg[ ], int ∗found)

  discover any CryptoAuth devices on a given logical bus number
- ATCA_STATUS hal_i2c_init (void ∗hal, ATCAIfaceCfg ∗cfg)

  initialize an I2C interface using given config
- ATCA_STATUS hal_i2c_post_init (ATCAIface iface)

  HAL implementation of I2C post init.
- ATCA_STATUS hal_i2c_send (ATCAIface iface, uint8_t ∗txdata, int txlength)

  HAL implementation of I2C send over ASF.
- ATCA_STATUS hal_i2c_receive (ATCAIface iface, uint8_t ∗rxdata, uint16_t ∗rxlength)

  HAL implementation of I2C receive function for ASF I2C.
- void change_i2c_speed (ATCAIface iface, uint32_t speed)

  method to change the bus speed of I2C
- ATCA_STATUS hal_i2c_wake (ATCAIface iface)

  wake up CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_idle (ATCAIface iface)

  idle CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_sleep (ATCAIface iface)

  sleep CryptoAuth device using I2C bus
- ATCA_STATUS hal_i2c_release (void ∗hal_data)

  manages reference count on given bus and releases resource if no more refences exist

15.137.1 Detailed Description

ATCA Hardware abstraction layer for XMEGA-A3BU I2C over ASF drivers.

Prerequisite: add I2C Master Polled support to application in Atmel Studio

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.138 hal_xmega_a3bu_i2c_asf.h File Reference

ATCA Hardware abstraction layer for XMEGA-A3BU I2C over ASF drivers.

```
#include <asf.h>
#include "twi_master.h"
```

Data Structures

- struct atcaI2Cmaster
  this is the hal_data for ATCA HAL created using ASF

Macros

- #define MAX_I2C_BUSES 4

Typedefs

- typedef struct atcaI2Cmaster ATCAI2CMaster_t
  this is the hal_data for ATCA HAL created using ASF

Functions

- void change_i2c_speed (ATCAIface iface, uint32_t speed)
  method to change the bus speed of I2C

15.138.1 Detailed Description

ATCA Hardware abstraction layer for XMEGA-A3BU I2C over ASF drivers.

Prerequisite: add I2C Master Polled support to application in Atmel Studio

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.139 hal_xmega_a3bu_timer_asf.c File Reference

ATCA Hardware abstraction layer for SAMD21 timer/delay over ASF drivers.

```
#include <asf.h>
#include <delay.h>
#include "atca_hal.h"
```

Functions

- void atca_delay_us (uint32_t delay)

  Timer API implemented at the HAL level.
- void atca_delay_10us (uint32_t delay)

  This function delays for a number of tens of microseconds.
- void atca_delay_ms (uint32_t delay)

  This function delays for a number of milliseconds.

15.139.1 Detailed Description

ATCA Hardware abstraction layer for SAMD21 timer/delay over ASF drivers.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.140 i2c_bitbang_samd21.c File Reference

Hardware Interface Functions - I2C bit-bang for SAMD21.

```
#include <asf.h>
#include <stdint.h>
#include "i2c_bitbang_samd21.h"
```

Macros

- #define DEFAULT_I2C_BUS 2

Functions

- void i2c_discover_buses (int i2c_bitbang_buses[ ], int max_buses)

    Assigns the logical bus number for discovering the devices.
- void i2c_set_pin (uint8_t sda, uint8_t scl)

    Set I2C data and clock pin. Other functions will use these pins.
- void i2c_enable (void)

    Configure GPIO pins for I2C clock and data as output.
- void i2c_disable (void)

    Configure GPIO pins for I2C clock and data as input.
- void i2c_send_start (void)

    Send a START condition.
- void i2c_send_ack (uint8_t ack)

    Send an ACK or NACK (after receive).
- void i2c_send_stop (void)

    Send a STOP condition.
- void i2c_send_wake_token (void)

    Send a Wake Token.
- ATCA_STATUS i2c_send_byte (uint8_t i2c_byte)

    Send one byte.
- ATCA_STATUS i2c_send_bytes (uint8_t count, uint8_t *data)

    Send a number of bytes.
- uint8_t i2c_receive_one_byte (uint8_t ack)

    Receive one byte (MSB first).
- void i2c_receive_byte (uint8_t *data)

    Receive one byte and send ACK.
- void i2c_receive_bytes (uint8_t count, uint8_t *data)

    Receive a number of bytes.

Variables

- I2CBuses i2c_buses_default
- uint8_t pin_sda
- uint8_t pin_scl

15.140.1 Detailed Description

Hardware Interface Functions - I2C bit-bang for SAMD21.

Copyright

15.140.2 Macro Definition Documentation

15.140.2.1 DEFAULT_I2C_BUS

```
#define DEFAULT_I2C_BUS 2
```

15.140.3 Function Documentation

15.140.3.1 i2c_disable()

```
void i2c_disable (
            void )
```

Configure GPIO pins for I2C clock and data as input.

15.140.3.2 i2c_discover_buses()

```
void i2c_discover_buses (
            int i2c_bitbang_buses[],
            int max_buses )
```

Assigns the logical bus number for discovering the devices.

Parameters

in	*i2c_bitbang_buses*	The logical bus numbers are assigned to the variables.
in	*max_buses*	Maximum number of bus used for discovering.

15.140.3.3 i2c_enable()

```
void i2c_enable (
            void )
```

Configure GPIO pins for I2C clock and data as output.

15.140.3.4 i2c_receive_byte()

```
void i2c_receive_byte (
            uint8_t * data )
```

Receive one byte and send ACK.

Parameters

out	*data*	pointer to received byte

15.140.3.5 i2c_receive_bytes()

```
void i2c_receive_bytes (
            uint8_t count,
            uint8_t * data )
```

Receive a number of bytes.

Parameters

out	*data*	pointer to receive buffer
in	*count*	number of bytes to receive

15.140.3.6 i2c_receive_one_byte()

```
uint8_t i2c_receive_one_byte (
            uint8_t ack )
```

Receive one byte (MSB first).

Parameters

in	*ack*	0:NACK, else:ACK

Returns

Number of bytes received

We don't need to delay after the last bit because it takes time to switch the pin to output for acknowledging.

15.140.3.7 i2c_send_ack()

```
void i2c_send_ack (
            uint8_t ack )
```

Send an ACK or NACK (after receive).

Parameters

in	*ack*	0: NACK, else: ACK

< Low data line indicates an ACK.

< High data line indicates a NACK.

Clock out acknowledgment.

15.140.3.8 i2c_send_byte()

```
ATCA_STATUS i2c_send_byte (
            uint8_t i2c_byte )
```

Send one byte.

Parameters

in	*i2c_byte*	byte to write

Returns

ATCA_STATUS

This avoids spikes but adds an if condition. We could parametrize the call to I2C_SET_OUTPUT and translate the msb to OUTSET or OUTCLR, but then the code would become target specific.

Send 8 bits of data.

Clock out the data bit.

Shifting while clock is high compensates for the time it takes to evaluate the bit while clock is low. That way, the low and high time of the clock pin is almost equal.

Clock in last data bit.

Set data line to be an input.

Wait for the ack.

15.140.3.9 i2c_send_bytes()

```
ATCA_STATUS i2c_send_bytes (
            uint8_t count,
            uint8_t * data )
```

Send a number of bytes.

Parameters

in	*count*	number of bytes to send
in	*data*	pointer to buffer containing bytes to send

Returns

ATCA_STATUS

15.140.3.10 i2c_send_start()

```
void i2c_send_start (
            void )
```

Send a START condition.

Set clock high in case we re-start.

15.140.3.11 i2c_send_stop()

```
void i2c_send_stop (
            void )
```

Send a STOP condition.

15.140.3.12 i2c_send_wake_token()

```
void i2c_send_wake_token (
            void )
```

Send a Wake Token.

15.140.3.13 i2c_set_pin()

```
void i2c_set_pin (
            uint8_t sda,
            uint8_t scl )
```

Set I2C data and clock pin. Other functions will use these pins.

Parameters

in	*sda*	definition of GPIO pin to be used as data pin
in	*scl*	definition of GPIO pin to be used as clock pin

15.140.4 Variable Documentation

15.140.4.1 i2c_buses_default

```
I2CBuses i2c_buses_default
```

Initial value:

```
= {
    {EXT3_PIN_3, EXT3_PIN_9,  EXT3_PIN_I2C_SDA, EXT3_PIN_13, EXT2_PIN_3, EXT2_PIN_5, EXT2_PIN_7, EXT2_PIN_9,
        EXT2_PIN_13, EXT2_PIN_15, EXT2_PIN_17, EXT1_PIN_3, EXT1_PIN_5, EXT1_PIN_7, EXT1_PIN_9,  EXT1_PIN_13,
       EXT1_PIN_15, EXT1_PIN_17},
    {EXT3_PIN_7, EXT3_PIN_10, EXT3_PIN_I2C_SCL, EXT3_PIN_14, EXT2_PIN_4, EXT2_PIN_6, EXT2_PIN_8,
       EXT2_PIN_10, EXT2_PIN_14, EXT2_PIN_16, EXT2_PIN_18, EXT1_PIN_4, EXT1_PIN_6, EXT1_PIN_8, EXT1_PIN_10,
       EXT1_PIN_14, EXT1_PIN_16, EXT1_PIN_18}
}
```

15.140.4.2 pin_scl

```
uint8_t pin_scl
```

15.140.4.3 pin_sda

```
uint8_t pin_sda
```

15.141 i2c_bitbang_samd21.h File Reference

definitions for bit-banged I2C

```
#include "atca_status.h"
#include <delay.h>
```

Data Structures

- struct I2CBuses

Macros

- #define MAX_I2C_BUSES 18
- #define I2C_ENABLE()
- #define I2C_DISABLE()
- #define I2C_CLOCK_LOW() port_pin_set_output_level(pin_scl, false)
- #define I2C_CLOCK_HIGH() port_pin_set_output_level(pin_scl, true)
- #define I2C_DATA_LOW() port_pin_set_output_level(pin_sda, false)
- #define I2C_DATA_HIGH() port_pin_set_output_level(pin_sda, true)
- #define I2C_DATA_IN() port_pin_get_input_level(pin_sda)
- #define I2C_SET_OUTPUT()
- #define I2C_SET_OUTPUT_HIGH() { I2C_SET_OUTPUT(); I2C_DATA_HIGH(); }
- #define I2C_SET_OUTPUT_LOW() { I2C_SET_OUTPUT(); I2C_DATA_LOW(); }
- #define I2C_SET_INPUT()
- #define DISABLE_INTERRUPT() cpu_irq_disable()
- #define ENABLE_INTERRUPT() cpu_irq_enable()
- #define I2C_CLOCK_DELAY_WRITE_LOW() delay_us(1)
- #define I2C_CLOCK_DELAY_WRITE_HIGH() delay_us(1)
- #define I2C_CLOCK_DELAY_READ_LOW() delay_us(1)
- #define I2C_CLOCK_DELAY_READ_HIGH() delay_us(1)
- #define I2C_CLOCK_DELAY_SEND_ACK() delay_us(1)
- #define I2C_HOLD_DELAY() delay_us(1)

  This delay is inserted to make the Start and Stop hold time at least 250 ns.
- #define I2C_ACK_TIMEOUT (4)

  loop count when waiting for an acknowledgment

Functions

- void i2c_set_pin (uint8_t sda, uint8_t scl)

  Set I2C data and clock pin. Other functions will use these pins.
- void i2c_discover_buses (int i2c_bitbang_buses[ ], int max_buses)

  Assigns the logical bus number for discovering the devices.
- void i2c_enable (void)

  Configure GPIO pins for I2C clock and data as output.
- void i2c_disable (void)

  Configure GPIO pins for I2C clock and data as input.
- void i2c_send_start (void)

  Send a START condition.
- void i2c_send_ack (uint8_t ack)

  Send an ACK or NACK (after receive).
- void i2c_send_stop (void)

  Send a STOP condition.
- void i2c_send_wake_token (void)

  Send a Wake Token.
- ATCA_STATUS i2c_send_byte (uint8_t i2c_byte)

  Send one byte.
- ATCA_STATUS i2c_send_bytes (uint8_t count, uint8_t ∗data)

  Send a number of bytes.
- uint8_t i2c_receive_one_byte (uint8_t ack)

  Receive one byte (MSB first).
- void i2c_receive_byte (uint8_t ∗data)

  Receive one byte and send ACK.
- void i2c_receive_bytes (uint8_t count, uint8_t ∗data)

  Receive a number of bytes.

Variables

- I2CBuses i2c_buses_default
- uint8_t pin_sda
- uint8_t pin_scl

15.141.1 Detailed Description

definitions for bit-banged I2C

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.141.2 Macro Definition Documentation

15.141.2.1 DISABLE_INTERRUPT

```
#define DISABLE_INTERRUPT( ) cpu_irq_disable()
```

15.141.2.2 ENABLE_INTERRUPT

```
#define ENABLE_INTERRUPT( ) cpu_irq_enable()
```

15.141.2.3 I2C_ACK_TIMEOUT

```
#define I2C_ACK_TIMEOUT (4)
```

loop count when waiting for an acknowledgment

15.141.2.4 I2C_CLOCK_DELAY_READ_HIGH

```
#define I2C_CLOCK_DELAY_READ_HIGH( ) delay_us(1)
```

15.141.2.5 I2C_CLOCK_DELAY_READ_LOW

```
#define I2C_CLOCK_DELAY_READ_LOW( ) delay_us(1)
```

15.141.2.6 I2C_CLOCK_DELAY_SEND_ACK

```
#define I2C_CLOCK_DELAY_SEND_ACK( ) delay_us(1)
```

15.141.2.7 I2C_CLOCK_DELAY_WRITE_HIGH

```
#define I2C_CLOCK_DELAY_WRITE_HIGH( ) delay_us(1)
```

15.141.2.8 I2C_CLOCK_DELAY_WRITE_LOW

```
#define I2C_CLOCK_DELAY_WRITE_LOW( ) delay_us(1)
```

15.141.2.9 I2C_CLOCK_HIGH

```
#define I2C_CLOCK_HIGH( ) port_pin_set_output_level(pin_scl, true)
```

15.141.2.10 I2C_CLOCK_LOW

```
#define I2C_CLOCK_LOW( ) port_pin_set_output_level(pin_scl, false)
```

15.141.2.11 I2C_DATA_HIGH

```
#define I2C_DATA_HIGH( ) port_pin_set_output_level(pin_sda, true)
```

15.141.2.12 I2C_DATA_IN

```
#define I2C_DATA_IN( ) port_pin_get_input_level(pin_sda)
```

15.141.2.13 I2C_DATA_LOW

```
#define I2C_DATA_LOW( ) port_pin_set_output_level(pin_sda, false)
```

15.141.2.14 I2C_DISABLE

```
#define I2C_DISABLE( )
```

Value:

```
{  struct port_config pin_conf; \
                                                   port_get_config_defaults(&pin_conf); \
                                                   pin_conf.direction  = PORT_PIN_DIR_INPUT; \
                                                   pin_conf.input_pull = PORT_PIN_PULL_UP; \
                                                   port_pin_set_config(pin_sda, &pin_conf); \
                                                   port_pin_set_config(pin_scl, &pin_conf); }
```

15.141.2.15 I2C_ENABLE

```
#define I2C_ENABLE( )
```

Value:

```
{  struct port_config pin_conf; \
                                                  port_get_config_defaults(&pin_conf); \
                                                  pin_conf.direction  = PORT_PIN_DIR_OUTPUT_WTH_READBACK; \
                                                  port_pin_set_config(pin_sda, &pin_conf); \
                                                  pin_conf.direction  = PORT_PIN_DIR_OUTPUT; \
                                                  port_pin_set_config(pin_scl, &pin_conf); }
```

15.141.2.16 I2C_HOLD_DELAY

```
#define I2C_HOLD_DELAY( ) delay_us(1)
```

This delay is inserted to make the Start and Stop hold time at least 250 ns.

15.141.2.17 I2C_SET_INPUT

```
#define I2C_SET_INPUT( )
```

Value:

```
{  struct port_config pin_conf; \
                                                   port_get_config_defaults(&pin_conf); \
                                                   pin_conf.direction  = PORT_PIN_DIR_INPUT; \
                                                   port_pin_set_config(pin_sda, &pin_conf); }
```

15.141.2.18 I2C_SET_OUTPUT

```
#define I2C_SET_OUTPUT( )
```

Value:

```
{  struct port_config pin_conf; \
                                    port_get_config_defaults(&pin_conf); \
                                    pin_conf.direction  = PORT_PIN_DIR_OUTPUT_WTH_READBACK; \
                                    port_pin_set_config(pin_sda, &pin_conf); }
```

15.141.2.19 I2C_SET_OUTPUT_HIGH

```
#define I2C_SET_OUTPUT_HIGH( ) { I2C_SET_OUTPUT(); I2C_DATA_HIGH(); }
```

15.141.2.20 I2C_SET_OUTPUT_LOW

```
#define I2C_SET_OUTPUT_LOW( ) { I2C_SET_OUTPUT(); I2C_DATA_LOW(); }
```

15.141.2.21 MAX_I2C_BUSES

```
#define MAX_I2C_BUSES 18
```

15.141.3 Function Documentation

15.141.3.1 i2c_disable()

```
void i2c_disable (
            void )
```

Configure GPIO pins for I2C clock and data as input.

15.141.3.2 i2c_discover_buses()

```
void i2c_discover_buses (
            int i2c_bitbang_buses[],
            int max_buses )
```

Assigns the logical bus number for discovering the devices.

Parameters

in	*i2c_bitbang_buses*	The logical bus numbers are assigned to the variables.
in	*max_buses*	Maximum number of bus used for discovering.

15.141.3.3 i2c_enable()

```
void i2c_enable (
            void )
```

Configure GPIO pins for I2C clock and data as output.

15.141.3.4 i2c_receive_byte()

```
void i2c_receive_byte (
            uint8_t * data )
```

Receive one byte and send ACK.

Parameters

out	*data*	pointer to received byte

15.141.3.5 i2c_receive_bytes()

```
void i2c_receive_bytes (
            uint8_t count,
            uint8_t * data )
```

Receive a number of bytes.

Parameters

out	*data*	pointer to receive buffer
in	*count*	number of bytes to receive

15.141.3.6 i2c_receive_one_byte()

```
uint8_t i2c_receive_one_byte (
            uint8_t ack )
```

Receive one byte (MSB first).

Parameters

in	*ack*	0:NACK, else:ACK

Returns

Number of bytes received

We don't need to delay after the last bit because it takes time to switch the pin to output for acknowledging.

15.141.3.7 i2c_send_ack()

```
void i2c_send_ack (
            uint8_t ack )
```

Send an ACK or NACK (after receive).

Parameters

in	*ack*	0: NACK, else: ACK

< Low data line indicates an ACK.

< High data line indicates a NACK.

Clock out acknowledgment.

15.141.3.8 i2c_send_byte()

```
ATCA_STATUS i2c_send_byte (
            uint8_t i2c_byte )
```

Send one byte.

Parameters

in	*i2c_byte*	byte to write

Returns

ATCA_STATUS

This avoids spikes but adds an if condition. We could parametrize the call to I2C_SET_OUTPUT and translate the msb to OUTSET or OUTCLR, but then the code would become target specific.

Send 8 bits of data.

Clock out the data bit.

Shifting while clock is high compensates for the time it takes to evaluate the bit while clock is low. That way, the low and high time of the clock pin is almost equal.

Clock in last data bit.

Set data line to be an input.

Wait for the ack.

15.141.3.9 i2c_send_bytes()

```
ATCA_STATUS i2c_send_bytes (
            uint8_t count,
            uint8_t * data )
```

Send a number of bytes.

Parameters

in	*count*	number of bytes to send
in	*data*	pointer to buffer containing bytes to send

Returns

ATCA_STATUS

15.141.3.10 i2c_send_start()

```
void i2c_send_start (
            void )
```

Send a START condition.

Set clock high in case we re-start.

15.141.3.11 i2c_send_stop()

```
void i2c_send_stop (
            void )
```

Send a STOP condition.

15.141.3.12 i2c_send_wake_token()

```
void i2c_send_wake_token (
            void )
```

Send a Wake Token.

15.141.3.13 i2c_set_pin()

```
void i2c_set_pin (
            uint8_t sda,
            uint8_t scl )
```

Set I2C data and clock pin. Other functions will use these pins.

Parameters

in	*sda*	definition of GPIO pin to be used as data pin
in	*scl*	definition of GPIO pin to be used as clock pin

15.141.4 Variable Documentation

15.141.4.1 i2c_buses_default

```
I2CBuses i2c_buses_default
```

15.141.4.2 pin_scl

```
uint8_t pin_scl
```

15.141.4.3 pin_sda

```
uint8_t pin_sda
```

15.142 io_protection_key.h File Reference

Provides required interface to access IO protection key.

```
#include "atca_status.h"
```

Functions

- ATCA_STATUS io_protection_get_key (uint8_t ∗io_key)
- ATCA_STATUS io_protection_set_key (uint8_t ∗io_key)

15.142.1 Detailed Description

Provides required interface to access IO protection key.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.142.2 Function Documentation

15.142.2.1 io_protection_get_key()

```
ATCA_STATUS io_protection_get_key (
            uint8_t * io_key )
```

15.142.2.2 io_protection_set_key()

```
ATCA_STATUS io_protection_set_key (
            uint8_t * io_key )
```

15.143 kit_phy.h File Reference

ATCA Hardware abstraction layer physical send & receive function definitions.

```
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS kit_phy_num_found (int8_t ∗num_found)

  Number of USB HID devices found.
- ATCA_STATUS kit_phy_send (ATCAIface iface, const char ∗txdata, int txlength)

  HAL implementation of kit protocol send .It is called by the top layer.
- ATCA_STATUS kit_phy_receive (ATCAIface iface, char ∗rxdata, int ∗rxsize)

  HAL implementation of kit protocol receive data.It is called by the top layer.

15.143.1 Detailed Description

ATCA Hardware abstraction layer physical send & receive function definitions.

This is included for kit protocol implementations. It is included in the kit protocol callback to actually send and recieve bytes.

Copyright

15.144 kit_protocol.c File Reference

Microchip Crypto Auth hardware interface object.

```
#include <stdlib.h>
#include <stdio.h>
#include "kit_phy.h"
#include "kit_protocol.h"
#include "basic/atca_helpers.h"
```

Macros

- #define KIT_MAX_SCAN_COUNT 4
- #define KIT_MAX_TX_BUF 32

Functions

- char ∗ strnchr (const char ∗s, size_t count, int c)
- const char ∗ kit_id_from_devtype (ATCADeviceType devtype)
- const char ∗ kit_interface_from_kittype (ATCAKitType kittype)
- ATCA_STATUS kit_init (ATCAIface iface)

  HAL implementation of kit protocol init. This function calls back to the physical protocol to send the bytes.
- ATCA_STATUS kit_send (ATCAIface iface, const uint8_t ∗txdata, int txlength)

  HAL implementation of kit protocol send. This function calls back to the physical protocol to send the bytes.
- ATCA_STATUS kit_receive (ATCAIface iface, uint8_t ∗rxdata, uint16_t ∗rxsize)

  HAL implementation to receive bytes and unwrap from kit protocol. This function calls back to the physical protocol to receive the bytes.
- ATCA_STATUS kit_wake (ATCAIface iface)

  Call the wake for kit protocol.
- ATCA_STATUS kit_idle (ATCAIface iface)

  Call the idle for kit protocol.
- ATCA_STATUS kit_sleep (ATCAIface iface)

  Call the sleep for kit protocol.
- ATCA_STATUS kit_wrap_cmd (const uint8_t ∗txdata, int txlen, char ∗pkitcmd, int ∗nkitcmd, char target)

  Wrap binary bytes in ascii kit protocol.
- ATCA_STATUS kit_parse_rsp (const char ∗pkitbuf, int nkitbuf, uint8_t ∗kitstatus, uint8_t ∗rxdata, int ∗datasize)

  Parse the response ascii from the kit.

15.144.1 Detailed Description

Microchip Crypto Auth hardware interface object.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.145 kit_protocol.h File Reference

```
#include "cryptoauthlib.h"
```

Macros

- #define KIT_TX_WRAP_SIZE (7)
- #define KIT_MSG_SIZE (32)
- #define KIT_RX_WRAP_SIZE (KIT_MSG_SIZE + 6)

Functions

- ATCA_STATUS kit_init (ATCAIface iface)
  HAL implementation of kit protocol init. This function calls back to the physical protocol to send the bytes.
- ATCA_STATUS kit_send (ATCAIface iface, const uint8_t *txdata, int txlength)
  HAL implementation of kit protocol send. This function calls back to the physical protocol to send the bytes.
- ATCA_STATUS kit_receive (ATCAIface iface, uint8_t *rxdata, uint16_t *rxsize)
  HAL implementation to receive bytes and unwrap from kit protocol. This function calls back to the physical protocol to receive the bytes.
- ATCA_STATUS kit_wrap_cmd (const uint8_t *txdata, int txlen, char *pkitcmd, int *nkitcmd, char target)
  Wrap binary bytes in ascii kit protocol.
- ATCA_STATUS kit_parse_rsp (const char *pkitbuf, int nkitbuf, uint8_t *kitstatus, uint8_t *rxdata, int *datasize)
  Parse the response ascii from the kit.
- ATCA_STATUS kit_wake (ATCAIface iface)
  Call the wake for kit protocol.
- ATCA_STATUS kit_idle (ATCAIface iface)
  Call the idle for kit protocol.
- ATCA_STATUS kit_sleep (ATCAIface iface)
  Call the sleep for kit protocol.

15.145.1 Detailed Description

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.146 license.txt File Reference

Functions

- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party software (including open source software) that may accompany Microchip software. THIS SOFTWARE IS SUPPLIED BY MICROCHIP "AS IS". NO WARRANTIES
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E↩XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE↩NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICRO↩CHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer ∗Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution ∗Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY T↩HE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR P↩URPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUR↩EMENT OF SUBSTITUTE GOODS OR SERVICES;LOSS OF USE, DATA, OR PROFITS;OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E↩XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE↩NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICRO↩CHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer ∗Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution ∗Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY T↩HE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR P↩URPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY OR CONSEQUENTIAL WHETHER IN STRICT OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE
- either version of the or (at your option) any later version. systemd is distributed in the hope that it will be useful

Variables

- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these terms

- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER EXPRESS
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR STATUTORY
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS SOFTWARE
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON INFRINGEMENT
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON MERCHANTABILITY
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY SPECIAL
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY PUNITIVE
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL LOSS
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL DAMAGE
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO T↩ HIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER CAUSED
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E↩ XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLOWED BY LAW

- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E← XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE← NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF FEES
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E← XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE← NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF ANY
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E← XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE← NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROC← HIP FOR THIS Alan Ott
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E← XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE← NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROC← HIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary forms
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E← XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE← NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROC← HIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without modification
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E← XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE← NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROC←

HIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are met

- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E↩ XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE↩ NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICRO↩ CHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer ∗Redistributions in binary form must reproduce the above copyright notice
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E↩ XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE↩ NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICRO↩ CHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer ∗Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution ∗Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED WARRANTIES
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUD↩ ING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer ∗Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution ∗Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED INCLUDING
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E↩ XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE↩ NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICRO↩ CHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer ∗Redistributions in binary form must reproduce the above copyright this list of conditions and the

following disclaimer in the documentation and or other materials provided with the distribution ∗Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY T←HE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED TO

- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E←XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE←NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICRO←CHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer ∗Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution ∗Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY T←HE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR P←URPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E←XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE←NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICRO←CHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer ∗Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution ∗Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY T←HE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR P←URPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY INCIDENTAL
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E←XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE←NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICRO←CHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer ∗Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution ∗Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY T←

HE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR P↩URPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY EXEMPLARY

- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E↩XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE↩NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICRO↩CHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer ∗Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution ∗Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY T↩HE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR P↩URPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY OR CONSEQUENTIAL WHETHER IN CONTRACT
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR E↩XPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTE↩NT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICRO↩CHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer ∗Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution ∗Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY T↩HE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR P↩URPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY OR CONSEQUENTIAL WHETHER IN STRICT LIABILITY
- c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO T↩HIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS B↩EEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EX↩TENT ALLOWED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer ∗Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution ∗Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY THE COPY↩RIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED

THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY OR CONSEQUENTIAL WHETHER IN STRICT OR EVEN IF ADVISED OF THE POSSIBILITY OF SUCH this code depends on the libudev h header file with the following license

- you can redistribute it and or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation
- either version of the License
- either version of the but WITHOUT ANY WARRANTY
- without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE See the GNU Lesser General Public License for more details You should have received a copy of the GNU Lesser General Public License along with systemd
- If not

15.146.1 Function Documentation

15.146.1.1 DAMAGES()

```
c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may
use Microchip software and any derivatives exclusively with Microchip products It is your
responsibility to comply with third party license terms applicable to your use of third party
WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A
PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTI←
AL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS B←
EEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLO←
WED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WI←
LL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal
Software All rights reserved Redistribution and use in source and binary with or without are
permitted provided that the following conditions are this list of conditions and the following
disclaimer* Redistributions in binary form must reproduce the above copyright this list of
conditions and the following disclaimer in the documentation and or other materials provided
with the distribution* Neither the name of Signal Software nor the names of its contributors
may be used to endorse or promote products derived from this software without specific prior
written permission THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS
AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FIT←
NESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTR←
IBUTORS BE LIABLE FOR ANY OR CONSEQUENTIAL DAMAGES (
            INCLUDING ,
            BUT NOT LIMITED TO,
            PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;LOSS OF USE,
            DATA ,
            OR PROFITS;OR BUSINESS INTERRUPTION )
```

15.146.1.2 or()

```
either version of the or (
            at your option )
```

15.146.1.3 software()

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party software (
including open source *software*)

15.146.1.4 TORT()

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTI↩ AL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS B↩ EEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLO↩ WED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WI↩ LL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer* Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution* Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FIT↩ NESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTR↩ IBUTORS BE LIABLE FOR ANY OR CONSEQUENTIAL WHETHER IN STRICT OR TORT (
INCLUDING NEGLIGENCE OR *OTHERWISE*)

15.146.2 Variable Documentation

15.146.2.1 ANY

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A P↩ ARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL C↩ OST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADV↩ ISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLOWED BY MICR↩ OCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF ANY

15.146.2.2 CAUSED

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A P↩
ARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL C↩
OST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER CAUSED

15.146.2.3 CONTRACT

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTI↩
AL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS B↩
EEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLO↩
WED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WI↩
LL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer* Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution* Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FIT↩
NESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTR↩
IBUTORS BE LIABLE FOR ANY OR CONSEQUENTIAL WHETHER IN CONTRACT

15.146.2.4 DAMAGE

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTI↩
AL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS B↩
EEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLO↩
WED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WI↩
LL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer *Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution *Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS

AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FIT↩
NESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTR↩
IBUTORS BE LIABLE FOR ANY OR CONSEQUENTIAL WHETHER IN STRICT OR EVEN IF ADVISED OF THE POSSI↩
BILITY OF SUCH DAMAGE

Initial value:

```
==============================================================================
If using the Linux HID driver (lib/hal/hal_linux_kit_hid.c)
```

15.146.2.5 DIRECT

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTI↩
AL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS B↩
EEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLO↩
WED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WI↩
LL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer* Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution* Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FIT↩
NESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTR↩
IBUTORS BE LIABLE FOR ANY DIRECT

15.146.2.6 EXEMPLARY

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTI↩
AL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS B↩
EEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLO↩
WED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WI↩
LL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer* Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution* Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FIT↩
NESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTR↩
IBUTORS BE LIABLE FOR ANY EXEMPLARY

15.146.2.7 EXPRESS

```
c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may
use Microchip software and any derivatives exclusively with Microchip products It is your
responsibility to comply with third party license terms applicable to your use of third party
WHETHER EXPRESS
```

15.146.2.8 FEES

```
c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may
use Microchip software and any derivatives exclusively with Microchip products It is your
responsibility to comply with third party license terms applicable to your use of third party
WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A P←
ARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL C←
OST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADV←
ISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLOWED BY MICR←
OCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE
AMOUNT OF FEES
```

15.146.2.9 forms

```
c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may
use Microchip software and any derivatives exclusively with Microchip products It is your
responsibility to comply with third party license terms applicable to your use of third party
WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A P←
ARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL C←
OST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADV←
ISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLOWED BY MICR←
OCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE
AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights
reserved Redistribution and use in source and binary forms
```

15.146.2.10 Foundation

```
you can redistribute it and or modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation
```

15.146.2.11 INCIDENTAL

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTI↩ AL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS B↩ EEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLO↩ WED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WI↩ LL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer* Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution* Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FIT↩ NESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTR↩ IBUTORS BE LIABLE FOR ANY INCIDENTAL

15.146.2.12 INCLUDING

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTI↩ AL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS B↩ EEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLO↩ WED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WI↩ LL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer* Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution* Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED INCLUDING

15.146.2.13 INDIRECT

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A

PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTI←
AL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS B←
EEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLO←
WED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WI←
LL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer *Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution *Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FIT←
NESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTR←
IBUTORS BE LIABLE FOR ANY INDIRECT

15.146.2.14 INFRINGEMENT

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON INFRINGEMENT

15.146.2.15 LAW

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A P←
ARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL C←
OST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADVI←
SED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLOWED BY LAW

15.146.2.16 LIABILITY

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTI←
AL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS B←
EEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLO←
WED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WI←
LL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer* Redistributions in binary form must reproduce the above copyright this list of

conditions and the following disclaimer in the documentation and or other materials provided with the distribution* Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FIT↩ NESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTR↩ IBUTORS BE LIABLE FOR ANY OR CONSEQUENTIAL WHETHER IN STRICT LIABILITY

15.146.2.17 license

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTI↩ AL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS B↩ EEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLO↩ WED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WI↩ LL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer* Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution* Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FIT↩ NESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTR↩ IBUTORS BE LIABLE FOR ANY OR CONSEQUENTIAL WHETHER IN STRICT OR EVEN IF ADVISED OF THE POSSI↩ BILITY OF SUCH this code depends on the libudev h header file with the following license

15.146.2.18 License

either version of the License

15.146.2.19 LOSS

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A P↩ ARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL L↩ OSS

15.146.2.20 MERCHANTABILITY

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON MERCHANTABILITY

15.146.2.21 met

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A P↩ARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL C↩OST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADV↩ISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLOWED BY MICR↩OCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are met

15.146.2.22 modification

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A P↩ARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL C↩OST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADV↩ISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLOWED BY MICR↩OCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without modification

15.146.2.23 not

If not

15.146.2.24 notice

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTI←AL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS B←EEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLO←WED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WI←LL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer* Redistributions in binary form must reproduce the above copyright notice

15.146.2.25 Ott

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A P←ARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL C←OST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADV←ISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLOWED BY MICR←OCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Ott

15.146.2.26 PUNITIVE

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A P←ARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY PUNITIVE

15.146.2.27 SOFTWARE

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A P←ARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL C←OST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS BEEN ADV←ISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLOWED BY MICR←OCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WILL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE

Initial value:

```
==============================================================================
If using the cross-platform HID driver (lib/hal/hal_all_platforms_kit_hidapi.c)
this code depends on the hidapi library with the following license:
Copyright (c) 2010
```

15.146.2.28 SPECIAL

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTI←AL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS B←EEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLO←WED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WI←LL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal Software All rights reserved Redistribution and use in source and binary with or without are permitted provided that the following conditions are this list of conditions and the following disclaimer *Redistributions in binary form must reproduce the above copyright this list of conditions and the following disclaimer in the documentation and or other materials provided with the distribution *Neither the name of Signal Software nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FIT←NESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTR←IBUTORS BE LIABLE FOR ANY SPECIAL

15.146.2.29 STATUTORY

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may use Microchip software and any derivatives exclusively with Microchip products It is your responsibility to comply with third party license terms applicable to your use of third party WHETHER IMPLIED OR STATUTORY

15.146.2.30 systemd

without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE See the GNU Lesser General Public License for more details You should have received a copy of the GNU Lesser General Public License along with systemd

15.146.2.31 terms

c Microchip Technology Inc and its subsidiaries Subject to your compliance with these terms

15.146.2.32 TO

```
c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may
use Microchip software and any derivatives exclusively with Microchip products It is your
responsibility to comply with third party license terms applicable to your use of third party
WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A
PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTI←↩
AL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS B←↩
EEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLO←↩
WED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WI←↩
LL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal
Software All rights reserved Redistribution and use in source and binary with or without are
permitted provided that the following conditions are this list of conditions and the following
disclaimer* Redistributions in binary form must reproduce the above copyright this list of
conditions and the following disclaimer in the documentation and or other materials provided
with the distribution* Neither the name of Signal Software nor the names of its contributors
may be used to endorse or promote products derived from this software without specific prior
written permission THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS
AND ANY EXPRESS OR IMPLIED BUT NOT LIMITED TO
```

15.146.2.33 WARRANTIES

```
c Microchip Technology Inc and its subsidiaries Subject to your compliance with these you may
use Microchip software and any derivatives exclusively with Microchip products It is your
responsibility to comply with third party license terms applicable to your use of third party
WHETHER IMPLIED OR APPLY TO THIS INCLUDING ANY IMPLIED WARRANTIES OF NON AND FITNESS FOR A
PARTICULAR PURPOSE IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTI←↩
AL COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE HOWEVER EVEN IF MICROCHIP HAS B←↩
EEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE TO THE FULLEST EXTENT ALLO←↩
WED BY MICROCHIP S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THIS SOFTWARE WI←↩
LL NOT EXCEED THE AMOUNT OF IF THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THIS Alan Signal
Software All rights reserved Redistribution and use in source and binary with or without are
permitted provided that the following conditions are this list of conditions and the following
disclaimer* Redistributions in binary form must reproduce the above copyright this list of
conditions and the following disclaimer in the documentation and or other materials provided
with the distribution* Neither the name of Signal Software nor the names of its contributors
may be used to endorse or promote products derived from this software without specific prior
written permission THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS AS IS
AND ANY EXPRESS OR IMPLIED WARRANTIES
```

15.146.2.34 WARRANTY

```
either version of the but WITHOUT ANY WARRANTY
```

15.147 README.md File Reference

15.148 README.md File Reference

15.149 README.md File Reference

15.150 README.md File Reference

15.151 README.md File Reference

15.152 README.md File Reference

15.153 readme.md File Reference

15.154 README.md File Reference

15.155 secure_boot.c File Reference

Provides required APIs to manage secure boot under various scenarios.

```
#include <string.h>
#include "secure_boot.h"
#include "io_protection_key.h"
#include "basic/atca_basic.h"
```

Functions

- ATCA_STATUS secure_boot_process (void)

  Handles secure boot functionality through initialization, execution, and de-initialization.
- ATCA_STATUS bind_host_and_secure_element_with_io_protection (uint16_t slot)

  Binds host MCU and Secure element with IO protection key.

15.155.1 Detailed Description

Provides required APIs to manage secure boot under various scenarios.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.155.2 Function Documentation

15.155.2.1 bind_host_and_secure_element_with_io_protection()

```
ATCA_STATUS bind_host_and_secure_element_with_io_protection (
            uint16_t slot )
```

Binds host MCU and Secure element with IO protection key.

Parameters

in	*slot*	The slot number of IO protection Key.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.155.2.2 secure_boot_process()

```
ATCA_STATUS secure_boot_process (
            void )
```

Handles secure boot functionality through initialization, execution, and de-initialization.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.156 secure_boot.h File Reference

Provides required APIs to manage secure boot under various scenarios.

```
#include "atca_status.h"
#include "secure_boot_memory.h"
#include "atca_command.h"
#include "crypto/atca_crypto_sw_sha2.h"
```

Data Structures

- struct secure_boot_config_bits
- struct secure_boot_parameters

Macros

- #define SECURE_BOOT_CONFIG_DISABLE 0
- #define SECURE_BOOT_CONFIG_FULL_BOTH 1
- #define SECURE_BOOT_CONFIG_FULL_SIGN 2
- #define SECURE_BOOT_CONFIG_FULL_DIG 3
- #define SECURE_BOOT_CONFIGURATION SECURE_BOOT_CONFIG_FULL_DIG
- #define SECURE_BOOT_DIGEST_ENCRYPT_ENABLED true
- #define SECURE_BOOT_UPGRADE_SUPPORT true

Functions

- ATCA_STATUS secure_boot_process (void)

    Handles secure boot functionality through initialization, execution, and de-initialization.
- ATCA_STATUS bind_host_and_secure_element_with_io_protection (uint16_t slot)

    Binds host MCU and Secure element with IO protection key.
- ATCA_STATUS host_generate_random_number (uint8_t *rand)

15.156.1 Detailed Description

Provides required APIs to manage secure boot under various scenarios.

Copyright

15.156.2 Macro Definition Documentation

15.156.2.1 SECURE_BOOT_CONFIG_DISABLE

```
#define SECURE_BOOT_CONFIG_DISABLE 0
```

15.156.2.2 SECURE_BOOT_CONFIG_FULL_BOTH

```
#define SECURE_BOOT_CONFIG_FULL_BOTH 1
```

15.156.2.3 SECURE_BOOT_CONFIG_FULL_DIG

```
#define SECURE_BOOT_CONFIG_FULL_DIG 3
```

15.156.2.4 SECURE_BOOT_CONFIG_FULL_SIGN

```
#define SECURE_BOOT_CONFIG_FULL_SIGN 2
```

15.156.2.5 SECURE_BOOT_CONFIGURATION

```
#define SECURE_BOOT_CONFIGURATION SECURE_BOOT_CONFIG_FULL_DIG
```

15.156.2.6 SECURE_BOOT_DIGEST_ENCRYPT_ENABLED

```
#define SECURE_BOOT_DIGEST_ENCRYPT_ENABLED true
```

15.156.2.7 SECURE_BOOT_UPGRADE_SUPPORT

```
#define SECURE_BOOT_UPGRADE_SUPPORT true
```

15.156.3 Function Documentation

15.156.3.1 bind_host_and_secure_element_with_io_protection()

```
ATCA_STATUS bind_host_and_secure_element_with_io_protection (
             uint16_t slot )
```

Binds host MCU and Secure element with IO protection key.

Parameters

in	*slot*	The slot number of IO protection Key.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.156.3.2 host_generate_random_number()

```
ATCA_STATUS host_generate_random_number (
             uint8_t * rand )
```

15.156.3.3 secure_boot_process()

```
ATCA_STATUS secure_boot_process (
            void  )
```

Handles secure boot functionality through initialization, execution, and de-initialization.

Returns

ATCA_SUCCESS on success, otherwise an error code.

15.157 secure_boot_memory.h File Reference

Provides interface to memory component for the secure boot.

```
#include "atca_status.h"
#include "atca_command.h"
```

Data Structures

- struct memory_parameters

Functions

- ATCA_STATUS secure_boot_init_memory (memory_parameters *memory_params)
- ATCA_STATUS secure_boot_read_memory (uint8_t *pu8_data, uint32_t *pu32_target_length)
- ATCA_STATUS secure_boot_write_memory (uint8_t *pu8_data, uint32_t *pu32_target_length)
- void secure_boot_deinit_memory (memory_parameters *memory_params)
- ATCA_STATUS secure_boot_mark_full_copy_completion (void)
- bool secure_boot_check_full_copy_completion (void)

15.157.1 Detailed Description

Provides interface to memory component for the secure boot.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.157.2 Function Documentation

15.157.2.1 secure_boot_check_full_copy_completion()

```
bool secure_boot_check_full_copy_completion (
            void )
```

15.157.2.2 secure_boot_deinit_memory()

```
void secure_boot_deinit_memory (
            memory_parameters * memory_params )
```

15.157.2.3 secure_boot_init_memory()

```
ATCA_STATUS secure_boot_init_memory (
            memory_parameters * memory_params )
```

15.157.2.4 secure_boot_mark_full_copy_completion()

```
ATCA_STATUS secure_boot_mark_full_copy_completion (
            void )
```

15.157.2.5 secure_boot_read_memory()

```
ATCA_STATUS secure_boot_read_memory (
            uint8_t * pu8_data,
            uint32_t * pu32_target_length )
```

15.157.2.6 secure_boot_write_memory()

```
ATCA_STATUS secure_boot_write_memory (
            uint8_t * pu8_data,
            uint32_t * pu32_target_length )
```

15.158 sha1_routines.c File Reference

Software implementation of the SHA1 algorithm.

```
#include "sha1_routines.h"
#include <string.h>
#include "atca_compiler.h"
```

Functions

- void CL_hashInit (CL_HashContext *ctx)

    Initialize context for performing SHA1 hash in software.
- void CL_hashUpdate (CL_HashContext *ctx, const U8 *src, int nbytes)

    Add arbitrary data to a SHA1 hash.
- void CL_hashFinal (CL_HashContext *ctx, U8 *dest)

    Complete the SHA1 hash in software and return the digest.
- void CL_hash (U8 *msg, int msgBytes, U8 *dest)

    Perform SHA1 hash of data in software.
- void shaEngine (U32 *buf, U32 *h)

15.158.1 Detailed Description

Software implementation of the SHA1 algorithm.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.158.2 Function Documentation

15.158.2.1 CL_hash()

```
void CL_hash (
            U8 * msg,
            int msgBytes,
            U8 * dest )
```

Perform SHA1 hash of data in software.

Parameters

in	*msg*	Data to be hashed
in	*msgBytes*	Data size in bytes
out	*dest*	Digest is returned here (20 bytes)

15.158.2.2 CL_hashFinal()

```
void CL_hashFinal (
            CL_HashContext * ctx,
            U8 * dest )
```

Complete the SHA1 hash in software and return the digest.

Parameters

in	*ctx*	Hash context
out	*dest*	Digest is returned here (20 bytes)

15.158.2.3 CL_hashInit()

```
void CL_hashInit (
            CL_HashContext * ctx )
```

Initialize context for performing SHA1 hash in software.

Parameters

in	*ctx*	Hash context

15.158.2.4 CL_hashUpdate()

```
void CL_hashUpdate (
            CL_HashContext * ctx,
            const U8 * src,
            int nbytes )
```

Add arbitrary data to a SHA1 hash.

Parameters

in	*ctx*	Hash context
in	*src*	Data to be added to the hash
in	*nbytes*	Data size in bytes

15.158.2.5 shaEngine()

```
void shaEngine (
            U32 * buf,
            U32 * h )
```

15.159 sha1_routines.h File Reference

Software implementation of the SHA1 algorithm.

```
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <stdint.h>
```

Data Structures

- struct CL_HashContext

Macros

- #define U8 uint8_t
- #define U16 uint16_t
- #define U32 uint32_t
- #define memcpy_P memmove
- #define strcpy_P strcpy
- #define _WDRESET()
- #define _NOP()
- #define leftRotate(x, n) (x) = (((x) << (n)) | ((x) >> (32 - (n))))

Functions

- void shaEngine (U32 *buf, U32 *h)
- void CL_hashInit (CL_HashContext *ctx)

  Initialize context for performing SHA1 hash in software.
- void CL_hashUpdate (CL_HashContext *ctx, const U8 *src, int nbytes)

  Add arbitrary data to a SHA1 hash.
- void CL_hashFinal (CL_HashContext *ctx, U8 *dest)

  Complete the SHA1 hash in software and return the digest.
- void CL_hash (U8 *msg, int msgBytes, U8 *dest)

  Perform SHA1 hash of data in software.

15.159.1 Detailed Description

Software implementation of the SHA1 algorithm.

Copyright

15.159.2 Macro Definition Documentation

15.159.2.1 _NOP

```
#define _NOP( )
```

15.159.2.2 _WDRESET

```
#define _WDRESET( )
```

15.159.2.3 leftRotate

```
#define leftRotate(
              x,
              n ) (x) = (((x) << (n)) | ((x) >> (32 - (n))))
```

15.159.2.4 memcpy_P

```
#define memcpy_P memmove
```

15.159.2.5 strcpy_P

```
#define strcpy_P strcpy
```

15.159.2.6 U16

```
#define U16 uint16_t
```

15.159.2.7 U32

```
#define U32 uint32_t
```

15.159.2.8 U8

```
#define U8 uint8_t
```

15.159.3 Function Documentation

15.159.3.1 CL_hash()

```
void CL_hash (
            U8 * msg,
            int msgBytes,
            U8 * dest )
```

Perform SHA1 hash of data in software.

Parameters

in	*msg*	Data to be hashed
in	*msgBytes*	Data size in bytes
out	*dest*	Digest is returned here (20 bytes)

15.159.3.2 CL_hashFinal()

```
void CL_hashFinal (
            CL_HashContext * ctx,
            U8 * dest )
```

Complete the SHA1 hash in software and return the digest.

Parameters

in	*ctx*	Hash context
out	*dest*	Digest is returned here (20 bytes)

15.159.3.3 CL_hashInit()

```
void CL_hashInit (
            CL_HashContext * ctx )
```

Initialize context for performing SHA1 hash in software.

Parameters

in	*ctx*	Hash context

15.159.3.4 CL_hashUpdate()

```
void CL_hashUpdate (
            CL_HashContext * ctx,
            const U8 * src,
            int nbytes )
```

Add arbitrary data to a SHA1 hash.

Parameters

in	*ctx*	Hash context
in	*src*	Data to be added to the hash
in	*nbytes*	Data size in bytes

15.159.3.5 shaEngine()

```
void shaEngine (
            U32 * buf,
            U32 * h )
```

15.160 sha2_routines.c File Reference

Software implementation of the SHA256 algorithm.

```
#include <string.h>
#include "sha2_routines.h"
#include "atca_compiler.h"
```

Macros

- #define rotate_right(value, places) ((value >> places) | (value << (32 - places)))

Functions

- void sw_sha256_init (sw_sha256_ctx *ctx)

  Intialize the software SHA256.
- void sw_sha256_update (sw_sha256_ctx *ctx, const uint8_t *msg, uint32_t msg_size)

  updates the running hash with the next block of data, called iteratively for the entire stream of data to be hashed using the SHA256 software
- void sw_sha256_final (sw_sha256_ctx *ctx, uint8_t digest[SHA256_DIGEST_SIZE])

  completes the final SHA256 calculation and returns the final digest/hash
- void sw_sha256 (const uint8_t *message, unsigned int len, uint8_t digest[SHA256_DIGEST_SIZE])

  single call convenience function which computes Hash of given data using SHA256 software

15.160.1 Detailed Description

Software implementation of the SHA256 algorithm.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.160.2 Macro Definition Documentation

15.160.2.1 rotate_right

```
#define rotate_right(
              value,
              places ) ((value >> places) | (value << (32 - places)))
```

15.160.3 Function Documentation

15.160.3.1 sw_sha256()

```
void sw_sha256 (
             const uint8_t * message,
             unsigned int len,
             uint8_t digest[SHA256_DIGEST_SIZE] )
```

single call convenience function which computes Hash of given data using SHA256 software

Parameters

in	*message*	pointer to stream of data to hash
in	*len*	size of data stream to hash
out	*digest*	result

15.160.3.2 sw_sha256_final()

```
void sw_sha256_final (
            sw_sha256_ctx * ctx,
            uint8_t digest[SHA256_DIGEST_SIZE] )
```

completes the final SHA256 calculation and returns the final digest/hash

Parameters

in	*ctx*	ptr to context data structure
out	*digest*	receives the computed digest of the SHA 256

15.160.3.3 sw_sha256_init()

```
void sw_sha256_init (
            sw_sha256_ctx * ctx )
```

Intialize the software SHA256.

Parameters

in	*ctx*	SHA256 hash context

15.160.3.4 sw_sha256_update()

```
void sw_sha256_update (
            sw_sha256_ctx * ctx,
            const uint8_t * msg,
            uint32_t msg_size )
```

updates the running hash with the next block of data, called iteratively for the entire stream of data to be hashed using the SHA256 software

Parameters

in	*ctx*	SHA256 hash context
in	*msg*	Raw blocks to be processed
in	*msg_size*	The size of the message passed

15.161 sha2_routines.h File Reference

Software implementation of the SHA256 algorithm.

```
#include <stdint.h>
```

Data Structures

- struct sw_sha256_ctx

Macros

- #define SHA256_DIGEST_SIZE (32)
- #define SHA256_BLOCK_SIZE (64)

Functions

- void sw_sha256_init (sw_sha256_ctx ∗ctx)

  Intialize the software SHA256.
- void sw_sha256_update (sw_sha256_ctx ∗ctx, const uint8_t ∗message, uint32_t len)

  updates the running hash with the next block of data, called iteratively for the entire stream of data to be hashed using the SHA256 software
- void sw_sha256_final (sw_sha256_ctx ∗ctx, uint8_t digest[SHA256_DIGEST_SIZE])

  completes the final SHA256 calculation and returns the final digest/hash
- void sw_sha256 (const uint8_t ∗message, unsigned int len, uint8_t digest[SHA256_DIGEST_SIZE])

  single call convenience function which computes Hash of given data using SHA256 software

15.161.1 Detailed Description

Software implementation of the SHA256 algorithm.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.161.2 Macro Definition Documentation

15.161.2.1 SHA256_BLOCK_SIZE

```
#define SHA256_BLOCK_SIZE (64)
```

15.161.2.2 SHA256_DIGEST_SIZE

```
#define SHA256_DIGEST_SIZE (32)
```

15.161.3 Function Documentation

15.161.3.1 sw_sha256()

```
void sw_sha256 (
            const uint8_t * message,
            unsigned int len,
            uint8_t digest[SHA256_DIGEST_SIZE] )
```

single call convenience function which computes Hash of given data using SHA256 software

Parameters

in	*message*	pointer to stream of data to hash
in	*len*	size of data stream to hash
out	*digest*	result

15.161.3.2 sw_sha256_final()

```
void sw_sha256_final (
            sw_sha256_ctx * ctx,
            uint8_t digest[SHA256_DIGEST_SIZE] )
```

completes the final SHA256 calculation and returns the final digest/hash

Parameters

in	*ctx*	ptr to context data structure
out	*digest*	receives the computed digest of the SHA 256

15.161.3.3 sw_sha256_init()

```
void sw_sha256_init (
            sw_sha256_ctx * ctx )
```

Intialize the software SHA256.

Parameters

in	*ctx*	SHA256 hash context

15.161.3.4 sw_sha256_update()

```
void sw_sha256_update (
            sw_sha256_ctx * ctx,
            const uint8_t * msg,
            uint32_t msg_size )
```

updates the running hash with the next block of data, called iteratively for the entire stream of data to be hashed using the SHA256 software

Parameters

in	*ctx*	SHA256 hash context
in	*msg*	Raw blocks to be processed
in	*msg_size*	The size of the message passed

15.162 swi_bitbang_samd21.c File Reference

Hardware Interface Functions - SWI bit-banged.

```
#include <asf.h>
#include <stdint.h>
#include "swi_bitbang_samd21.h"
#include "atca_command.h"
```

Functions

- void swi_set_pin (uint8_t id)

  Set SWI signal pin. Other functions will use this pin.
- void swi_enable (void)

  Configure GPIO pin for SWI signal as output.
- void swi_disable (void)

  Configure GPIO pin for SWI signal as input.
- void swi_set_signal_pin (uint8_t is_high)

  Set signal pin Low or High.
- void swi_send_wake_token (void)

  Send a Wake Token.
- void swi_send_bytes (uint8_t count, uint8_t ∗buffer)

  Send a number of bytes.This function should not be called directly ,instead should use hal_swi_send() which call this function.

- void swi_send_byte (uint8_t byte)

  Send one byte.
- ATCA_STATUS swi_receive_bytes (uint8_t count, uint8_t ∗buffer)

  Receive a number of bytes.This function should not be called directly ,instead should use hal_swi_receive() which call this function.

Variables

- SWIBuses swi_buses_default

15.162.1 Detailed Description

Hardware Interface Functions - SWI bit-banged.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.162.2 Function Documentation

15.162.2.1 swi_disable()

```
void swi_disable (
            void  )
```

Configure GPIO pin for SWI signal as input.

15.162.2.2 swi_enable()

```
void swi_enable (
            void  )
```

Configure GPIO pin for SWI signal as output.

15.162.2.3 swi_receive_bytes()

```
ATCA_STATUS swi_receive_bytes (
            uint8_t count,
            uint8_t * buffer )
```

Receive a number of bytes.This function should not be called directly ,instead should use hal_swi_receive() which call this function.

Parameters

in	*count*	number of bytes to receive
out	*buffer*	pointer to receive buffer

Returns

ATCA_SUCCESS on success, otherwise an error code.

Receive bits and store in buffer.

Detect start bit.

Wait for falling edge.

Wait for rising edge.

let's just wait the maximum time for the falling edge of a zero bit to arrive after we have detected the rising edge of the start bit.

Detect possible edge indicating zero bit.

Wait for rising edge of zero pulse before returning. Otherwise we might interpret its rising edge as the next start pulse.

Update byte at current buffer index.

received "one" bit

Indicate that we timed out after having received at least one byte.

15.162.2.4 swi_send_byte()

```
void swi_send_byte (
            uint8_t byte )
```

Send one byte.

Parameters

in	*byte*	byte to send

15.162.2.5 swi_send_bytes()

```
void swi_send_bytes (
            uint8_t count,
            uint8_t * buffer )
```

Send a number of bytes.This function should not be called directly ,instead should use hal_swi_send() which call this function.

Parameters

in	*count*	number of bytes to send.
in	*buffer*	pointer to buffer containing bytes to send

< Send Logic 1 (7F)

< Send Logic 0 (7D)

15.162.2.6 swi_send_wake_token()

```
void swi_send_wake_token (
            void  )
```

Send a Wake Token.

15.162.2.7 swi_set_pin()

```
void swi_set_pin (
            uint8_t id )
```

Set SWI signal pin. Other functions will use this pin.

Parameters

in	*id*	definition of GPIO pin to be used

15.162.2.8 swi_set_signal_pin()

```
void swi_set_signal_pin (
            uint8_t is_high )
```

Set signal pin Low or High.

Parameters

in	*is_high*	0: Low, else: High.

15.162.3 Variable Documentation

15.162.3.1 swi_buses_default

```
SWIBuses swi_buses_default
```

Initial value:
```
= {
    { EXT3_PIN_3, EXT3_PIN_9, EXT3_PIN_I2C_SDA, EXT3_PIN_13, EXT2_PIN_13, EXT2_PIN_5, EXT2_PIN_7,
      EXT2_PIN_9, EXT2_PIN_3, EXT2_PIN_15, EXT2_PIN_17, EXT1_PIN_3, EXT1_PIN_5, EXT1_PIN_7, EXT1_PIN_9,
      EXT1_PIN_13, EXT1_PIN_15, EXT1_PIN_17, EXT3_PIN_7, EXT3_PIN_10, EXT3_PIN_I2C_SCL, EXT3_PIN_14,
      EXT2_PIN_4, EXT2_PIN_6, EXT2_PIN_8, EXT2_PIN_10, EXT2_PIN_14, EXT2_PIN_16, EXT2_PIN_18, EXT1_PIN_4,
      EXT1_PIN_6, EXT1_PIN_8, EXT1_PIN_10, EXT1_PIN_14, EXT1_PIN_16, EXT1_PIN_18 }
}
```

15.163 swi_bitbang_samd21.h File Reference

Hardware Interface Functions - SWI bit-banged.

```
#include "atca_status.h"
#include <delay.h>
```

Data Structures

- struct SWIBuses

Macros

- #define MAX_SWI_BUSES 36

  SAMD21 xplainned pro has 36 free GPIO pins available.

Macros for Bit-Banged SWI Timing

Times to drive bits at 230.4 kbps.

- #define BIT_DELAY_1L delay_us(3)
- #define BIT_DELAY_1H delay_us(3)

  should be 4.34 us, is 4.05us
- #define BIT_DELAY_5 delay_us(26)
- #define BIT_DELAY_7 delay_us(34)
- #define RX_TX_DELAY delay_us(65)
- #define START_PULSE_TIME_OUT (600)
- #define ZERO_PULSE_TIME_OUT (40)

Functions

- void swi_set_pin (uint8_t id)

  Set SWI signal pin. Other functions will use this pin.
- void swi_enable (void)

  Configure GPIO pin for SWI signal as output.
- void swi_disable (void)

  Configure GPIO pin for SWI signal as input.
- void swi_set_signal_pin (uint8_t is_high)

  Set signal pin Low or High.
- void swi_send_wake_token (void)

  Send a Wake Token.
- void swi_send_bytes (uint8_t count, uint8_t ∗buffer)

  Send a number of bytes.This function should not be called directly ,instead should use hal_swi_send() which call this function.
- void swi_send_byte (uint8_t byte)

  Send one byte.
- ATCA_STATUS swi_receive_bytes (uint8_t count, uint8_t ∗buffer)

  Receive a number of bytes.This function should not be called directly ,instead should use hal_swi_receive() which call this function.

Variables

- SWIBuses swi_buses_default

15.163.1 Detailed Description

Hardware Interface Functions - SWI bit-banged.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.163.2 Macro Definition Documentation

15.163.2.1 BIT_DELAY_1H

```
#define BIT_DELAY_1H delay_us(3)
```

should be 4.34 us, is 4.05us

15.163.2.2 BIT_DELAY_1L

```
#define BIT_DELAY_1L delay_us(3)
```

delay macro for width of one pulse (start pulse or zero pulse) should be 4.34 us, is 4.05 us

15.163.2.3 BIT_DELAY_5

```
#define BIT_DELAY_5 delay_us(26)
```

time to keep pin high for five pulses plus stop bit (used to bit-bang CryptoAuth 'zero' bit) should be 26.04 us, is 26.92 us

15.163.2.4 BIT_DELAY_7

```
#define BIT_DELAY_7 delay_us(34)
```

time to keep pin high for seven bits plus stop bit (used to bit-bang CryptoAuth 'one' bit) should be 34.72 us, is 35.13 us

15.163.2.5 MAX_SWI_BUSES

```
#define MAX_SWI_BUSES 36
```

SAMD21 xplainned pro has 36 free GPIO pins available.

15.163.2.6 RX_TX_DELAY

```
#define RX_TX_DELAY delay_us(65)
```

turn around time when switching from receive to transmit should be 93 us (Setting little less value as there would be other process before these steps)

15.163.2.7 START_PULSE_TIME_OUT

```
#define START_PULSE_TIME_OUT (600)
```

Lets set the timeout value for start pulse detection to the uint8_t maximum. This value is decremented while waiting for the falling edge of a start pulse.

15.163.2.8 ZERO_PULSE_TIME_OUT

```
#define ZERO_PULSE_TIME_OUT (40)
```

Maximum time between rising edge of start pulse and falling edge of zero pulse is 8.6 us. Therefore, a value of 40 (around 15 us) gives ample time to detect a zero pulse and also leaves enough time to detect the following start pulse. This value is decremented while waiting for the falling edge of a zero pulse.

15.163.3 Function Documentation

15.163.3.1 swi_disable()

```
void swi_disable (
            void )
```

Configure GPIO pin for SWI signal as input.

15.163.3.2 swi_enable()

```
void swi_enable (
            void )
```

Configure GPIO pin for SWI signal as output.

15.163.3.3 swi_receive_bytes()

```
ATCA_STATUS swi_receive_bytes (
            uint8_t count,
            uint8_t * buffer )
```

Receive a number of bytes.This function should not be called directly ,instead should use hal_swi_receive() which call this function.

Parameters

in	*count*	number of bytes to receive
out	*buffer*	pointer to receive buffer

Returns

ATCA_SUCCESS on success, otherwise an error code.

Receive bits and store in buffer.

Detect start bit.

Wait for falling edge.

Wait for rising edge.

let's just wait the maximum time for the falling edge of a zero bit to arrive after we have detected the rising edge of the start bit.

Detect possible edge indicating zero bit.

Wait for rising edge of zero pulse before returning. Otherwise we might interpret its rising edge as the next start pulse.

Update byte at current buffer index.

received "one" bit

Indicate that we timed out after having received at least one byte.

15.163.3.4 swi_send_byte()

```
void swi_send_byte (
            uint8_t byte )
```

Send one byte.

Parameters

in	*byte*	byte to send

15.163.3.5 swi_send_bytes()

```
void swi_send_bytes (
            uint8_t count,
            uint8_t * buffer )
```

Send a number of bytes.This function should not be called directly ,instead should use hal_swi_send() which call this function.

Parameters

in	*count*	number of bytes to send.
in	*buffer*	pointer to buffer containing bytes to send

< Send Logic 1 (7F)

< Send Logic 0 (7D)

15.163.3.6 swi_send_wake_token()

```
void swi_send_wake_token (
            void )
```

Send a Wake Token.

15.163.3.7 swi_set_pin()

```
void swi_set_pin (
            uint8_t id )
```

Set SWI signal pin. Other functions will use this pin.

Parameters

in	*id*	definition of GPIO pin to be used

15.163.3.8 swi_set_signal_pin()

```
void swi_set_signal_pin (
            uint8_t is_high )
```

Set signal pin Low or High.

Parameters

in	*is_high*	0: Low, else: High.

15.163.4 Variable Documentation

15.163.4.1 swi_buses_default

```
SWIBuses swi_buses_default
```

15.164 swi_uart_at90usb1287_asf.c File Reference

ATXMEGA's ATCA Hardware abstraction layer for SWI interface over AT90USB1287 UART drivers.

```
#include <stdlib.h>
#include <stdio.h>
#include "usart_serial.h"
#include "swi_uart_at90usb1287_asf.h"
#include "basic/atca_helpers.h"
```

Functions

- ATCA_STATUS swi_uart_init (ATCASWIMaster_t ∗instance)

  Implementation of SWI UART init.
- ATCA_STATUS swi_uart_deinit (ATCASWIMaster_t ∗instance)

  Implementation of SWI UART deinit.
- void swi_uart_setbaud (ATCASWIMaster_t ∗instance, uint32_t baudrate)

  implementation of SWI UART change baudrate.
- void swi_uart_mode (ATCASWIMaster_t ∗instance, uint8_t mode)

  implementation of SWI UART change mode.
- void swi_uart_discover_buses (int swi_uart_buses[ ], int max_buses)

  discover UART buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge
- ATCA_STATUS swi_uart_send_byte (ATCASWIMaster_t ∗instance, uint8_t data)

  HAL implementation of SWI UART send byte over ASF. This function send one byte over UART.
- ATCA_STATUS swi_uart_receive_byte (ATCASWIMaster_t ∗instance, uint8_t ∗data)

  HAL implementation of SWI UART receive bytes over ASF. This function receive one byte over UART.

15.164.1 Detailed Description

ATXMEGA's ATCA Hardware abstraction layer for SWI interface over AT90USB1287 UART drivers.

Prerequisite: add UART Polled support to application in Atmel Studio

Copyright

15.165 swi_uart_at90usb1287_asf.h File Reference

ATMEGA's ATCA Hardware abstraction layer for SWI interface over AT90USB1287 UART drivers.

```
#include <asf.h>
#include "cryptoauthlib.h"
#include "serial.h"
```

Data Structures

- struct atcaSWImaster

  This is the hal_data for ATCA HAL.

Macros

- #define MAX_SWI_BUSES 1
- #define RECEIVE_MODE 0
- #define TRANSMIT_MODE 1
- #define RX_DELAY 10
- #define TX_DELAY 90

Typedefs

- typedef struct atcaSWImaster ATCASWIMaster_t

    this is the hal_data for ATCA HAL for SWI UART

Functions

- ATCA_STATUS swi_uart_init (ATCASWIMaster_t ∗instance)

    Implementation of SWI UART init.
- ATCA_STATUS swi_uart_deinit (ATCASWIMaster_t ∗instance)

    Implementation of SWI UART deinit.
- void swi_uart_setbaud (ATCASWIMaster_t ∗instance, uint32_t baudrate)

    implementation of SWI UART change baudrate.
- void swi_uart_mode (ATCASWIMaster_t ∗instance, uint8_t mode)

    implementation of SWI UART change mode.
- void swi_uart_discover_buses (int swi_uart_buses[ ], int max_buses)

    discover UART buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge
- ATCA_STATUS swi_uart_send_byte (ATCASWIMaster_t ∗instance, uint8_t data)

    HAL implementation of SWI UART send byte over ASF. This function send one byte over UART.
- ATCA_STATUS swi_uart_receive_byte (ATCASWIMaster_t ∗instance, uint8_t ∗data)

    HAL implementation of SWI UART receive bytes over ASF. This function receive one byte over UART.

15.165.1 Detailed Description

ATMEGA's ATCA Hardware abstraction layer for SWI interface over AT90USB1287 UART drivers.

Prerequisite: add UART Polled support to application in Atmel Studio

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.166 swi_uart_samd21_asf.c File Reference

ATXMEGA's ATCA Hardware abstraction layer for SWI interface over UART drivers.

```
#include <stdlib.h>
#include <stdio.h>
#include "swi_uart_samd21_asf.h"
#include "basic/atca_helpers.h"
```

Functions

- ATCA_STATUS swi_uart_init (ATCASWIMaster_t ∗instance)

  Implementation of SWI UART init.
- ATCA_STATUS swi_uart_deinit (ATCASWIMaster_t ∗instance)

  Implementation of SWI UART deinit.
- void swi_uart_setbaud (ATCASWIMaster_t ∗instance, uint32_t baudrate)

  implementation of SWI UART change baudrate.
- void swi_uart_mode (ATCASWIMaster_t ∗instance, uint8_t mode)

  implementation of SWI UART change mode.
- void swi_uart_discover_buses (int swi_uart_buses[ ], int max_buses)

  discover UART buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge
- ATCA_STATUS swi_uart_send_byte (ATCASWIMaster_t ∗instance, uint8_t data)

  HAL implementation of SWI UART send byte over ASF. This function send one byte over UART.
- ATCA_STATUS swi_uart_receive_byte (ATCASWIMaster_t ∗instance, uint8_t ∗data)

  HAL implementation of SWI UART receive bytes over ASF. This function receive one byte over UART.

Variables

- struct port_config pin_conf

15.166.1 Detailed Description

ATXMEGA's ATCA Hardware abstraction layer for SWI interface over UART drivers.

Prerequisite: add UART Polled support to application in Atmel Studio

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.167 swi_uart_samd21_asf.h File Reference

ATXMEGA's ATCA Hardware abstraction layer for SWI interface over UART drivers.

```
#include <asf.h>
#include "cryptoauthlib.h"
```

Data Structures

- struct atcaSWImaster

  This is the hal_data for ATCA HAL.

Macros

- #define MAX_SWI_BUSES 6
- #define RECEIVE_MODE 0
- #define TRANSMIT_MODE 1
- #define RX_DELAY 10
- #define TX_DELAY 90
- #define DEBUG_PIN_1 EXT2_PIN_5
- #define DEBUG_PIN_2 EXT2_PIN_6

Typedefs

- typedef struct atcaSWImaster ATCASWIMaster_t

  this is the hal_data for ATCA HAL for ASF SERCOM

Functions

- ATCA_STATUS swi_uart_init (ATCASWIMaster_t ∗instance)

  Implementation of SWI UART init.
- ATCA_STATUS swi_uart_deinit (ATCASWIMaster_t ∗instance)

  Implementation of SWI UART deinit.
- void swi_uart_setbaud (ATCASWIMaster_t ∗instance, uint32_t baudrate)

  implementation of SWI UART change baudrate.
- void swi_uart_mode (ATCASWIMaster_t ∗instance, uint8_t mode)

  implementation of SWI UART change mode.
- void swi_uart_discover_buses (int swi_uart_buses[ ], int max_buses)

  discover UART buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge
- ATCA_STATUS swi_uart_send_byte (ATCASWIMaster_t ∗instance, uint8_t data)

  HAL implementation of SWI UART send byte over ASF. This function send one byte over UART.
- ATCA_STATUS swi_uart_receive_byte (ATCASWIMaster_t ∗instance, uint8_t ∗data)

  HAL implementation of SWI UART receive bytes over ASF. This function receive one byte over UART.

15.167.1 Detailed Description

ATXMEGA's ATCA Hardware abstraction layer for SWI interface over UART drivers.

Prerequisite: add UART Polled support to application in Atmel Studio

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.168 swi_uart_start.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <peripheral_clk_config.h>
#include "swi_uart_start.h"
#include "basic/atca_helpers.h"
```

Macros

- #define USART_BAUD_RATE(baud, sercom_freq) (65536 - ((65536 ∗ 16.0F ∗ baud) / sercom_freq))

Functions

- ATCA_STATUS swi_uart_init (ATCASWIMaster_t ∗instance)

  Implementation of SWI UART init.
- ATCA_STATUS swi_uart_deinit (ATCASWIMaster_t ∗instance)

  Implementation of SWI UART deinit.
- void swi_uart_setbaud (ATCASWIMaster_t ∗instance, uint32_t baudrate)

  implementation of SWI UART change baudrate.
- void swi_uart_mode (ATCASWIMaster_t ∗instance, uint8_t mode)

  implementation of SWI UART change mode.
- void swi_uart_discover_buses (int swi_uart_buses[ ], int max_buses)

  discover UART buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge
- ATCA_STATUS swi_uart_send_byte (ATCASWIMaster_t ∗instance, uint8_t data)

  HAL implementation of SWI UART send byte over ASF. This function send one byte over UART.
- ATCA_STATUS swi_uart_receive_byte (ATCASWIMaster_t ∗instance, uint8_t ∗data)

  HAL implementation of SWI UART receive bytes over ASF. This function receive one byte over UART.

15.168.1 Detailed Description

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.168.2 Macro Definition Documentation

15.168.2.1 USART_BAUD_RATE

```
#define USART_BAUD_RATE(
            baud,
            sercom_freq ) (65536 - ((65536 * 16.0F * baud) / sercom_freq))
```

15.169 swi_uart_start.h File Reference

```
#include <stdlib.h>
#include "atmel_start.h"
#include "cryptoauthlib.h"
```

Data Structures

- struct atcaSWImaster

    This is the hal_data for ATCA HAL.

Macros

- #define MAX_SWI_BUSES 6
- #define RECEIVE_MODE 0
- #define TRANSMIT_MODE 1
- #define RX_DELAY 10
- #define TX_DELAY 93

Typedefs

- typedef struct atcaSWImaster ATCASWIMaster_t

    this is the hal_data for ATCA HAL for ASF SERCOM

Functions

- ATCA_STATUS swi_uart_init (ATCASWIMaster_t ∗instance)

    Implementation of SWI UART init.
- ATCA_STATUS swi_uart_deinit (ATCASWIMaster_t ∗instance)

    Implementation of SWI UART deinit.
- void swi_uart_setbaud (ATCASWIMaster_t ∗instance, uint32_t baudrate)

    implementation of SWI UART change baudrate.
- void swi_uart_mode (ATCASWIMaster_t ∗instance, uint8_t mode)

    implementation of SWI UART change mode.
- void swi_uart_discover_buses (int swi_uart_buses[ ], int max_buses)

    discover UART buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge
- ATCA_STATUS swi_uart_send_byte (ATCASWIMaster_t ∗instance, uint8_t data)

    HAL implementation of SWI UART send byte over ASF. This function send one byte over UART.
- ATCA_STATUS swi_uart_receive_byte (ATCASWIMaster_t ∗instance, uint8_t ∗data)

    HAL implementation of SWI UART receive bytes over ASF. This function receive one byte over UART.

15.169.1 Detailed Description

Copyright

15.170 swi_uart_xmega_a3bu_asf.c File Reference

ATXMEGA's ATCA Hardware abstraction layer for SWI interface over XMEGA UART drivers.

```
#include <stdlib.h>
#include <stdio.h>
#include "swi_uart_xmega_a3bu_asf.h"
#include "basic/atca_helpers.h"
```

Macros

- #define DEBUG_PIN 1
- #define DEBUG_PIN_1 IOPORT_CREATE_PIN(PORTB, 0)
- #define DEBUG_PIN_2 IOPORT_CREATE_PIN(PORTB, 1)

Functions

- ATCA_STATUS swi_uart_init (ATCASWIMaster_t ∗instance)

  Implementation of SWI UART init.
- ATCA_STATUS swi_uart_deinit (ATCASWIMaster_t ∗instance)

  Implementation of SWI UART deinit.
- void swi_uart_setbaud (ATCASWIMaster_t ∗instance, uint32_t baudrate)

  implementation of SWI UART change baudrate.
- void swi_uart_mode (ATCASWIMaster_t ∗instance, uint8_t mode)

  implementation of SWI UART change mode.
- void swi_uart_discover_buses (int swi_uart_buses[ ], int max_buses)

  discover UART buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge
- ATCA_STATUS swi_uart_send_byte (ATCASWIMaster_t ∗instance, uint8_t data)

  HAL implementation of SWI UART send byte over ASF. This function send one byte over UART.
- ATCA_STATUS swi_uart_receive_byte (ATCASWIMaster_t ∗instance, uint8_t ∗data)

  HAL implementation of SWI UART receive bytes over ASF. This function receive one byte over UART.

15.170.1 Detailed Description

ATXMEGA's ATCA Hardware abstraction layer for SWI interface over XMEGA UART drivers.

Prerequisite: add UART Polled support to application in Atmel Studio

Copyright

15.170.2 Macro Definition Documentation

15.170.2.1 DEBUG_PIN

```
#define DEBUG_PIN 1
```

15.170.2.2 DEBUG_PIN_1

```
#define DEBUG_PIN_1 IOPORT_CREATE_PIN(PORTB, 0)
```

15.170.2.3 DEBUG_PIN_2

```
#define DEBUG_PIN_2 IOPORT_CREATE_PIN(PORTB, 1)
```

15.171 swi_uart_xmega_a3bu_asf.h File Reference

ATXMEGA's ATCA Hardware abstraction layer for SWI interface over XMEGA UART drivers.

```
#include <asf.h>
#include "cryptoauthlib.h"
#include "serial.h"
```

Data Structures

- struct atcaSWImaster

  This is the hal_data for ATCA HAL.

Macros

- #define MAX_SWI_BUSES 6
- #define RECEIVE_MODE 0
- #define TRANSMIT_MODE 1
- #define RX_DELAY 10
- #define TX_DELAY 90

Typedefs

- typedef struct atcaSWImaster ATCASWIMaster_t

  this is the hal_data for ATCA HAL for SWI UART

Functions

- ATCA_STATUS swi_uart_init (ATCASWIMaster_t ∗instance)

  Implementation of SWI UART init.
- ATCA_STATUS swi_uart_deinit (ATCASWIMaster_t ∗instance)

  Implementation of SWI UART deinit.
- void swi_uart_setbaud (ATCASWIMaster_t ∗instance, uint32_t baudrate)

  implementation of SWI UART change baudrate.
- void swi_uart_mode (ATCASWIMaster_t ∗instance, uint8_t mode)

  implementation of SWI UART change mode.
- void swi_uart_discover_buses (int swi_uart_buses[ ], int max_buses)

  discover UART buses available for this hardware this maintains a list of logical to physical bus mappings freeing the application of the a-priori knowledge
- ATCA_STATUS swi_uart_send_byte (ATCASWIMaster_t ∗instance, uint8_t data)

  HAL implementation of SWI UART send byte over ASF. This function send one byte over UART.
- ATCA_STATUS swi_uart_receive_byte (ATCASWIMaster_t ∗instance, uint8_t ∗data)

  HAL implementation of SWI UART receive bytes over ASF. This function receive one byte over UART.

15.171.1 Detailed Description

ATXMEGA's ATCA Hardware abstraction layer for SWI interface over XMEGA UART drivers.

Prerequisite: add UART Polled support to application in Atmel Studio

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.172 symmetric_authentication.c File Reference

Contains API for performing the symmetric Authentication between the Host and the device.

```
#include "cryptoauthlib.h"
#include "host/atca_host.h"
#include "symmetric_authentication.h"
```

Functions

- ATCA_STATUS symmetric_authenticate (uint8_t slot, const uint8_t *master_key, const uint8_t *rand_number)

  Function which does the authentication between the host and device.

15.172.1 Detailed Description

Contains API for performing the symmetric Authentication between the Host and the device.

Copyright

(c) 2015-2018 Microchip Technology Inc. and its subsidiaries.

15.172.2 Function Documentation

15.172.2.1 symmetric_authenticate()

```
ATCA_STATUS symmetric_authenticate (
            uint8_t slot,
            const uint8_t * master_key,
            const uint8_t * rand_number )
```

Function which does the authentication between the host and device.

Parameters

in	*slot*	The slot number used for the symmetric authentication.
in	*master_key*	The master key used for the calculating the symmetric key.
in	*rand_number*	The 20 byte rand_number from the host.

Returns

ATCA_SUCCESS on successful authentication, otherwise an error code.

15.173 symmetric_authentication.h File Reference

Contains API for performing the symmetric Authentication between the Host and the device.

```
#include "cryptoauthlib.h"
```

Functions

- ATCA_STATUS symmetric_authenticate (uint8_t slot, const uint8_t *master_key, const uint8_t *rand_↩ number)

  Function which does the authentication between the host and device.

15.173.1 Detailed Description

Contains API for performing the symmetric Authentication between the Host and the device.

Copyright

15.173.2 Function Documentation

15.173.2.1 symmetric_authenticate()

```
ATCA_STATUS symmetric_authenticate (
            uint8_t slot,
            const uint8_t * master_key,
            const uint8_t * rand_number )
```

Function which does the authentication between the host and device.

Parameters

in	*slot*	The slot number used for the symmetric authentication.
in	*master_key*	The master key used for the calculating the symmetric key.
in	*rand_number*	The 20 byte rand_number from the host.

Returns

ATCA_SUCCESS on successful authentication, otherwise an error code.

15.174 tng22_cert_def_1_signer.c File Reference

TNG 22 signer certificate definition.

```
#include "atcacert/atcacert_def.h"
#include "tng22_cert_def_1_signer.h"
```

Variables

- const uint8_t g_tng22_cert_template_1_signer [TNG22_CERT_TEMPLATE_1_SIGNER_SIZE]
- const atcacert_def_t g_tng22_cert_def_1_signer

15.174.1 Detailed Description

TNG 22 signer certificate definition.

Copyright

15.174.2 Variable Documentation

15.174.2.1 g_tng22_cert_def_1_signer

```
const atcacert_def_t g_tng22_cert_def_1_signer
```

15.174.2.2 g_tng22_cert_template_1_signer

```
const uint8_t g_tng22_cert_template_1_signer[TNG22_CERT_TEMPLATE_1_SIGNER_SIZE]
```

15.175 tng22_cert_def_1_signer.h File Reference

TNG 22 signer certificate definition.

```
#include "atcacert/atcacert_def.h"
```

- #define TNG22_CERT_TEMPLATE_1_SIGNER_SIZE 520
- const atcacert_def_t g_tng22_cert_def_1_signer

15.175.1 Detailed Description

TNG 22 signer certificate definition.

Copyright

(c) 2015-2019 Microchip Technology Inc. and its subsidiaries.

15.176 tng22_cert_def_2_device.c File Reference

TNG 22 device certificate definition.

```
#include "atcacert/atcacert_def.h"
#include "tng22_cert_def_2_device.h"
#include "tng22_cert_def_1_signer.h"
```

Variables

- const uint8_t g_tng22_cert_template_2_device [TNG22_CERT_TEMPLATE_2_DEVICE_SIZE]
- const atcacert_cert_element_t g_tng22_cert_elements_2_device [TNG22_CERT_ELEMENTS_2_DEVICE_COUNT]
- const atcacert_def_t g_tng22_cert_def_2_device

15.176.1 Detailed Description

TNG 22 device certificate definition.

Copyright

(c) 2015-2019 Microchip Technology Inc. and its subsidiaries.

15.176.2 Variable Documentation

15.176.2.1 g_tng22_cert_def_2_device

```
const atcacert_def_t g_tng22_cert_def_2_device
```

15.176.2.2 g_tng22_cert_elements_2_device

```
const atcacert_cert_element_t g_tng22_cert_elements_2_device[TNG22_CERT_ELEMENTS_2_DEVICE_COUNT]
```

15.176.2.3 g_tng22_cert_template_2_device

```
const uint8_t g_tng22_cert_template_2_device[TNG22_CERT_TEMPLATE_2_DEVICE_SIZE]
```

15.177 tng22_cert_def_2_device.h File Reference

TNG 22 device certificate definition.

```
#include "atcacert/atcacert_def.h"
```

- #define TNG22_CERT_TEMPLATE_2_DEVICE_SIZE 505
- #define TNG22_CERT_ELEMENTS_2_DEVICE_COUNT 2
- const atcacert_def_t g_tng22_cert_def_2_device

15.177.1 Detailed Description

TNG 22 device certificate definition.

Copyright

(c) 2015-2019 Microchip Technology Inc. and its subsidiaries.

15.178 tng_atca.c File Reference

```
#include "tng_atca.h"
```

Functions

- ATCA_STATUS tng_get_type (tng_type_t ∗type)

  Get the type of TNG device.
- ATCA_STATUS tng_get_device_pubkey (uint8_t ∗public_key)

  Uses GenKey command to calculate the public key from the primary device public key.

15.179 tng_atca.h File Reference

```
#include "basic/atca_basic.h"
```

Macros

- #define TNG22_PRIMARY_KEY_SLOT 0
- #define TNGTN_PRIMARY_KEY_SLOT 1

Enumerations

- enum tng_type_t { TNGTYPE_UNKNOWN, TNGTYPE_22, TNGTYPE_TN }

Functions

- ATCA_STATUS tng_get_type (tng_type_t ∗type)

  Get the type of TNG device.
- ATCA_STATUS tng_get_device_pubkey (uint8_t ∗public_key)

  Uses GenKey command to calculate the public key from the primary device public key.

15.180 tng_atcacert_client.c File Reference

Client side certificate I/O functions for TNG devices.

```
#include "tng_atca.h"
#include "atcacert/atcacert_client.h"
#include "tng_atcacert_client.h"
#include "tng22_cert_def_2_device.h"
#include "tng22_cert_def_1_signer.h"
#include "tngtn_cert_def_2_device.h"
#include "tngtn_cert_def_1_signer.h"
#include "tng_root_cert.h"
```

Functions

- int tng_atcacert_max_device_cert_size (size_t ∗max_cert_size)

  Return the maximum possible certificate size in bytes for a TNG device certificate. Certificate can be variable size, so this gives an appropriate buffer size when reading the certificate.
- int tng_atcacert_read_device_cert (uint8_t ∗cert, size_t ∗cert_size, const uint8_t ∗signer_cert)

  Reads the device certificate for a TNG device.
- int tng_atcacert_device_public_key (uint8_t ∗public_key, uint8_t ∗cert)

  Reads the device public key.
- int tng_atcacert_max_signer_cert_size (size_t ∗max_cert_size)

  Return the maximum possible certificate size in bytes for a TNG signer certificate. Certificate can be variable size, so this gives an appropriate buffer size when reading the certificate.
- int tng_atcacert_read_signer_cert (uint8_t ∗cert, size_t ∗cert_size)

Reads the signer certificate for a TNG device.
- int tng_atcacert_signer_public_key (uint8_t ∗public_key, uint8_t ∗cert)

    Reads the signer public key.
- int tng_atcacert_root_cert_size (size_t ∗cert_size)

    Get the size of the TNG root cert.
- int tng_atcacert_root_cert (uint8_t ∗cert, size_t ∗cert_size)

    Get the TNG root cert.
- int tng_atcacert_root_public_key (uint8_t ∗public_key)

    Gets the root public key.

15.180.1 Detailed Description

Client side certificate I/O functions for TNG devices.

Copyright

(c) 2015-2019 Microchip Technology Inc. and its subsidiaries.

15.180.2 Function Documentation

15.180.2.1 tng_atcacert_device_public_key()

```
int tng_atcacert_device_public_key (
            uint8_t * public_key,
            uint8_t * cert )
```

Reads the device public key.

Parameters

out	*public_key*	Public key will be returned here. Format will be the X and Y integers in big-endian format. 64 bytes for P256 curve.
in	*cert*	If supplied, the device public key is used from this certificate. If set to NULL, the device public key is read from the device.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

15.180.2.2 tng_atcacert_max_signer_cert_size()

```
int tng_atcacert_max_signer_cert_size (
            size_t * max_cert_size )
```

Return the maximum possible certificate size in bytes for a TNG signer certificate. Certificate can be variable size, so this gives an appropriate buffer size when reading the certificate.

Parameters

out	*max_cert_size*	Maximum certificate size will be returned here in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

15.180.2.3 tng_atcacert_read_device_cert()

```
int tng_atcacert_read_device_cert (
            uint8_t * cert,
            size_t * cert_size,
            const uint8_t * signer_cert )
```

Reads the device certificate for a TNG device.

Parameters

out	*cert*	Buffer to received the certificate (DER format).
in,out	*cert_size*	As input, the size of the cert buffer in bytes. As output, the size of the certificate returned in cert in bytes.
in	*signer_cert*	If supplied, the signer public key is used from this certificate. If set to NULL, the signer public key is read from the device.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

15.180.2.4 tng_atcacert_read_signer_cert()

```
int tng_atcacert_read_signer_cert (
            uint8_t * cert,
            size_t * cert_size )
```

Reads the signer certificate for a TNG device.

Parameters

out	*cert*	Buffer to received the certificate (DER format).
in,out	*cert_size*	As input, the size of the cert buffer in bytes. As output, the size of the certificate returned in cert in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

15.180.2.5 tng_atcacert_root_cert()

```
int tng_atcacert_root_cert (
            uint8_t * cert,
            size_t * cert_size )
```

Get the TNG root cert.

Parameters

out	*cert*	Buffer to received the certificate (DER format).
in,out	*cert_size*	As input, the size of the cert buffer in bytes. As output, the size of the certificate returned in cert in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

15.180.2.6 tng_atcacert_root_cert_size()

```
int tng_atcacert_root_cert_size (
            size_t * cert_size )
```

Get the size of the TNG root cert.

Parameters

out	*cert_size*	Certificate size will be returned here in bytes.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

15.180.2.7 tng_atcacert_root_public_key()

```
int tng_atcacert_root_public_key (
            uint8_t * public_key )
```

Gets the root public key.

Parameters

out	*public_key*	Public key will be returned here. Format will be the X and Y integers in big-endian format. 64 bytes for P256 curve.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

15.180.2.8 tng_atcacert_signer_public_key()

```
int tng_atcacert_signer_public_key (
            uint8_t * public_key,
            uint8_t * cert )
```

Reads the signer public key.

Parameters

out	*public_key*	Public key will be returned here. Format will be the X and Y integers in big-endian format. 64 bytes for P256 curve.
in	*cert*	If supplied, the signer public key is used from this certificate. If set to NULL, the signer public key is read from the device.

Returns

ATCACERT_E_SUCCESS on success, otherwise an error code.

15.181 tng_atcacert_client.h File Reference

Client side certificate I/O functions for TNG devices.

```
#include <stdint.h>
#include "atcacert/atcacert.h"
```

Functions

- int tng_atcacert_max_device_cert_size (size_t *max_cert_size)

  Return the maximum possible certificate size in bytes for a TNG device certificate. Certificate can be variable size, so this gives an appropriate buffer size when reading the certificate.
- int tng_atcacert_read_device_cert (uint8_t *cert, size_t *cert_size, const uint8_t *signer_cert)

  Reads the device certificate for a TNG device.

- int tng_atcacert_device_public_key (uint8_t ∗public_key, uint8_t ∗cert)

  Reads the device public key.
- int tng_atcacert_max_signer_cert_size (size_t ∗max_cert_size)

  Return the maximum possible certificate size in bytes for a TNG signer certificate. Certificate can be variable size, so this gives an appropriate buffer size when reading the certificate.
- int tng_atcacert_read_signer_cert (uint8_t ∗cert, size_t ∗cert_size)

  Reads the signer certificate for a TNG device.
- int tng_atcacert_signer_public_key (uint8_t ∗public_key, uint8_t ∗cert)

  Reads the signer public key.
- int tng_atcacert_root_cert_size (size_t ∗cert_size)

  Get the size of the TNG root cert.
- int tng_atcacert_root_cert (uint8_t ∗cert, size_t ∗cert_size)

  Get the TNG root cert.
- int tng_atcacert_root_public_key (uint8_t ∗public_key)

  Gets the root public key.

15.181.1 Detailed Description

Client side certificate I/O functions for TNG devices.

Copyright

15.182 tng_root_cert.c File Reference

TNG root certificate (DER)

```
#include <stdint.h>
#include <stddef.h>
```

Variables

- const uint8_t g_cryptoauth_root_ca_002_cert [501]
- const size_t g_cryptoauth_root_ca_002_cert_size = sizeof(g_cryptoauth_root_ca_002_cert)

15.182.1 Detailed Description

TNG root certificate (DER)

Copyright

15.182.2 Variable Documentation

15.182.2.1 g_cryptoauth_root_ca_002_cert

```
const uint8_t g_cryptoauth_root_ca_002_cert[501]
```

15.182.2.2 g_cryptoauth_root_ca_002_cert_size

```
const size_t g_cryptoauth_root_ca_002_cert_size = sizeof(g_cryptoauth_root_ca_002_cert)
```

15.183 tng_root_cert.h File Reference

TNG root certificate (DER)

```
#include <stdint.h>
```

- #define CRYPTOAUTH_ROOT_CA_002_PUBLIC_KEY_OFFSET 266
- const uint8_t g_cryptoauth_root_ca_002_cert [ ]
- const size_t g_cryptoauth_root_ca_002_cert_size

15.183.1 Detailed Description

TNG root certificate (DER)

Copyright

(c) 2015-2019 Microchip Technology Inc. and its subsidiaries.

15.184 tngtn_cert_def_1_signer.c File Reference

TNG TN signer certificate definition.

```
#include "atcacert/atcacert_def.h"
#include "tng22_cert_def_1_signer.h"
```

Variables

- const uint8_t g_tng22_cert_template_1_signer [ ]
- const atcacert_def_t g_tngtn_cert_def_1_signer

15.184.1 Detailed Description

TNG TN signer certificate definition.

Copyright

15.184.2 Variable Documentation

15.184.2.1 g_tng22_cert_template_1_signer

```
const uint8_t g_tng22_cert_template_1_signer[ ]
```

15.185 tngtn_cert_def_1_signer.h File Reference

TNG TN signer certificate definition.

```
#include "atcacert/atcacert_def.h"
```

Variables

- const atcacert_def_t g_tngtn_cert_def_1_signer

15.185.1 Detailed Description

TNG TN signer certificate definition.

Copyright

15.186 tngtn_cert_def_2_device.c File Reference

TNG TN device certificate definition.

```
#include "atcacert/atcacert_def.h"
#include "tng22_cert_def_2_device.h"
#include "tngtn_cert_def_1_signer.h"
```

Variables

- const uint8_t g_tng22_cert_template_2_device [ ]
- const atcacert_cert_element_t g_tng22_cert_elements_2_device [ ]
- const atcacert_def_t g_tngtn_cert_def_2_device

15.186.1 Detailed Description

TNG TN device certificate definition.

Copyright

(c) 2015-2019 Microchip Technology Inc. and its subsidiaries.

15.186.2 Variable Documentation

15.186.2.1 g_tng22_cert_elements_2_device

```
const atcacert_cert_element_t g_tng22_cert_elements_2_device[]
```

15.186.2.2 g_tng22_cert_template_2_device

```
const uint8_t g_tng22_cert_template_2_device[]
```

15.187 tngtn_cert_def_2_device.h File Reference

TNG TN device certificate definition.

```
#include "atcacert/atcacert_def.h"
```

Variables

- const atcacert_def_t g_tngtn_cert_def_2_device

15.187.1 Detailed Description

TNG TN device certificate definition.

Copyright

(c) 2015-2019 Microchip Technology Inc. and its subsidiaries.

Index