Arm® TrustZone® CryptoCell-312

SW Revision: r0p0

Software Developers Manual

Confidential – Final

Arm 100777_0000_02_en

Release Information

Software Developers Manual

Release information

Document History

Issue	Date	Confidentiality	Change
0000-00	20 July 2017	Confidential	First official release for boot r0p0-00bet0.
0000-01	21 October 2017	Confidential	First release for boot r0p0-00eac0.
0000-02	22 February 2018	Confidential	First release for runtime SW r0p0-00eac0.

Confidential Proprietary Notice

Confidentiality status

This document is Confidential. This document may only be used and distributed in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Product status

The information in this document is Final, that is for a developed product.

Web address

http://www.arm.com

Web address

http://www.arm.com

Contents

List of Tables

Preface

This preface introduces the Arm® TrustZone® CryptoCell-312 APIs.

About this book

This book provides an overview of the Arm® TrustZone® CryptoCell-312 software APIs.

Intended audience

This document is written for programmers using the CryptoCell-312 cryptographic APIs.

Familiarity with the basics of security and cryptography is assumed.

Using this book

This book is organized into the following chapters:

Chapter 1 - *Runtime APIs*

This chapter provides an overview of the Arm® TrustZone® CryptoCell-312 runtime-software APIs, including relevant Mbed™ TLS APIs.

Chapter 2 - *SBROM APIs*

This chapter provides an overview of the Arm® TrustZone® CryptoCell-312 SBROM APIs.

Appendix A – *Revisions*

This appendix describes the technical changes between released issues of this book.

For further support in using this document please contact your Arm® account/product manager.

Glossary

The Arm glossary is a list of terms used in Arm documentation, together with definitions for those terms. The Arm glossary does not contain terms that are industry standard unless the Arm meaning differs from the generally accepted meaning.

See the *Arm glossary* for more information.

Typographic Conventions

italic	Introduces special terminology, denotes cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
`monospace`	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
`monospace`	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`monospace` *italic*	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
`monospace bold`	Denotes language keywords when used outside example code.
<and>	Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example: `MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>`
SMALL CAPITALS	Used in body text for a few terms that have specific technical meanings that are defined in the *Arm Glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Additional Reading

Arm publications

This book contains information that is specific to this product. See the following documents for other relevant information:

- *Arm Architecture Reference Manual Armv8, for Armv8-A architecture profile* (**DDI0487A/012816**)

The following confidential books are available to licensees:

- *Arm® TrustZone® CryptoCell-312 Software Integrators Manual* (**Arm 100776**)

Referenced standards

This book contains references to the following specifications:

- *ANSI X9.31-1988: Public Key Cryptography Using Reversible Algorithms for the Financial Services Industry (rDSA), compliant excluding C.9.*
- *ANSI X9.42-2003: Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography, compliant with sections 7.1, 7.2, 7.3, 7.4, 7.5.1, 7.7.1, 7.7.2, 8.1.1, 8.1.2, 8.1.3, 8.1.4 and Annex B.*
- *X9.62-2005: Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA), compliant sections 7.2, 7.3, and 7.4.1 – prime curves.*
- *X9.63-2011: Public Key Cryptography for the Financial Services Industry – Key Agreement and Key Transport Using Elliptic Curve Cryptography, compliant with sections 5.2, 5.3, 5.4.1, 5.6.2, 5.6.3, 5.7, 5.9, 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7 and 6.8 (EC over FP).*
- *BSI AIS-31: Functionality Classes and Evaluation Methodology for True Random Number Generators, version 3.1, compliant in an implementation using FETRNG driver.*
- *ChaCha, a variant of Salsa20.*
- *Curve25519: New Diffie-Hellman Speed Records.*
- *Ed25519: High-Speed High-Security Signatures.*
- *FIPS Publication 180-4: Secure Hash Standard (SHS), compliant excluding support for truncated hash operation.*

- *FIPS Publication 186-4: Digital Signature Standard (DSS), compliant with sections 5.1, 6.2, 6.3, 6.4, B.1.2, B.2.2, B.3.6, B.4.2, C.3.1, C.3.3, C.3.5, C.9, and D.1.2.*
- *FIPS Publication 197: Advanced Encryption Standard, support only 128-bit and 256-bit keys.*
- *FIPS Publication 198-1: The Keyed-Hash Message Authentication Code (HMAC).*
- *ISO/IEC 9797-1: Message Authentication Codes (MACs) -- Part 1: Mechanisms using a block cipher, compliant with CBC-MAC without padding, output transformation based on sections 6.2, 6.3.1, 6.4, 6.5.1, and 7.1.*
- *ISO/IEC 18033-2:2006: Information technology -- Security techniques -- Encryption algorithms -- Part 2: Asymmetric ciphers, compliant with sections 10.2, 10.2.1, 10.2.3 and 10.2.4.*
- *IEEE 802.15.4: IEEE Standard for Local and metropolitan area networks— Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs), compliant with CCM* (section 7 and Annex B).*
- *IEEE 1363-2000: IEEE Standard for Standard Specifications for Public-Key Cryptography, compliant with sections 7.2.1, 8 (excluding 8.2.6, 8.2.7, 8.2.8, 8.2.9), 10.3, 11, 12.2, 13 (excluding RIPEMD-160) and 14 (excluding RIPEMD-160).*
- *NIST SP 800-22: A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications, the second phase in the CryptoCell-312 TRNG characterization process is compliant with this.*
- *NIST SP 800-38A: Recommendation for Block Cipher Modes of Operation: Methods and Techniques, compliant with sections 6.1, 6.2, 6.4, and 6.5.*
- *NIST SP 800-38B: Recommendation for Block Cipher Modes of Operation: the CMAC Mode for Authentication.*
- *NIST SP 800-38C: Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality.*
- *NIST SP 800-38D: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC.*
- *NIST SP 800-38F: Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping, compliant with section 6.*
- *NIST SP 800-56A Rev. 2: Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography, compliant with sections 5.1, 5.2, 5.3, 5.4, 5.5.1.1, 5.6.1, 5.6.2.3, 5.7.1.1, 5.7.1.2 and 5.8.2.*
- *NIST SP 800-90A: Recommendation for Random Number Generation Using Deterministic Random Bit Generators – App C., compliant with section 10.2 – DRBG mechanism based on block ciphers.*
- *NIST SP 800-90B: DRAFT Recommendation for the Entropy Sources Used for Random Bit Generation, compliant with section 4.4 tests in runtime SW.*
- *Public-Key Cryptography Standards (PKCS) #1: RSA Encryption Standard Version 1.5, November 1993*
- *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1, June 2002*
- *Public-Key Cryptography Standards (PKCS) #3: Diffie Hellman Key Agreement Standard*
- *RFC-2409: The Internet Key Exchange (IKE).*
- *RFC-3526: More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE).*
- *RFC-3610: Counter with CBC-MAC (CCM).*
- *RFC-4492: Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS).*
- *RFC-4615: The Advanced Encryption Standard-Cipher-based Message Authentication Code-Pseudo-Random Function-128 (AES-CMAC-PRF-128) Algorithm for the Internet Key Exchange Protocol (IKE).*

- *RFC-5114: Additional Diffie-Hellman Groups for Use with IETF Standards.*
- *RFC-5869: HMAC-based Extract-and-Expand Key Derivation Function (HKDF)*, May 2010.
- RFC-6979: Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA).
- *Standards for Efficient Cryptography Group (SECG): SEC1 Elliptic Curve Cryptography*, 2000.
- *Standards for Efficient Cryptography Group (SECG): SEC2 Recommended Elliptic Curve Domain Parameters*, Version 1.0, 2000.
- *Wireless Transport Layer Security: Wireless Application Protocol (WAP-261-WTLS-20010406-a)*

Feedback

If you have comments on content, send an e-mail to *errata@arm.com*. Give:

- The title Arm® TrustZone® CryptoCell-312 Software Developers Manual.
- The number 100777_0000_02.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

Note

Arm tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.

1 Runtime APIs

This chapter provides an overview of the Arm® TrustZone® CryptoCell-312 runtime-software APIs, including relevant Mbed™ TLS APIs.

1.1 CryptoCell-312 runtime-software API overview

This documentation describes the runtime APIs provided by Arm CryptoCell-312. It provides the programmer with all information necessary for integrating and using the runtime APIs in the target environment.

The API layer enables using the CryptoCell cryptographic algorithms, for example, AES, hash, RSA and ECC.

Cryptographic algorithms can be divided into two main categories:

- Symmetric algorithms are mostly used for message confidentiality.
- The symmetric encryption algorithms are accessible via the generic cipher layer. For more information, see *mbedtls_cipher_setup()*.
- Asymmetric algorithms are mostly used for key exchange and message integrity.
- The asymmetric encryption algorithms are accessible via the generic public key layer.

The following algorithms are provided:

- Symmetric:
  - AES. *CryptoCell-312 hardware limitations for AES*.
- Asymmetric:
  - Diffie-Hellman-Merkle. See *mbedtls_dhm_read_public()*, *mbedtls_dhm_make_public()* and *mbedtls_dhm_calc_secret()*.
  - RSA. See *mbedtls_rsa_public()* and *mbedtls_rsa_private()*.
  - Elliptic Curves over GF(p). See *mbedtls_ecp_point_init()*.
  - Elliptic Curve Digital Signature Algorithm (ECDSA). See *mbedtls_ecdsa_init()*.
  - Elliptic Curve Diffie Hellman (ECDH). See *mbedtls_ecdh_init()*.

The documentation is automatically generated from the source code using Doxygen.

For more information on Doxygen, see *http://www.stack.nl/~dimitri/doxygen/*.

The *Modules* section introduces the high-level module concepts used throughout this documentation.

1.2 Deprecated list

- Global *mbedtls_aes_decrypt* (*mbedtls_aes_context* *ctx, const unsigned char input[16], unsigned char output[16])

  Superseded by mbedtls_aes_decrypt_ext() in 2.5.0.

- Global *mbedtls_aes_encrypt* (*mbedtls_aes_context* *ctx, const unsigned char input[16], unsigned char output[16])

  Superseded by mbedtls_aes_encrypt_ext() in 2.5.0.

- Global *MBEDTLS_DHM_RFC3526_MODP_2048_P*

  The hex-encoded primes from RFC 3625 are deprecated and superseded by the corresponding macros providing them as binary constants. Their hex-encoded constants are likely to be removed in a future version of the library.

- Global *MBEDTLS_DHM_RFC5114_MODP_P*

  The hex-encoded primes from RFC 5114 are deprecated and are likely to be removed in a future version of the library without replacement.

- Global *mbedtls_md_init_ctx* (*mbedtls_md_context_t* *ctx, const mbedtls_md_info_t *md_info) MBEDTLS_DEPRECATED

  Superseded by *mbedtls_md_setup()* in 2.0.0

- Global *mbedtls_rsa_pkcs1_decrypt* (*mbedtls_rsa_context* *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, size_t *olen, const unsigned char *input, unsigned char *output, size_t output_max_len)

  It is deprecated and discouraged to call this function in *MBEDTLS_RSA_PUBLIC* mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to *MBEDTLS_RSA_PRIVATE*.

- Global *mbedtls_rsa_pkcs1_encrypt* (*mbedtls_rsa_context* *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, size_t ilen, const unsigned char *input, unsigned char *output)

  It is deprecated and discouraged to call this function in *MBEDTLS_RSA_PRIVATE* mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to *MBEDTLS_RSA_PUBLIC*.

- Global *mbedtls_rsa_pkcs1_sign* (*mbedtls_rsa_context* *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, mbedtls_md_type_t md_alg, unsigned int hashlen, const unsigned char *hash, unsigned char *sig)

  It is deprecated and discouraged to call this function in *MBEDTLS_RSA_PUBLIC* mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to *MBEDTLS_RSA_PRIVATE*.

- Global *mbedtls_rsa_pkcs1_verify* (*mbedtls_rsa_context* *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, mbedtls_md_type_t md_alg, unsigned int hashlen, const unsigned char *hash, const unsigned char *sig)

  It is deprecated and discouraged to call this function in *MBEDTLS_RSA_PRIVATE* mode. Future versions of the library are likely to remove the mode argument and have it set to *MBEDTLS_RSA_PUBLIC*.

- Global *mbedtls_rsa_rsaes_oaep_decrypt* (*mbedtls_rsa_context* *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, const unsigned char *label, size_t label_len, size_t *olen, const unsigned char *input, unsigned char *output, size_t output_max_len)

  It is deprecated and discouraged to call this function in *MBEDTLS_RSA_PUBLIC* mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to *MBEDTLS_RSA_PRIVATE*.

- Global *mbedtls_rsa_rsaes_oaep_encrypt* (*mbedtls_rsa_context* *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, const unsigned char *label, size_t label_len, size_t ilen, const unsigned char *input, unsigned char *output)

  It is deprecated and discouraged to call this function in *MBEDTLS_RSA_PRIVATE* mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to *MBEDTLS_RSA_PUBLIC*.

- Global *mbedtls_rsa_rsaes_pkcs1_v15_decrypt* (*mbedtls_rsa_context* *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, size_t *olen, const unsigned char *input, unsigned char *output, size_t output_max_len)

  It is deprecated and discouraged to call this function in *MBEDTLS_RSA_PUBLIC* mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to *MBEDTLS_RSA_PRIVATE*.

- Global *mbedtls_rsa_rsaes_pkcs1_v15_encrypt* (*mbedtls_rsa_context* *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, size_t ilen, const unsigned char *input, unsigned char *output)

  It is deprecated and discouraged to call this function in *MBEDTLS_RSA_PRIVATE* mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to *MBEDTLS_RSA_PUBLIC*.

- Global *mbedtls_rsa_rsassa_pkcs1_v15_sign* (*mbedtls_rsa_context* *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, mbedtls_md_type_t md_alg, unsigned int hashlen, const unsigned char *hash, unsigned char *sig)

  It is deprecated and discouraged to call this function in *MBEDTLS_RSA_PUBLIC* mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to *MBEDTLS_RSA_PRIVATE*.

- Global *mbedtls_rsa_rsassa_pkcs1_v15_verify* (*mbedtls_rsa_context* *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, mbedtls_md_type_t md_alg, unsigned int hashlen, const unsigned char *hash, const unsigned char *sig)

  It is deprecated and discouraged to call this function in *MBEDTLS_RSA_PRIVATE* mode. Future versions of the library are likely to remove the mode argument and have it set to *MBEDTLS_RSA_PUBLIC*.

- Global *mbedtls_rsa_rsassa_pss_sign* (*mbedtls_rsa_context* *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, mbedtls_md_type_t md_alg, unsigned int hashlen, const unsigned char *hash, unsigned char *sig)

  It is deprecated and discouraged to call this function in *MBEDTLS_RSA_PUBLIC* mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to *MBEDTLS_RSA_PRIVATE*.

- Global *mbedtls_rsa_rsassa_pss_verify* (*mbedtls_rsa_context* *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, mbedtls_md_type_t md_alg, unsigned int hashlen, const unsigned char *hash, const unsigned char *sig)

It is deprecated and discouraged to call this function in *MBEDTLS_RSA_PRIVATE* mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to *MBEDTLS_RSA_PUBLIC*.

1.3 Data structures

Following are the data structures:

- *CC_PalTrngParams_t*
- *CCAesHwKeyData_t*
- *CCAesUserContext_t* (The context prototype of the user)
- *CCAesUserKeyData_t*
- *CCAssetBuff_t* (The asset buffer)
- *CCCmpuData_t*
- *CCCmpuUniqueBuff_t* (The device use of the unique buffer)
- *CCDmpuData_t*
- *CCDmpuHbkBuff_t* (The device use of the Hbk buffer)
- *CCEcdhFipsKatContext_t*
- *CCEcdhTempData_t*
- *CCEcdsaFipsKatContext_t*
- *CCEcdsaSignUserContext_t* (The context definition of the user for the signing operation)
- *CCEcdsaVerifyUserContext_t* (The context definition of the user for the verification operation)
- *CCEciesTempData_t*
- *CCEcpkiBuildTempData_t*
- *CCEcpkiDomain_t* (The structure containing the EC domain parameters in little-endian form)
- *CCEcpkiKgFipsContext_t*
- *CCEcpkiKgTempData_t*
- *CCEcpkiPointAffine_t*
- *CCEcpkiPrivKey_t*
- *CCEcpkiPublKey_t*
- *CCEcpkiUserPrivKey_t* (The user structure prototype of the EC private key)
- *CCEcpkiUserPublKey_t* (The user structure prototype of the EC public key)
- *CCHashUserContext_t* (The context prototype of the user)
- *CCRndContext_t*
- *CCRndState_t* (The structure for the RND state)
- *CCRndWorkBuff_t*
- *CCSbCertInfo_t*
- *EcdsaSignContext_t*
- *EcdsaVerifyContext_t*
- *HmacHash_t*
- *mbedtls_aes_context* (The AES context-type definition)
- *mbedtls_ccm_context* (The CCM context-type definition. The CCM context is passed to the APIs called)
- *mbedtls_chacha_user_context* (The context prototype of the user)
- *mbedtls_cipher_context_t*
- *mbedtls_cipher_info_t*
- *mbedtls_cmac_context_t*
- *mbedtls_ctr_drbg_context* (The CTR_DRBG context structure)
- *mbedtls_dhm_context* (The DHM context structure)

- *mbedtls_ecdh_context* (The ECDH context structure)
- *mbedtls_ecp_curve_info*
- *mbedtls_ecp_group* (ECP group structure)
- *mbedtls_ecp_keypair* (ECP key pair structure)
- *mbedtls_ecp_point* (ECP point structure (jacobian coordinates))
- *mbedtls_gcm_context* (The GCM context structure)
- *mbedtls_md_context_t*
- *mbedtls_mng_apbcconfig*
- *mbedtls_platform_context* (The platform context structure)
- *mbedtls_rsa_context* (The RSA context structure)
- *mbedtls_sha1_context* (The SHA-1 context structure)
- *mbedtls_sha256_context* (The SHA-256 context structure)
- *mbedtls_sha512_context* (The SHA-512 context structure)
- *mbedtls_srp_context*
- *mbedtls_srp_group_param* (Group parameters for the SRP)
- *mbedtls_util_keydata*

1.4 File list

Here is a list of all documented files with brief descriptions:

Table 1-1: List of files

Filename	Description
aes.h	This file contains the Mbed TLS AES APIs.
bootimagesverifier_def.h	This file contains definitions used for the Secure Boot and Secure Debug APIs.
cc_address_defs.h	This file contains general definitions for CryptoCell APIs.
cc_aes_defs.h	This file contains the type definitions that are used by the CryptoCell AES APIs.
cc_aes_defs_proj.h	This file contains definitions that are used for CryptoCell AES APIs.
cc_cmpu.h	This file contains all of the ICV production library APIs, their enums and definitions.
cc_dmpu.h	This file contains all of the OEM production library APIs, their enums and definitions.
cc_ecpki_domains_defs.h	This file contains CryptoCell ECPKI domains supported by the project.
cc_ecpki_types.h	This file contains all the type definitions that are used for the CryptoCell ECPKI APIs.
cc_error.h	This file defines the error return code types and the numbering spaces for each module of the layers listed.
cc_general_defs.h	This file contains general definitions of the CryptoCell runtime SW APIs.
cc_hash_defs.h	This file contains definitions of the CryptoCell hash APIs.
cc_hash_defs_proj.h	This file contains the project-specific definitions of hash APIs.
cc_lib.h	This file contains all of the CryptoCell library basic APIs, their enums and definitions.
cc_pal_abort.h	This file includes all PAL APIs.
cc_pal_barrier.h	This file contains the definitions and APIs for memory-barrier implementation.
cc_pal_compiler.h	This file contains CryptoCell PAL platform-dependent compiler-related definitions.
cc_pal_error.h	This file contains the error definitions of the platform-dependent PAL APIs.
cc_pal_init.h	This file contains the PAL layer entry point.
cc_pal_log.h	This file contains the PAL layer log definitions. The log is disabled by default.
cc_pal_mem.h	This file contains functions for memory operations.
cc_pal_memmap.h	This file contains functions for memory mapping.
cc_pal_mutex.h	This file contains functions for resource management (mutex operations).
cc_pal_pm.h	This file contains the definitions and APIs for power-management implementation.
cc_pal_sb_plat.h	This file contains platform-dependent definitions used in the Boot Services code.
cc_pal_trng.h	This file contains APIs for retrieving TRNG user parameters.

Filename	Description
cc_pal_types.h	This file contains definitions and types of CryptoCell PAL platform-dependent APIs.
cc_pka_defs_hw.h	This file contains all of the enums and definitions that are used in PKA APIs.
cc_pka_hw_plat_defs.h	This file contains the platform-dependent definitions of the CryptoCell PKA APIs.
cc_prod.h	This file contains all of the enums and definitions that are used for the ICV and OEM production libraries.
cc_prod_error.h	This file contains the error definitions of the CryptoCell production-library APIs.
cc_rnd_common.h	This file contains the CryptoCell random-number generation APIs.
cc_sram_map.h	This file contains internal SRAM mapping definitions.
cc_util_error.h	This file contains the error definitions of the CryptoCell utility APIs.
ccm.h	This file contains the Mbed TLS CCM APIs.
cipher.h	This file contains the Mbed TLS generic cipher wrapper.
cmac.h	This file contains the Mbed TLS CMAC APIs.
ctr_drbg.h	This file contains the Mbed TLS CTR_DRBG APIs.
dhm.h	This file contains the Mbed TLS Diffie-Hellman-Merkle key exchange APIs.
ecdh.h	This file contains the Mbed TLS Elliptic Curve Diffie-Hellman (ECDH) protocol APIs.
ecdsa.h	This file contains the Mbed TLS Elliptic Curve Digital Signature Algorithm (ECDSA) APIs.
ecp.h	This file contains the Mbed TLS elliptic curves over GF(p) APIs.
gcm.h	This file contains the Mbed TLS Galois/Counter Mode (GCM) APIs.
mbedtls_aes_ext_dma.h	This file contains all the CryptoCell AES external DMA APIs, their enums and definitions.
mbedtls_cc_aes_crypt_additional.h	This file contains all CryptoCell AES APIs that are currently not supported by Mbed TLS.
mbedtls_cc_aes_key_wrap.h	This file contains all of the CryptoCell key-wrapping APIs, their enums and definitions.
mbedtls_cc_aes_key_wrap_error.h	This file contains the error definitions of the CryptoCell AES key-wrapping APIs.
mbedtls_cc_ccm_star.h	This file contains the CryptoCell AES-CCM star APIs, their enums and definitions.
mbedtls_cc_chacha.h	This file contains all of the CryptoCell ChaCha APIs, their enums and definitions.
mbedtls_cc_chacha_error.h	This file contains the error definitions of the CryptoCell ChaCha APIs.
mbedtls_cc_chacha_poly.h	This file contains all of the CryptoCell ChaCha-POLY APIs, their enums and definitions.
mbedtls_cc_chacha_poly_error.h	This file contains the errors definitions of the CryptoCell ChaCha-POLY APIs.
mbedtls_cc_ecies.h	This file contains the CryptoCell Elliptic Curve Integrated Encryption Scheme (ECIES) APIs

Filename	Description
mbedtls_cc_hkdf.h	This file contains the CryptoCell HMAC key-derivation function API.
mbedtls_cc_hkdf_error.h	This file contains the error definitions of the CryptoCell HKDF APIs.
mbedtls_cc_mng.h	This file contains all the CryptoCell Management APIs, their enums and definitions.
mbedtls_cc_mng_error.h	This file contains the error definitions of the CryptoCell management APIs.
mbedtls_cc_poly.h	This file contains all of the CryptoCell POLY APIs, their enums and definitions.
mbedtls_cc_poly_error.h	This file contains the error definitions of the CryptoCell POLY APIs.
mbedtls_cc_sbrt.h	This file contains CryptoCell Secure Boot certificate-chain processing APIs.
mbedtls_cc_sha512_t.h	This file contains all of the CryptoCell SHA-512 truncated APIs, their enums and definitions.
mbedtls_cc_srp.h	This file contains all of the CryptoCell SRP APIs, their enums and definitions.
mbedtls_cc_srp_error.h	This file contains the error definitions of the CryptoCell SRP APIs.
mbedtls_cc_util_asset_prov.h	This file contains CryptoCell runtime-library ICV and OEM asset-provisioning APIs and definitions.
mbedtls_cc_util_defs.h	This file contains general definitions of the CryptoCell utility APIs.
mbedtls_cc_util_key_derivation.h	This file contains the CryptoCell utility key-derivation function APIs.
mbedtls_cc_util_key_derivation_defs.h	This file contains the definitions for the key-derivation API.
mbedtls_ccm_common.h	This file contains the common definitions of the CryptoCell AES-CCM star APIs.
mbedtls_chacha_ext_dma.h	This file contains all the CryptoCell ChaCha external DMA APIs, their enums and definitions.
mbedtls_ext_dma_error.h	This file contains the error definitions of the CryptoCell external DMA APIs.
mbedtls_hash_ext_dma.h	This file contains all the CryptoCell hash external DMA APIs, their enums and definitions.
md.h	This file contains the Mbed TLS generic message-digest wrapper.
platform.h	The Mbed TLS platform abstraction layer.
rsa.h	This file contains the Mbed TLS RSA public-key cryptosystem APIs.
secureboot_basetypes.h	This file contains basic type definitions for the Secure Boot.
secureboot_defs.h	This file contains type definitions for the Secure Boot.
secureboot_gen_defs.h	This file contains all of the definitions and structures used for the Secure Boot and Secure Debug.
sha1.h	The SHA-1 cryptographic hash function.
sha256.h	The SHA-224 and SHA-256 cryptographic hash function.
sha512.h	The SHA-384 and SHA-512 cryptographic hash function.

1.5 Module documentation

1.5.1 CryptoCell library basic APIs

Contains CryptoCell library basic APIs.

Macros

- #define *DX_VERSION_PRODUCT_BIT_SHIFT* 0x18UL
- #define *DX_VERSION_PRODUCT_BIT_SIZE* 0x8UL

Enumerations

- enum *CClibRetCode_t* { *CC_LIB_RET_OK* = 0, *CC_LIB_RET_EINVAL_CTX_PTR*, *CC_LIB_RET_EINVAL_WORK_BUF_PTR*, *CC_LIB_RET_HAL*, *CC_LIB_RET_PAL*, *CC_LIB_RET_RND_INST_ERR*, *CC_LIB_RET_EINVAL_PIDR*, *CC_LIB_RET_EINVAL_CIDR*, *CC_LIB_AO_WRITE_FAILED_ERR*, *CC_LIB_RESERVE32B* = 0x7FFFFFFFL }

Functions

- *CClibRetCode_t* *CC_LibInit* (*CCRndContext_t* *rndContext_ptr, *CCRndWorkBuff_t* *rndWorkBuff_ptr)

  This function performs global initialization of the CryptoCell runtime library.
- *CClibRetCode_t* *CC_LibFini* (*CCRndContext_t* *rndContext_ptr)

  This function finalizes library operations.

Detailed description

Contains CryptoCell library basic APIs.

Macro definition documentation

#define DX_VERSION_PRODUCT_BIT_SHIFT 0x18UL

Internal definition for the product register.

#define DX_VERSION_PRODUCT_BIT_SIZE 0x8UL

Internal definition for the product register size.

Enumeration type documentation

enum *CClibRetCode_t*

Definitions for error returns from *CC_LibInit* or *CC_LibFini* functions.

Enumerator:

Enum	Description
`CC_LIB_RET_OK`	Success definition.
`CC_LIB_RET_EINVAL_CTX_PTR`	Illegal context pointer.
`CC_LIB_RET_EINVAL_WORK_BUF_PTR`	Illegal work-buffer pointer.
`CC_LIB_RET_HAL`	Error returned from the HAL layer.

Enum	Description
`CC_LIB_RET_PAL`	Error returned from the PAL layer.
`CC_LIB_RET_RND_INST_ERR`	RND instantiation failed.
`CC_LIB_RET_EINVAL_PIDR`	Invalid peripheral ID.
`CC_LIB_RET_EINVAL_CIDR`	Invalid component ID.
`CC_LIB_AO_WRITE_FAILED_ERR`	Error returned from AO write operation.
`CC_LIB_RESERVE32B`	Reserved.

Function documentation

CClibRetCode_t CC_LibFini (CCRndContext_t * rndContext_ptr)

This function finalizes library operations.

It frees the associated resources (mutexes) and call HAL and PAL terminate functions. The function also cleans the RND context.

Returns:

CC_LIB_RET_OK on success.

A non-zero value on failure.

Parameters:

I/O	Parameter	Description
in,out	`rndContext_ptr`	A pointer to the RND context buffer that was initialized in *CC_LibInit*.

CClibRetCode_t CC_LibInit (CCRndContext_t * rndContext_ptr, CCRndWorkBuff_t * rndWorkBuff_ptr)

This function performs global initialization of the CryptoCell runtime library.

It must be called once per CryptoCell cold-boot cycle. Among other initializations, this function initializes the RND (TRNG seeding) context. An initialized RND context is required for calling RND APIs, as well as asymmetric-cryptography key-generation and signatures. The primary context returned by this function can be used as a single global context for all RND needs. Alternatively, other contexts may be initialized and used with a more limited scope, for specific applications or specific threads.

Note

The Mutexes, if used, are initialized by this API. Therefore, unlike the other APIs in the library, this API is not thread-safe.

Returns:

CC_LIB_RET_OK on success.

A non-zero value on failure.

Parameters:

I/O	Parameter	Description
in,out	`rndContext_ptr`	A pointer to the RND context buffer allocated by the user. The context is used to maintain the RND state as well as pointers to a function used for random vector generation. This context must be saved and provided as parameter to any API that uses the RND module.
in	`rndWorkBuff_ptr`	Scratchpad for the work of the RND module.

1.5.2 General base error codes for CryptoCell

Contains general base-error codes for CryptoCell.

Macros

- #define *CC_ERROR_BASE* 0x00F00000UL
- #define *CC_ERROR_LAYER_RANGE* 0x00010000UL
- #define *CC_ERROR_MODULE_RANGE* 0x00000100UL
- #define *CC_LAYER_ERROR_IDX* 0x00UL
- #define *LLF_LAYER_ERROR_IDX* 0x01UL
- #define *GENERIC_ERROR_IDX* 0x05UL
- #define *AES_ERROR_IDX* 0x00UL
- #define *DES_ERROR_IDX* 0x01UL
- #define *HASH_ERROR_IDX* 0x02UL
- #define *HMAC_ERROR_IDX* 0x03UL
- #define *RSA_ERROR_IDX* 0x04UL
- #define *DH_ERROR_IDX* 0x05UL
- #define *ECPKI_ERROR_IDX* 0x08UL
- #define *RND_ERROR_IDX* 0x0CUL
- #define *COMMON_ERROR_IDX* 0x0DUL
- #define *KDF_ERROR_IDX* 0x11UL
- #define *HKDF_ERROR_IDX* 0x12UL
- #define *AESCCM_ERROR_IDX* 0x15UL
- #define *FIPS_ERROR_IDX* 0x17UL
- #define *PKA_MODULE_ERROR_IDX* 0x21UL
- #define *CHACHA_ERROR_IDX* 0x22UL
- #define *EC_MONT_EDW_ERROR_IDX* 0x23UL
- #define *CHACHA_POLY_ERROR_IDX* 0x24UL
- #define *POLY_ERROR_IDX* 0x25UL
- #define *SRP_ERROR_IDX* 0x26UL
- #define *AESGCM_ERROR_IDX* 0x27UL
- #define *AES_KEYWRAP_ERROR_IDX* 0x28UL
- #define *MNG_ERROR_IDX* 0x29UL
- #define *PROD_ERROR_IDX* 0x2AUL
- #define *FFCDH_ERROR_IDX* 0x2BUL
- #define *FFC_DOMAIN_ERROR_IDX* 0x2CUL
- #define *EXT_DMA_ERROR_IDX* 0x2DUL

- #define *CC_AES_MODULE_ERROR_BASE*
- #define *CC_DES_MODULE_ERROR_BASE*
- #define *CC_HASH_MODULE_ERROR_BASE*
- #define *CC_HMAC_MODULE_ERROR_BASE*
- #define *CC_RSA_MODULE_ERROR_BASE*
- #define *CC_DH_MODULE_ERROR_BASE*
- #define *CC_ECPKI_MODULE_ERROR_BASE*
- #define *LLF_ECPKI_MODULE_ERROR_BASE*
- #define *CC_RND_MODULE_ERROR_BASE*
- #define *LLF_RND_MODULE_ERROR_BASE*
- #define *CC_COMMON_MODULE_ERROR_BASE*
- #define *CC_KDF_MODULE_ERROR_BASE*
- #define *CC_HKDF_MODULE_ERROR_BASE*
- #define *CC_AESCCM_MODULE_ERROR_BASE*
- #define *CC_FIPS_MODULE_ERROR_BASE*
- #define *PKA_MODULE_ERROR_BASE*
- #define *CC_CHACHA_MODULE_ERROR_BASE*
- #define *CC_EC_MONT_EDW_MODULE_ERROR_BASE*
- #define *CC_CHACHA_POLY_MODULE_ERROR_BASE*
- #define *CC_POLY_MODULE_ERROR_BASE*
- #define *CC_SRP_MODULE_ERROR_BASE*
- #define *CC_AESGCM_MODULE_ERROR_BASE*
- #define *CC_AES_KEYWRAP_MODULE_ERROR_BASE*
- #define *CC_MNG_MODULE_ERROR_BASE*
- #define *CC_PROD_MODULE_ERROR_BASE*
- #define *CC_FFCDH_MODULE_ERROR_BASE*
- #define *CC_FFC_DOMAIN_MODULE_ERROR_BASE*
- #define *CC_EXT_DMA_MODULE_ERROR_BASE*
- #define *GENERIC_ERROR_BASE* (*CC_ERROR_BASE* + (*CC_ERROR_LAYER_RANGE* * *GENERIC_ERROR_IDX*))
- #define *CC_FATAL_ERROR* (*GENERIC_ERROR_BASE* + 0x00UL)
- #define *CC_OUT_OF_RESOURCE_ERROR* (*GENERIC_ERROR_BASE* + 0x01UL)
- #define *CC_ILLEGAL_RESOURCE_VAL_ERROR* (*GENERIC_ERROR_BASE* + 0x02UL)
- #define *CC_CRYPTO_RETURN_ERROR*(retCode, retcodeInfo, funcHandler) ((retCode) == 0 ? *CC_OK* : funcHandler(retCode, retcodeInfo))

Detailed description

Contains general base-error codes for CryptoCell.

Macro definition documentation

#define AES_ERROR_IDX 0x00UL

The AES error index.

#define AES_KEYWRAP_ERROR_IDX 0x28UL

The AES key-wrap error index.

#define AESCCM_ERROR_IDX 0x15UL

The AESCCM error index.

#define AESGCM_ERROR_IDX 0x27UL

The AESGCM error index.

#define CC_AES_KEYWRAP_MODULE_ERROR_BASE

```
Value:(CC_ERROR_BASE + \
                          (CC_ERROR_LAYER_RANGE * CC_LAYER_ERROR_IDX) + \
                          (CC_ERROR_MODULE_RANGE * AES_KEYWRAP_ERROR_IDX ) )
```

The error base address of the AES key-wrap module - 0x00F02800.

#define CC_AES_MODULE_ERROR_BASE

```
Value:(CC_ERROR_BASE + \
                                     (CC_ERROR_LAYER_RANGE *
CC_LAYER_ERROR_IDX) + \
                                     (CC_ERROR_MODULE_RANGE * AES_ERROR_IDX
) )
```

The error base address of the AES module - 0x00F00000.

#define CC_AESCCM_MODULE_ERROR_BASE

```
Value:(CC_ERROR_BASE + \
                                       (CC_ERROR_LAYER_RANGE *
CC_LAYER_ERROR_IDX) + \
                                       (CC_ERROR_MODULE_RANGE *
AESCCM_ERROR_IDX ) )
```

The error base address of the AES CCM module - 0x00F01500.

#define CC_AESGCM_MODULE_ERROR_BASE

```
Value:(CC_ERROR_BASE + \
                                       (CC_ERROR_LAYER_RANGE *
CC_LAYER_ERROR_IDX) + \
                                       (CC_ERROR_MODULE_RANGE *
AESGCM_ERROR_IDX ) )
```

The error base address of the AES GCM module - 0x00F02700.

#define CC_CHACHA_MODULE_ERROR_BASE

```
Value:(CC_ERROR_BASE + \
                                           (CC_ERROR_LAYER_RANGE *
CC_LAYER_ERROR_IDX) + \
                                           (CC_ERROR_MODULE_RANGE *
CHACHA_ERROR_IDX ) )
```

The error base address of the ChaCha module - 0x00F02200.

#define CC_CHACHA_POLY_MODULE_ERROR_BASE

```
Value:(CC_ERROR_BASE + \
                                           (CC_ERROR_LAYER_RANGE *
CC_LAYER_ERROR_IDX) + \
                                           (CC_ERROR_MODULE_RANGE *
CHACHA_POLY_ERROR_IDX ) )
```

The error base address of the Chacha-POLY module - 0x00F02400.

#define CC_COMMON_MODULE_ERROR_BASE

```
Value:(CC_ERROR_BASE + \
                                    (CC_ERROR_LAYER_RANGE *
CC_LAYER_ERROR_IDX) + \
                                    (CC_ERROR_MODULE_RANGE *
COMMON_ERROR_IDX ) )
```

The error base address of the common module - 0x00F00D00.

#define CC_CRYPTO_RETURN_ERROR(retCode, retcodeInfo, funcHandler) ((retCode) == 0 ? CC_OK : funcHandler(retCode, retcodeInfo))

A macro that defines the CryptoCell return value.

#define CC_DES_MODULE_ERROR_BASE

```
Value:(CC_ERROR_BASE + \
                                    (CC_ERROR_LAYER_RANGE *
CC_LAYER_ERROR_IDX) + \
                                    (CC_ERROR_MODULE_RANGE * DES_ERROR_IDX
) )
```

The error base address of the DES module - 0x00F00100.

#define CC_DH_MODULE_ERROR_BASE

```
Value:(CC_ERROR_BASE + \
                                  (CC_ERROR_LAYER_RANGE *
CC_LAYER_ERROR_IDX) + \
                                  (CC_ERROR_MODULE_RANGE * DH_ERROR_IDX ) )
```

The error base address of the DH module - 0x00F00500.

#define CC_EC_MONT_EDW_MODULE_ERROR_BASE

```
Value:(CC_ERROR_BASE + \
                                          (CC_ERROR_LAYER_RANGE *
CC_LAYER_ERROR_IDX) + \
                                          (CC_ERROR_MODULE_RANGE *
EC_MONT_EDW_ERROR_IDX ) )
```

The error base address of the EC MONT_EDW module - 0x00F02300.

#define CC_ECPKI_MODULE_ERROR_BASE

```
Value:(CC_ERROR_BASE + \
                                    (CC_ERROR_LAYER_RANGE *
CC_LAYER_ERROR_IDX) + \
                                    (CC_ERROR_MODULE_RANGE *
ECPKI_ERROR_IDX ) )
```

The error base address of the ECPKI module - 0x00F00800.

#define CC_ERROR_BASE 0x00F00000UL

The definitions of the error number-space used for the different modules

The error base number for CryptoCell.

#define CC_ERROR_LAYER_RANGE 0x00010000UL

The error range number assigned for each layer.

#define CC_ERROR_MODULE_RANGE   0x00000100UL

The error range number assigned to each module on its specified layer.

#define CC_EXT_DMA_MODULE_ERROR_BASE

```
Value:(CC_ERROR_BASE + \
                                            (CC_ERROR_LAYER_RANGE *
CC_LAYER_ERROR_IDX) + \
                                            (CC_ERROR_MODULE_RANGE *
EXT_DMA_ERROR_IDX ) )
```

The error base address of the External DMA module - 0x00F02B00.

#define CC_FATAL_ERROR   (GENERIC_ERROR_BASE + 0x00UL)

CryptoCell fatal error.

#define CC_FFC_DOMAIN_MODULE_ERROR_BASE

```
Value:(CC_ERROR_BASE + \
                                            (CC_ERROR_LAYER_RANGE *
CC_LAYER_ERROR_IDX) + \
                                            (CC_ERROR_MODULE_RANGE *
FFC_DOMAIN_ERROR_IDX ) )
```

The error base address of the FFCDH module - 0x00F02B00.

#define CC_FFCDH_MODULE_ERROR_BASE

```
Value:(CC_ERROR_BASE + \
                                            (CC_ERROR_LAYER_RANGE *
CC_LAYER_ERROR_IDX) + \
                                            (CC_ERROR_MODULE_RANGE *
FFCDH_ERROR_IDX ) )
```

The error base address of the FFCDH module - 0x00F02B00.

#define CC_FIPS_MODULE_ERROR_BASE

```
Value:(CC_ERROR_BASE + \
                                        (CC_ERROR_LAYER_RANGE *
CC_LAYER_ERROR_IDX) + \
                                        (CC_ERROR_MODULE_RANGE *
FIPS_ERROR_IDX ) )
```

The error base address of the FIPS module - 0x00F01700.

#define CC_HASH_MODULE_ERROR_BASE

```
Value:(CC_ERROR_BASE + \
                                      (CC_ERROR_LAYER_RANGE *
CC_LAYER_ERROR_IDX) + \
                                      (CC_ERROR_MODULE_RANGE * HASH_ERROR_IDX
) )
```

The error base address of the hash module - 0x00F00200.

#define CC_HKDF_MODULE_ERROR_BASE

```
Value:(CC_ERROR_BASE + \
                                 (CC_ERROR_LAYER_RANGE *
CC_LAYER_ERROR_IDX) + \
                                  (CC_ERROR_MODULE_RANGE * HKDF_ERROR_IDX )
)
```

The error base address of the HKDF module - 0x00F01100.

#define CC_HMAC_MODULE_ERROR_BASE

```
Value:(CC_ERROR_BASE + \
                                     (CC_ERROR_LAYER_RANGE *
CC_LAYER_ERROR_IDX) + \
                                     (CC_ERROR_MODULE_RANGE * HMAC_ERROR_IDX
) )
```

The error base address of the HMAC module - 0x00F00300.

#define CC_ILLEGAL_RESOURCE_VAL_ERROR (GENERIC_ERROR_BASE + 0x02UL)

CryptoCell illegal resource value error.

#define CC_KDF_MODULE_ERROR_BASE

```
Value:(CC_ERROR_BASE + \
                                  (CC_ERROR_LAYER_RANGE *
CC_LAYER_ERROR_IDX) + \
                                  (CC_ERROR_MODULE_RANGE * KDF_ERROR_IDX ) )
```

The error base address of the KDF module - 0x00F01100.

#define CC_LAYER_ERROR_IDX 0x00UL

The CryptoCell error-layer index.

#define CC_MNG_MODULE_ERROR_BASE

```
Value:(CC_ERROR_BASE + \
                        (CC_ERROR_LAYER_RANGE * CC_LAYER_ERROR_IDX) + \
                        (CC_ERROR_MODULE_RANGE * MNG_ERROR_IDX ) )
```

The error base address of the Management module - 0x00F02900.

#define CC_OUT_OF_RESOURCE_ERROR (GENERIC_ERROR_BASE + 0x01UL)

CryptoCell out of resources error.

#define CC_POLY_MODULE_ERROR_BASE

```
Value:(CC_ERROR_BASE + \
                                           (CC_ERROR_LAYER_RANGE *
CC_LAYER_ERROR_IDX) + \
                                           (CC_ERROR_MODULE_RANGE *
POLY_ERROR_IDX ) )
```

The error base address of the POLY module - 0x00F02500.

#define CC_PROD_MODULE_ERROR_BASE

```
Value:(CC_ERROR_BASE + \
                                     (CC_ERROR_LAYER_RANGE *
CC_LAYER_ERROR_IDX) + \
                                     (CC_ERROR_MODULE_RANGE * PROD_ERROR_IDX
) )
```

The error base address of the production library - 0x00F02A00

#define CC_RND_MODULE_ERROR_BASE

```
Value:(CC_ERROR_BASE + \
                                        (CC_ERROR_LAYER_RANGE *
CC_LAYER_ERROR_IDX) + \
                                        (CC_ERROR_MODULE_RANGE * RND_ERROR_IDX
) )
```

The error base address of the RND module - 0x00F00C00.

#define CC_RSA_MODULE_ERROR_BASE

```
Value:(CC_ERROR_BASE + \
                                      (CC_ERROR_LAYER_RANGE *
CC_LAYER_ERROR_IDX) + \
                                      (CC_ERROR_MODULE_RANGE * RSA_ERROR_IDX )
)
```

The error base address of the RSA module - 0x00F00400.

#define CC_SRP_MODULE_ERROR_BASE

```
Value:(CC_ERROR_BASE + \
                                              (CC_ERROR_LAYER_RANGE *
CC_LAYER_ERROR_IDX) + \
                                              (CC_ERROR_MODULE_RANGE *
SRP_ERROR_IDX ) )
```

The error base address of the SRP module - 0x00F02600.

#define CHACHA_ERROR_IDX   0x22UL

The ChaCha error index.

#define CHACHA_POLY_ERROR_IDX   0x24UL

The ChaCha-POLY error index.

#define COMMON_ERROR_IDX   0x0DUL

The Common error index.

#define DES_ERROR_IDX   0x01UL

The DES error index.

#define DH_ERROR_IDX   0x05UL

The DH error index.

#define EC_MONT_EDW_ERROR_IDX   0x23UL

The EC Montgomery and Edwards error index.

#define ECPKI_ERROR_IDX   0x08UL

The ECPKI error index.

#define EXT_DMA_ERROR_IDX   0x2DUL

The external DMA error index.

#define FFC_DOMAIN_ERROR_IDX   0x2CUL

The FFC domain error index.

#define FFCDH_ERROR_IDX   0x2BUL

The FFCDH error index.

#define FIPS_ERROR_IDX   0x17UL

The FIPS error index.

#define GENERIC_ERROR_BASE   (CC_ERROR_BASE + (CC_ERROR_LAYER_RANGE * GENERIC_ERROR_IDX))

The generic error base address of the user - 0x00F50000

#define GENERIC_ERROR_IDX   0x05UL

The generic error-layer index.

#define HASH_ERROR_IDX   0x02UL

The hash error index.

#define HKDF_ERROR_IDX   0x12UL

The HKDF error index.

#define HMAC_ERROR_IDX   0x03UL

The HMAC error index.

#define KDF_ERROR_IDX   0x11UL

The KDF error index.

#define LLF_ECPKI_MODULE_ERROR_BASE

```
Value:(CC_ERROR_BASE + \
                                    (CC_ERROR_LAYER_RANGE *
LLF_LAYER_ERROR_IDX) + \
                                    (CC_ERROR_MODULE_RANGE *
ECPKI_ERROR_IDX ) )
```

The error base address of the low-level ECPKI module - 0x00F10800.

#define LLF_LAYER_ERROR_IDX   0x01UL

The error-layer index for low-level functions.

#define LLF_RND_MODULE_ERROR_BASE

```
Value:(CC_ERROR_BASE + \
                                    (CC_ERROR_LAYER_RANGE *
LLF_LAYER_ERROR_IDX) + \
                                    (CC_ERROR_MODULE_RANGE * RND_ERROR_IDX
) )
```

The error base address of the low-level RND module - 0x00F10C00.

#define MNG_ERROR_IDX   0x29UL

The management error index.

#define PKA_MODULE_ERROR_BASE

```
Value:(CC_ERROR_BASE + \
                                          (CC_ERROR_LAYER_RANGE *
CC_LAYER_ERROR_IDX) + \
                                          (CC_ERROR_MODULE_RANGE *
PKA_MODULE_ERROR_IDX ) )
```

The error base address of the PKA module - 0x00F02100.

#define PKA_MODULE_ERROR_IDX 0x21UL

The PKA error index.

#define POLY_ERROR_IDX 0x25UL

The POLY error index.

#define PROD_ERROR_IDX 0x2AUL

The production error index.

#define RND_ERROR_IDX 0x0CUL

The RND error index.

#define RSA_ERROR_IDX 0x04UL

The RSA error index.

#define SRP_ERROR_IDX 0x26UL

The SRP error index.

1.5.3 CryptoCell general definitions

Contains general definitions of the CryptoCell runtime SW APIs.

Data structures

- struct *HmacHash_t*

Macros

- #define *CC_HASH_NAME_MAX_SIZE* 10
- #define *CC_AES_KDR_MAX_SIZE_BYTES* 32
- #define *CC_AES_KDR_MAX_SIZE_WORDS* (*CC_AES_KDR_MAX_SIZE_BYTES*/sizeof(uint32_t))
- #define *CC_LCS_CHIP_MANUFACTURE_LCS* 0x0
- #define *CC_LCS_SECURE_LCS* 0x5

Typedefs

- typedef uint32_t *CCSramAddr_t*
- typedef uint32_t *CCDmaAddr_t*

Variables

- const *HmacHash_t* *HmacHashInfo_t* [*CC_HASH_NumOfModes*]
- const uint8_t *HmacSupportedHashModes_t* [*CC_HASH_NumOfModes*]
- const char *HashAlgMode2mbedtlsString* [*CC_HASH_NumOfModes*][*CC_HASH_NAME_MAX_SIZE*]

Detailed description

Contains general definitions of the CryptoCell runtime SW APIs.

Contains general CryptoCell definitions.

Macro definition documentation

#define CC_AES_KDR_MAX_SIZE_BYTES   32

Maximal size of AES HUK in Bytes.

#define CC_AES_KDR_MAX_SIZE_WORDS   (*CC_AES_KDR_MAX_SIZE_BYTES*/sizeof(uint32_t))

Maximal size of AES HUK in words.

#define CC_HASH_NAME_MAX_SIZE   10

The maximal size of the hash string.

#define CC_LCS_CHIP_MANUFACTURE_LCS   0x0

Chip Manufacturer (CM) LCS definition. The CM LCS value.

#define CC_LCS_SECURE_LCS   0x5

Secure LCS definition. The Secure LCS value.

Typedef documentation

typedef uint32_t *CCDmaAddr_t*

The DMA address type.

typedef uint32_t *CCSramAddr_t*

The SRAM address type.

Variable Documentation

const char HashAlgMode2mbedtlsString[*CC_HASH_NumOfModes*][*CC_HASH_NAME_MAX_SIZE*]

Hash string names.

const *HmacHash_t* HmacHashInfo_t[*CC_HASH_NumOfModes*]

Hash parameters for HMAC operation.

const uint8_t HmacSupportedHashModes_t[*CC_HASH_NumOfModes*]

Supported hash modes.

1.5.4 CryptoCell AES APIs

Contains all CryptoCell AES APIs.

AES is a symmetric block cipher that uses a combination of both substitution and permutation. It is fast in both software and hardware.

AES has a fixed block size of 128 bits, and supports the following key sizes:

- 128 bits.
- 192 bits.
- 256 bits.

For the implementation of AES, see *aes.h*.

CryptoCell-312 hardware limitations for AES

The CrytoCell-312 hardware accelerates the following AES operations:

- ECB.
- CBC.
- CTR.
- CMAC. For the implementation of CMAC, see *cmac.h*.
- OFB.
- CCM. For the implementation of CCM, see *ccm.h*.
- CCM star. For the implementation of CCM star, see *mbedtls_cc_ccm_star.h*.
- GCM. For the implementation of GCM, see *gcm.h*.

To support the accelerated algorithms, the following conditions must be met:

- The input and output buffers must be DMA-able.
- The input and output buffers must be physically contingous blocks in memory.
- Buffer size must be up to 64KB.
- The context must also be DMA-able, as partial and final results are written to the context.
- Only integrated operations are supported for CCM, CCM star and GCM algorithms.

Typical usage of AES in CryptoCell-312

The following is a typical AES Block operation flow:

1. *mbedtls_aes_init()*.
2. *mbedtls_aes_setkey_enc()*.
3. mbedtls_aes_crypt_cbc().

Modules

- *CryptoCell-specific AES APIs*
- Contains all CryptoCell AES APIs currently not supported by MbedTls. *Definitions of CryptoCell AES APIs*
- Contains CryptoCell AES API type definitions. *CryptoCell AES-CCM star APIs*
- Contains the CryptoCell AES-CCM star APIs. *CryptoCell AES key-wrapping APIs*
- Contains CryptoCell key-wrapping APIs. *CryptoCell-312 hardware limitations for AES*
- *Typical usage of AES in CryptoCell-312*

CryptoCell-specific AES APIs

Contains all CryptoCell AES APIs currently not supported by MbedTls.

Functions

- int *mbedtls_aes_crypt_ofb* (*mbedtls_aes_context* *ctx, size_t length, size_t *nc_off, unsigned char nonce_counter[16], unsigned char stream_block[16], const unsigned char *input, unsigned char *output)

  This function encrypts or decrypts AES-OFB buffer.

Detailed description

Contains all CryptoCell AES APIs currently not supported by MbedTls.

Function documentation

int mbedtls_aes_crypt_ofb (*mbedtls_aes_context* * ctx, size_t length, size_t * nc_off, unsigned char nonce_counter[16], unsigned char stream_block[16], const unsigned char * input, unsigned char * output)

This function encrypts or decrypts AES-OFB buffer.

Note

Due to the nature of OFB, you must use the same key schedule for both encryption and decryption, so that a context is initialized with *mbedtls_aes_setkey_enc()* for both *MBEDTLS_AES_ENCRYPT* and *MBEDTLS_AES_DECRYPT*.

Parameters:

Parameter	Description
`ctx`	The AES context.
`length`	The length of the data.
`nc_off`	Not supported. Set to 0.
`nonce_counter`	The 128-bit nonce and counter.
`stream_block`	Not supported.
`input`	The input data stream.
`output`	The output data stream.

Returns:

0 on success.

Definitions of CryptoCell AES APIs

Contains CryptoCell AES API type definitions.

Data structures

- struct *CCAesUserContext_t*
- The context prototype of the user. struct *CCAesUserKeyData_t*
- struct *CCAesHwKeyData_t*

Macros

- #define *CC_AES_CRYPTO_BLOCK_SIZE_IN_WORDS* 4
- #define *CC_AES_BLOCK_SIZE_IN_BYTES* (*CC_AES_CRYPTO_BLOCK_SIZE_IN_WORDS* * sizeof(uint32_t))
- #define *CC_AES_IV_SIZE_IN_WORDS* *CC_AES_CRYPTO_BLOCK_SIZE_IN_WORDS*
- #define *CC_AES_IV_SIZE_IN_BYTES* (*CC_AES_IV_SIZE_IN_WORDS* * sizeof(uint32_t))
- #define *CC_AES_USER_CTX_SIZE_IN_WORDS* (4+8+8+4)
- #define *CC_AES_KEY_MAX_SIZE_IN_WORDS* 8
- #define *CC_AES_KEY_MAX_SIZE_IN_BYTES* (*CC_AES_KEY_MAX_SIZE_IN_WORDS* * sizeof(uint32_t))

Typedefs

- typedef uint8_t *CCAesIv_t*[*CC_AES_IV_SIZE_IN_BYTES*]
- typedef uint8_t *CCAesKeyBuffer_t*[*CC_AES_KEY_MAX_SIZE_IN_BYTES*]

- typedef struct *CCAesUserContext_t* *CCAesUserContext_t*

  The context prototype of the user.

- typedef struct *CCAesUserKeyData_t* *CCAesUserKeyData_t*
- typedef struct *CCAesHwKeyData_t* *CCAesHwKeyData_t*

Enumerations

- enum *CCAesEncryptMode_t* { *CC_AES_ENCRYPT* = 0, *CC_AES_DECRYPT* = 1, *CC_AES_NUM_OF_ENCRYPT_MODES*, *CC_AES_ENCRYPT_MODE_LAST* = 0x7FFFFFFF }
- enum *CCAesOperationMode_t* { *CC_AES_MODE_ECB* = 0, *CC_AES_MODE_CBC* = 1, *CC_AES_MODE_CBC_MAC* = 2, *CC_AES_MODE_CTR* = 3, *CC_AES_MODE_XCBC_MAC* = 4, *CC_AES_MODE_CMAC* = 5, *CC_AES_MODE_XTS* = 6, *CC_AES_MODE_CBC_CTS* = 7, *CC_AES_MODE_OFB* = 8, *CC_AES_NUM_OF_OPERATION_MODES*, *CC_AES_OPERATION_MODE_LAST* = 0x7FFFFFFF }
- enum *CCAesPaddingType_t* { *CC_AES_PADDING_NONE* = 0, *CC_AES_PADDING_PKCS7* = 1, *CC_AES_NUM_OF_PADDING_TYPES*, *CC_AES_PADDING_TYPE_LAST* = 0x7FFFFFFF }
- enum *CCAesKeyType_t* { *CC_AES_USER_KEY* = 0, *CC_AES_PLATFORM_KEY* = 1, *CC_AES_CUSTOMER_KEY* = 2, *CC_AES_NUM_OF_KEY_TYPES*, *CC_AES_KEY_TYPE_LAST* = 0x7FFFFFFF }

Detailed description

Contains CryptoCell AES API type definitions.

Macro definition documentation

#define CC_AES_BLOCK_SIZE_IN_BYTES (*CC_AES_CRYPTO_BLOCK_SIZE_IN_WORDS* * sizeof(uint32_t))

The size of the AES block in Bytes.

#define CC_AES_CRYPTO_BLOCK_SIZE_IN_WORDS 4

The size of the AES block in words.

#define CC_AES_IV_SIZE_IN_BYTES (*CC_AES_IV_SIZE_IN_WORDS* * sizeof(uint32_t))

The size of the IV buffer in Bytes.

#define CC_AES_IV_SIZE_IN_WORDS *CC_AES_CRYPTO_BLOCK_SIZE_IN_WORDS*

The size of the IV buffer in words.

#define CC_AES_KEY_MAX_SIZE_IN_BYTES (*CC_AES_KEY_MAX_SIZE_IN_WORDS* * sizeof(uint32_t))

The maximal size of the AES key in bytes.

#define CC_AES_KEY_MAX_SIZE_IN_WORDS 8

The maximal size of the AES key in words.

#define CC_AES_USER_CTX_SIZE_IN_WORDS (4+8+8+4)

The size of the context prototype of the user in words. See *CCAesUserContext_t*.

Typedef documentation

typedef struct CCAesHwKeyData_t CCAesHwKeyData_t

The AES HW key Data.

typedef uint8_t CCAesIv_t[CC_AES_IV_SIZE_IN_BYTES]

Defines the IV buffer. A 16-Byte array.

typedef uint8_t CCAesKeyBuffer_t[CC_AES_KEY_MAX_SIZE_IN_BYTES]

Defines the AES key data buffer.

typedef struct CCAesUserContext_t CCAesUserContext_t

The context prototype of the user.

The argument type that is passed by the user to the AES APIs. The context saves the state of the operation, and must be saved by the user till the end of the API flow.

typedef struct CCAesUserKeyData_t CCAesUserKeyData_t

The AES key data of the user.

Enumeration type documentation

enum CCAesEncryptMode_t

The AES operation:

- Encrypt
- Decrypt

Enumerator:

Enum	Description
CC_AES_ENCRYPT	An AES encrypt operation.
CC_AES_DECRYPT	An AES decrypt operation.
CC_AES_NUM_OF_ENCRYPT_MODES	The maximal number of operations.
CC_AES_ENCRYPT_MODE_LAST	Reserved.

enum CCAesKeyType_t

The AES key type.

Enumerator:

Enum	Description
CC_AES_USER_KEY	The user key.
CC_AES_PLATFORM_KEY	The Kplt hardware key.
CC_AES_CUSTOMER_KEY	The Kcst hardware key.
CC_AES_NUM_OF_KEY_TYPES	The maximal number of AES key types.
CC_AES_KEY_TYPE_LAST	Reserved.

enum CCAesOperationMode_t

The AES operation mode.

Enumerator:

Enum	Description
CC_AES_MODE_ECB	ECB mode.
CC_AES_MODE_CBC	CBC mode.
CC_AES_MODE_CBC_MAC	CBC-MAC mode.
CC_AES_MODE_CTR	CTR mode.
CC_AES_MODE_XCBC_MAC	XCBC-MAC mode.
CC_AES_MODE_CMAC	CMAC mode.
CC_AES_MODE_XTS	XTS mode.
CC_AES_MODE_CBC_CTS	CBC-CTS mode.
CC_AES_MODE_OFB	OFB mode.
CC_AES_NUM_OF_OPERATION_MODES	The maximal number of AES modes.
CC_AES_OPERATION_MODE_LAST	Reserved.

enum *CCAesPaddingType_t*

The AES padding type.

Enumerator:

Enum	Description
CC_AES_PADDING_NONE	No padding.
CC_AES_PADDING_PKCS7	PKCS7 padding.
CC_AES_NUM_OF_PADDING_TYPES	The maximal number of AES padding modes.
CC_AES_PADDING_TYPE_LAST	Reserved.

1.5.5 CryptoCell AES-CCM star APIs

Contains the CryptoCell AES-CCM star APIs.

Modules

- *Common definitions of the CryptoCell AES-CCM star APIs*

Contains the CryptoCell AES-CCM star APIs. Functions

- void *mbedtls_ccm_star_init* (*mbedtls_ccm_context* *ctx)

  This function initializes the CCM star context.

- int *mbedtls_ccm_star_setkey* (*mbedtls_ccm_context* *ctx, *mbedtls_cipher_id_t* cipher, const unsigned char *key, unsigned int keybits)

  This function initializes the CCM star context set in the ctx parameter and sets the encryption or decryption key.

- void *mbedtls_ccm_star_free* (*mbedtls_ccm_context* *ctx)

  This function releases and clears the specified CCM star context and underlying cipher sub-context.

- int *mbedtls_ccm_star_encrypt_and_tag* (*mbedtls_ccm_context* *ctx, size_t length, const unsigned char *iv, size_t iv_len, const unsigned char *add, size_t add_len, const unsigned char *input, unsigned char *output, unsigned char *tag, size_t tag_len)

  This function encrypts a buffer using CCM star.

- int *mbedtls_ccm_star_auth_decrypt* (*mbedtls_ccm_context* *ctx, size_t length, const unsigned char *iv, size_t iv_len, const unsigned char *add, size_t add_len, const unsigned char *input, unsigned char *output, const unsigned char *tag, size_t tag_len)

  This function performs a CCM star authenticated decryption of a buffer.

- int *mbedtls_ccm_star_nonce_generate* (unsigned char *src_addr, uint32_t frame_counter, uint8_t size_of_t, unsigned char *nonce_buf)

  This function receives the MAC source address, the frame counter, and the MAC size, and returns the required nonce for AES-CCM*, as defined in IEEE 802.15.4: IEEE Standard for Local and metropolitan area networks— Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs).

Detailed description

Contains the CryptoCell AES-CCM star APIs.

Function documentation

int mbedtls_ccm_star_auth_decrypt (*mbedtls_ccm_context* * ctx, size_t length, const unsigned char * iv, size_t iv_len, const unsigned char * add, size_t add_len, const unsigned char * input, unsigned char * output, const unsigned char * tag, size_t tag_len)

This function performs a CCM star authenticated decryption of a buffer.

Parameters:

Parameter	Description
`ctx`	The CCM star context to use for decryption.
`length`	The length of the input data in Bytes.
`iv`	Initialization vector.
`iv_len`	The length of the IV in Bytes.
`add`	The additional data field.
`add_len`	The length of additional data in Bytes.
`input`	The buffer holding the input data.
`output`	The buffer holding the output data.
`tag`	The buffer holding the tag.
`tag_len`	The length of the tag in Bytes.

Returns:

0 if successful and authenticated, or *MBEDTLS_ERR_CCM_AUTH_FAILED* if the tag does not match.

int mbedtls_ccm_star_encrypt_and_tag (*mbedtls_ccm_context* * ctx, size_t length, const unsigned char * iv, size_t iv_len, const unsigned char * add, size_t add_len, const unsigned char * input, unsigned char * output, unsigned char * tag, size_t tag_len)

This function encrypts a buffer using CCM star.

Parameters:

Parameter	Description
`ctx`	The CCM star context to use for encryption.
`length`	The length of the input data in Bytes.
`iv`	Initialization vector (nonce).
`iv_len`	The length of the IV in Bytes. Must be 13 Bytes.
`add`	The additional data field.
`add_len`	The length of additional data in Bytes. Must be less than 2^16 - 2^8.
`input`	The buffer holding the input data.
`output`	The buffer holding the output data. Must be at least `length` Bytes wide.
`tag`	The buffer holding the tag.
`tag_len`	The length of the tag to generate in Bytes: 4, 6, 8, 10, 14 or 16. **Note** The tag is written to a separate buffer. To concatenate the `tag` with the `output`, as done in RFC-3610: Counter with CBC-MAC (CCM) , use `tag = output + length`, and make sure that the output buffer is at least `length + tag_len` wide.

Returns:

0 on success.

void mbedtls_ccm_star_free (*mbedtls_ccm_context* * ctx)

This function releases and clears the specified CCM star context and underlying cipher sub-context.

Parameters:

Parameter	Description
`ctx`	The CCM star context to free.

void mbedtls_ccm_star_init (*mbedtls_ccm_context* * ctx)

This function initializes the CCM star context.

It makes the context ready for *mbedtls_ccm_star_setkey()* or *mbedtls_ccm_star_free()*.

Parameters:

Parameter	Description
`ctx`	The CCM star context to initialize.

int mbedtls_ccm_star_nonce_generate (unsigned char * src_addr, uint32_t frame_counter, uint8_t size_of_t, unsigned char * nonce_buf)

This function receives the MAC source address, the frame counter, and the MAC size, and returns the required nonce for AES-CCM*, as defined in IEEE 802.15.4: IEEE Standard for Local and metropolitan area networks— Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs).

Note

This API should be called before *mbedtls_ccm_star_init()*, and the generated nonce should be provided to this function.

Parameters:

Parameter	Description
`src_addr`	The MAC address in EUI-64 format.
`frame_counter`	The MAC frame counter.
`size_of_t`	The size of the AES-CCM* MAC tag in Bytes: 4, 6, 8, 10, 14 or 16.
`nonce_buf`	The required nonce for AES-CCM*.

Returns:

CC_OK on success. A non-zero value on failure, as defined cc_aesccm_error.h.

int mbedtls_ccm_star_setkey (*mbedtls_ccm_context* * ctx, *mbedtls_cipher_id_t* cipher, const unsigned char * key, unsigned int keybits)

This function initializes the CCM star context set in the ctx parameter and sets the encryption or decryption key.

Parameters:

Parameter	Description
`ctx`	The CCM star context to initialize.
`cipher`	The 128-bit block cipher to use.
`key`	The encryption or decryption key.
`keybits`	The size of the key in bits. This must be acceptable by the cipher.

Returns:

0 on success, or a cipher-specific error code on failure.

Common definitions of the CryptoCell AES-CCM star APIs

Contains the CryptoCell AES-CCM star APIs.

Macros

- #define *MBEDTLS_AESCCM_STAR_NONCE_SIZE_BYTES* 13
- #define *MBEDTLS_AESCCM_STAR_SOURCE_ADDRESS_SIZE_BYTES* 8
- #define *MBEDTLS_AESCCM_MODE_CCM* 0
- #define *MBEDTLS_AESCCM_MODE_STAR* 1

Detailed description

Contains the CryptoCell AES-CCM star APIs.

Macro definition documentation

#define MBEDTLS_AESCCM_MODE_CCM 0

AES CCM mode: CCM.

#define MBEDTLS_AESCCM_MODE_STAR 1

AES CCM mode: CCM star.

#define MBEDTLS_AESCCM_STAR_NONCE_SIZE_BYTES 13

The size of the AES CCM star nonce in Bytes.

#define MBEDTLS_AESCCM_STAR_SOURCE_ADDRESS_SIZE_BYTES 8

The size of source address of the AES CCM star in Bytes.

1.5.6 CryptoCell AES key-wrapping APIs

Contains CryptoCell key-wrapping APIs.

Modules

- *Specific errors of the CryptoCell AES key-wrapping APIs*

Contains the CryptoCell AES key-wrapping-API error definitions.

Macros

- #define *CC_AES_KEYWRAP_SEMIBLOCK_SIZE_BYTES* (*CC_AES_BLOCK_SIZE_IN_BYTES* >> 1)
- #define *CC_AES_KEYWRAP_SEMIBLOCK_SIZE_WORDS* (*CC_AES_KEYWRAP_SEMIBLOCK_SIZE_BYTES* >> 2)
- #define *CC_AES_KEYWRAP_SEMIBLOCK_TO_BYTES_SHFT* 3
- #define *CC_AES_KEYWRAP_MAX_PAD_LEN* 7
- #define *CC_AES_KEYWRAP_ICV1* {0xA6A6A6A6, 0xA6A6A6A6}
- #define *CC_AES_KEYWRAP_ICV2* {0xA65959A6, 0x00000000}

Typedefs

- typedef enum *keyWrapMode* *mbedtls_keywrap_mode_t*

Enumerations

- enum *keyWrapMode* { *CC_AES_KEYWRAP_KW_MODE* = 0, *CC_AES_KEYWRAP_KWP_MODE* = 1, *CC_AES_KEYWRAP_NUM_OF_MODES* = 2, *CC_AES_KEYWRAP_RESERVE32B* = INT32_MAX }

Functions

- CCError_t *mbedtls_aes_key_wrap* (*mbedtls_keywrap_mode_t* keyWrapFlag, uint8_t *keyBuf, size_t keySize, uint8_t *pPlainText, size_t plainTextSize, uint8_t *pCipherText, size_t *pCipherTextSize)

  This is the AES wrapping or encryption function.

- CCError_t *mbedtls_aes_key_unwrap* (*mbedtls_keywrap_mode_t* keyWrapFlag, uint8_t *keyBuf, size_t keySize, uint8_t *pCipherText, size_t cipherTextSize, uint8_t *pPlainText, size_t *pPlainTextSize)

  This is the AES unwrapping or decryption function.

Detailed description

Contains CryptoCell key-wrapping APIs.

Macro definition documentation

#define CC_AES_KEYWRAP_ICV1 {0xA6A6A6A6, 0xA6A6A6A6}

ICVs - Integrity Check Value

The 64-bit default ICV for KW mode.

#define CC_AES_KEYWRAP_ICV2 {0xA65959A6, 0x00000000}

The 32-bit default ICV for KWP mode.

#define CC_AES_KEYWRAP_MAX_PAD_LEN 7

AES key-wrapping with padding (KWP) maximum Bytes of padding.

#define CC_AES_KEYWRAP_SEMIBLOCK_SIZE_BYTES (*CC_AES_BLOCK_SIZE_IN_BYTES* >> 1)

The size of the AES key-wrapping semiblock in Bytes.

#define CC_AES_KEYWRAP_SEMIBLOCK_SIZE_WORDS (*CC_AES_KEYWRAP_SEMIBLOCK_SIZE_BYTES* >> 2)

The size of the AES key-wrapping semiblock in words.

#define CC_AES_KEYWRAP_SEMIBLOCK_TO_BYTES_SHFT 3

The AES key-wrapping semiblock to Bytes shift.

Typedef documentation

typedef enum *keyWrapMode* *mbedtls_keywrap_mode_t*

Supported modes of the AES key-wrapping operation: KW and KWP, as defined in NIST SP 800-38F: Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping.

Enumeration type documentation

enum *keyWrapMode*

Supported modes of the AES key-wrapping operation: KW and KWP, as defined in NIST SP 800-38F: Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping.

Enumerator:

Enum	Description
`CC_AES_KEYWRAP_KW_MODE`	KW mode.
`CC_AES_KEYWRAP_KWP_MODE`	KWP mode.
`CC_AES_KEYWRAP_NUM_OF_MODES`	Allowed number of AES key-wrapping modes.

Enum	Description
`CC_AES_KEYWRAP_RESERVE32B`	Reserved.

Function documentation

CCError_t mbedtls_aes_key_unwrap (*mbedtls_keywrap_mode_t* keyWrapFlag, uint8_t * keyBuf, size_t keySize, uint8_t * pCipherText, size_t cipherTextSize, uint8_t * pPlainText, size_t * pPlainTextSize)

This is the AES unwrapping or decryption function.

AES key-wrapping specifies a deterministic authenticated-encryption mode of operation of the AES, according to NIST SP 800-38F: Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping. Its purpose is to protect cryptographic keys. It uses units of 8 Bytes called semiblocks. The minimal number of input semiblocks is:

- For KW mode: 2 semiblocks.
- For KWP mode: 1 semiblock.

The maximal size of the output in bytes is 64KB. This is a system restriction. Input to key-wrapping includes the following elements:

- Payload - text data that is both authenticated and encrypted.
- Key - The encryption key for the AES operation.

Returns:

CC_OK on success.

A non-zero value on failure, as defined in *mbedtls_cc_aes_key_wrap_error.h*.

Parameters:

I/O	Parameter	Description
in	`keyWrapFlag`	The enumerator defining the key-wrapping mode: KW or KWP.
in	`keyBuf`	A pointer to AES key-wrapping key.
in	`keySize`	The size of the key in Bytes. Valid values are: ● 16 Bytes ● 24 Bytes ● 32 Bytes
in	`pCipherText`	A pointer to the cipher-text data for decryption. The buffer must be contiguous.
in	`cipherTextSize`	The size of the cipher-text data in Bytes.
out	`pPlainText`	A pointer to the plain-text output data. The buffer must be contiguous.
in,out	`pPlainTextSize`	● Input: A pointer to the size of the plain-text output data buffer. ● Output: The actual size of the plain-text output data in Bytes.

CCError_t mbedtls_aes_key_wrap (*mbedtls_keywrap_mode_t* keyWrapFlag, uint8_t * keyBuf, size_t keySize, uint8_t * pPlainText, size_t plainTextSize, uint8_t * pCipherText, size_t * pCipherTextSize)

This is the AES wrapping or encryption function.

AES key-wrapping specifies a deterministic authenticated-encryption mode of operation of the AES, according to NIST SP 800-38F: Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping. Its purpose is to protect cryptographic keys. It uses units of 8 Bytes called semiblocks. The minimal number of input semiblocks is:

- For KW mode: 2 semiblocks.
- For KWP mode: 1 semiblock.

The maximal size of the output in Bytes is 64KB. This is a system restriction. The input to key-wrapping includes the following elements:

- Payload - text data that is both authenticated and encrypted.
- Key - The encryption key for the AES operation.

Returns:

CC_OK	on success.

A non-zero value on failure, as defined in *mbedtls_cc_aes_key_wrap_error.h*.

Parameters:

I/O	Parameter	Description
in	`keyWrapFlag`	The key-wrapping mode: KW or KWP.
in	`keyBuf`	A pointer to AES key-wrapping key.
in	`keySize`	The size of the key in Bytes. Valid values are: ● 16 Bytes ● 24 Bytes ● 32 Bytes
in	`pPlainText`	A pointer to the plain-text data for encryption. The buffer must be contiguous.
in	`plainTextSize`	The size of the plain-text data in Bytes.
out	`pCipherText`	A pointer to the cipher-text output data. The buffer must be contiguous.
in,out	`pCipherTextSize`	● Input: A pointer to the size of the cipher-text output data buffer. ● Output: The actual size of the cipher-text output data in Bytes.

Specific errors of the CryptoCell AES key-wrapping APIs

Contains the CryptoCell AES key-wrapping-API error definitions.

Macros

- #define *CC_AES_KEYWRAP_DATA_IN_POINTER_INVALID_ERROR* (*CC_AES_KEYWRAP_MODULE_ERROR_BASE* + 0x00UL)
- #define *CC_AES_KEYWRAP_DATA_OUT_POINTER_INVALID_ERROR* (*CC_AES_KEYWRAP_MODULE_ERROR_BASE* + 0x01UL)
- #define *CC_AES_KEYWRAP_INVALID_KEY_POINTER_ERROR* (*CC_AES_KEYWRAP_MODULE_ERROR_BASE* + 0x02UL)

- #define CC_AES_KEYWRAP_ILLEGAL_KEY_SIZE_ERROR (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0x03UL)
- #define CC_AES_KEYWRAP_SEMIBLOCKS_NUM_ILLEGAL (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0x04UL)
- #define CC_AES_KEYWRAP_ILLEGAL_PARAMETER_PTR_ERROR (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0x05UL)
- #define CC_AES_KEYWRAP_INVALID_ENCRYPT_MODE_ERROR (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0x06UL)
- #define CC_AES_KEYWRAP_DATA_IN_SIZE_ILLEGAL (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0x07UL)
- #define CC_AES_KEYWRAP_DATA_OUT_SIZE_ILLEGAL (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0x08UL)
- #define CC_AES_KEYWRAP_INVALID_KEYWRAP_MODE_ERROR (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0x09UL)
- #define CC_AES_KEYWRAP_UNWRAP_COMPARISON_ERROR (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0x0AUL)
- #define CC_AES_KEYWRAP_IS_NOT_SUPPORTED (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0xFFUL)

Detailed description

Contains the CryptoCell AES key-wrapping-API error definitions.

Macro definition documentation

#define CC_AES_KEYWRAP_DATA_IN_POINTER_INVALID_ERROR (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0x00UL)

Invalid data-in text pointer.

#define CC_AES_KEYWRAP_DATA_IN_SIZE_ILLEGAL (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0x07UL)

Illegal data-in size.

#define CC_AES_KEYWRAP_DATA_OUT_POINTER_INVALID_ERROR (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0x01UL)

Invalid data-out text pointer.

#define CC_AES_KEYWRAP_DATA_OUT_SIZE_ILLEGAL (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0x08UL)

Illegal data-out size.

#define CC_AES_KEYWRAP_ILLEGAL_KEY_SIZE_ERROR (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0x03UL)

Invalid key size.

#define CC_AES_KEYWRAP_ILLEGAL_PARAMETER_PTR_ERROR (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0x05UL)

Invalid parameter pointer.

#define CC_AES_KEYWRAP_INVALID_ENCRYPT_MODE_ERROR (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0x06UL)

Invalid encryption mode.

#define CC_AES_KEYWRAP_INVALID_KEY_POINTER_ERROR (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0x02UL)

Invalid key pointer.

#define CC_AES_KEYWRAP_INVALID_KEYWRAP_MODE_ERROR (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0x09UL)

Illegal key-wrapping mode.

#define CC_AES_KEYWRAP_IS_NOT_SUPPORTED (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0xFFUL)

Not supported.

#define CC_AES_KEYWRAP_SEMIBLOCKS_NUM_ILLEGAL (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0x04UL)

Illegal semiblocks number.

#define CC_AES_KEYWRAP_UNWRAP_COMPARISON_ERROR (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0x0AUL)

Key Unwrap comparison failure.

1.5.7 CryptoCell ECPKI type definitions

Contains CryptoCell ECPKI API type definitions.

Data structures

- struct *CCEcpkiDomain_t*
- The structure containing the EC domain parameters in little-endian form. struct *CCEcpkiPointAffine_t*
- struct *CCEcpkiPublKey_t*
- struct *CCEcpkiUserPublKey_t*
- The user structure prototype of the EC public key. struct *CCEcpkiPrivKey_t*
- struct *CCEcpkiUserPrivKey_t*

- The user structure prototype of the EC private key. struct *CCEcdhTempData_t*
- struct *CCEcpkiBuildTempData_t*
- struct *EcdsaSignContext_t*
- struct *CCEcdsaSignUserContext_t*
- The context definition of the user for the signing operation. struct *EcdsaVerifyContext_t*
- struct *CCEcdsaVerifyUserContext_t*
- The context definition of the user for the verification operation. struct *CCEcpkiKgTempData_t*
- struct *CCEciesTempData_t*
- struct *CCEcpkiKgFipsContext_t*
- struct *CCEcdsaFipsKatContext_t*
- struct *CCEcdhFipsKatContext_t*

Macros

- #define *CC_PKA_DOMAIN_LLF_BUFF_SIZE_IN_WORDS* (10 + 3**CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS*)
- #define *CC_ECPKI_FIPS_ORDER_LENGTH* (256/*CC_BITS_IN_BYTE*)

Typedefs

- typedef struct *CCEcpkiUserPublKey_t* *CCEcpkiUserPublKey_t*

  The user structure prototype of the EC public key.
- typedef struct *CCEcpkiUserPrivKey_t* *CCEcpkiUserPrivKey_t*

  The user structure prototype of the EC private key.
- typedef struct *CCEcdhTempData_t* *CCEcdhTempData_t*
- typedef struct *CCEcpkiBuildTempData_t* *CCEcpkiBuildTempData_t*
- typedef uint32_t *CCEcdsaSignIntBuff_t*[*CC_PKA_ECDSA_SIGN_BUFF_MAX_LENGTH_IN_WORDS*]
- typedef struct *CCEcdsaSignUserContext_t* *CCEcdsaSignUserContext_t*

  The context definition of the user for the signing operation.
- typedef uint32_t *CCEcdsaVerifyIntBuff_t*[*CC_PKA_ECDSA_VERIFY_BUFF_MAX_LENGTH_IN_WORDS*]
- typedef struct *CCEcdsaVerifyUserContext_t* *CCEcdsaVerifyUserContext_t*

  The context definition of the user for the verification operation.
- typedef struct *CCEcpkiKgTempData_t* *CCEcpkiKgTempData_t*
- typedef struct *CCEciesTempData_t* *CCEciesTempData_t*
- typedef struct *CCEcpkiKgFipsContext_t* *CCEcpkiKgFipsContext_t*
- typedef struct *CCEcdsaFipsKatContext_t* *CCEcdsaFipsKatContext_t*
- typedef struct *CCEcdhFipsKatContext_t* *CCEcdhFipsKatContext_t*

Enumerations

- enum *CCEcpkiDomainID_t* { *CC_ECPKI_DomainID_secp192k1*, *CC_ECPKI_DomainID_secp192r1*, *CC_ECPKI_DomainID_secp224k1*, *CC_ECPKI_DomainID_secp224r1*, *CC_ECPKI_DomainID_secp256k1*, *CC_ECPKI_DomainID_secp256r1*, *CC_ECPKI_DomainID_secp384r1*, *CC_ECPKI_DomainID_secp521r1*, *CC_ECPKI_DomainID_Builded*, *CC_ECPKI_DomainID_OffMode*, *CC_ECPKI_DomainIDLast* = 0x7FFFFFFF }EC domain idetifiers.
- enum *CCEcpkiHashOpMode_t* { *CC_ECPKI_HASH_SHA1_mode* = 0, *CC_ECPKI_HASH_SHA224_mode* = 1, *CC_ECPKI_HASH_SHA256_mode* = 2, *CC_ECPKI_HASH_SHA384_mode* = 3, *CC_ECPKI_HASH_SHA512_mode* = 4, *CC_ECPKI_AFTER_HASH_SHA1_mode* = 5, *CC_ECPKI_AFTER_HASH_SHA224_mode* = 6, *CC_ECPKI_AFTER_HASH_SHA256_mode* = 7, *CC_ECPKI_AFTER_HASH_SHA384_mode* = 8, *CC_ECPKI_AFTER_HASH_SHA512_mode* = 9, *CC_ECPKI_HASH_NumOfModes*, *CC_ECPKI_HASH_OpModeLast* = 0x7FFFFFFF }Hash operation mode.
- enum *CCEcpkiPointCompression_t* { *CC_EC_PointCompressed* = 2, *CC_EC_PointUncompressed* = 4, *CC_EC_PointContWrong* = 5, *CC_EC_PointHybrid* = 6, *CC_EC_PointCompresOffMode* = 8, *CC_ECPKI_PointCompressionLast* = 0x7FFFFFFF }
- enum *ECPublKeyCheckMode_t* { *CheckPointersAndSizesOnly* = 0, *ECpublKeyPartlyCheck* = 1, *ECpublKeyFullCheck* = 2, PublKeyChecingOffMode, *EC_PublKeyCheckModeLast* = 0x7FFFFFFF }
- enum *CCEcpkiScaProtection_t* { SCAP_Inactive, *SCAP_Active*, *SCAP_OFF_MODE*, *SCAP_LAST* = 0x7FFFFFFF }

Detailed description

Contains CryptoCell ECPKI API type definitions.

Macro definition documentation

#define CC_ECPKI_FIPS_ORDER_LENGTH (256/*CC_BITS_IN_BYTE*)

The order length for FIPS ECC tests.

#define CC_PKA_DOMAIN_LLF_BUFF_SIZE_IN_WORDS (10 + 3**CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS*)

The size of the internal buffer in words.

Typedef documentation

typedef struct *CCEcdhFipsKatContext_t* *CCEcdhFipsKatContext_t*

ECDH KAT data structures for FIPS certification.

typedef struct *CCEcdhTempData_t* *CCEcdhTempData_t*

The type of the ECDH temporary data.

typedef struct *CCEcdsaFipsKatContext_t* *CCEcdsaFipsKatContext_t*

ECDSA KAT data structures for FIPS certification. The ECDSA KAT tests are defined for domain 256r1.

typedef uint32_t CCEcdsaSignIntBuff_t[*CC_PKA_ECDSA_SIGN_BUFF_MAX_LENGTH_IN_WORDS*]

The internal buffer used in the signing process.

typedef struct *CCEcdsaSignUserContext_t* *CCEcdsaSignUserContext_t*

The context definition of the user for the signing operation.

This context saves the state of the operation, and must be saved by the user until the end of the API flow.

typedef uint32_t CCEcdsaVerifyIntBuff_t[*CC_PKA_ECDSA_VERIFY_BUFF_MAX_LENGTH_IN_WORDS*]

The internal buffer used in the verification process.

typedef struct *CCEcdsaVerifyUserContext_t* *CCEcdsaVerifyUserContext_t*

The context definition of the user for the verification operation.

The context saves the state of the operation, and must be saved by the user until the end of the API flow.

typedef struct *CCEciesTempData_t* *CCEciesTempData_t*

The temporary data definition of the ECIES.

typedef struct *CCEcpkiBuildTempData_t* *CCEcpkiBuildTempData_t*

EC build temporary data.

typedef struct *CCEcpkiKgFipsContext_t* *CCEcpkiKgFipsContext_t*

ECPKI data structures for FIPS certification.

typedef struct *CCEcpkiKgTempData_t* *CCEcpkiKgTempData_t*

The temporary data type of the ECPKI KG.

typedef struct *CCEcpkiUserPrivKey_t* *CCEcpkiUserPrivKey_t*

The user structure prototype of the EC private key.

This structure must be saved by the user. It is used as input to ECC functions, for example, CC_EcdsaSign().

typedef struct *CCEcpkiUserPublKey_t* *CCEcpkiUserPublKey_t*

The user structure prototype of the EC public key.

This structure must be saved by the user. It is used as input to ECC functions, for example, CC_EcdsaVerify().

Enumeration type documentation

enum *CCEcpkiDomainID_t*

EC domain idetifiers.

For more information, see Standards for Efficient Cryptography Group (SECG): SEC2 Recommended Elliptic Curve Domain Parameters, Version 1.0.

Enumerator:

Enum	Description
`CC_ECPKI_DomainID_secp192k1`	EC secp192k1.

Enum	Description
`CC_ECPKI_DomainID_secp192r1`	EC secp192r1.
`CC_ECPKI_DomainID_secp224k1`	EC secp224k1.
`CC_ECPKI_DomainID_secp224r1`	EC secp224r1.
`CC_ECPKI_DomainID_secp256k1`	EC secp256k1.
`CC_ECPKI_DomainID_secp256r1`	EC secp256r1.
`CC_ECPKI_DomainID_secp384r1`	EC secp384r1.
`CC_ECPKI_DomainID_secp521r1`	EC secp521r1.
`CC_ECPKI_DomainID_Builded`	User given, not identified.
`CC_ECPKI_DomainID_OffMode`	Reserved.
`CC_ECPKI_DomainIDLast`	Reserved.

enum CCEcpkiHashOpMode_t

Hash operation mode.

Defines hash modes according to IEEE 1363-2000: IEEE Standard for Standard Specifications for Public-Key Cryptography.

Enumerator:

Enum	Description
`CC_ECPKI_HASH_SHA1_mode`	The message data will be hashed with SHA-1.
`CC_ECPKI_HASH_SHA224_mode`	The message data will be hashed with SHA-224.
`CC_ECPKI_HASH_SHA256_mode`	The message data will be hashed with SHA-256.
`CC_ECPKI_HASH_SHA384_mode`	The message data will be hashed with SHA-384.
`CC_ECPKI_HASH_SHA512_mode`	The message data will be hashed with SHA-512.
`CC_ECPKI_AFTER_HASH_SHA1_mode`	The message data is a digest of SHA-1 and will not be hashed.

Enum	Description
CC_ECPKI_AFTER_HASH_SHA224_mode	The message data is a digest of SHA-224 and will not be hashed.
CC_ECPKI_AFTER_HASH_SHA256_mode	The message data is a digest of SHA-256 and will not be hashed.
CC_ECPKI_AFTER_HASH_SHA384_mode	The message data is a digest of SHA-384 and will not be hashed.
CC_ECPKI_AFTER_HASH_SHA512_mode	The message data is a digest of SHA-512 and will not be hashed.
CC_ECPKI_HASH_NumOfModes	The maximal number of hash modes.
CC_ECPKI_HASH_OpModeLast	Reserved.

enum *CCEcpkiPointCompression_t*

EC point-compression identifiers.

Enumerator:

Enum	Description
CC_EC_PointCompressed	A compressed point.
CC_EC_PointUncompressed	An uncompressed point.
CC_EC_PointContWrong	An incorrect point-control value.
CC_EC_PointHybrid	A hybrid point.
CC_EC_PointCompresOffMode	Reserved.
CC_ECPKI_PointCompressionLast	Reserved.

enum *CCEcpkiScaProtection_t*

SW SCA protection type.

Enumerator:

Enum	Description
SCAP_Active	SCA protection inactive.
SCAP_OFF_MODE	SCA protection active.
SCAP_LAST	Reserved.

enum *ECPublKeyCheckMode_t*

EC key checks.

Enumerator:

Enum	Description
`CheckPointersAndSizesOnly`	Check only preliminary input parameters.
`ECpublKeyPartlyCheck`	Check preliminary input parameters and verify that the EC public-key point is on the curve.
`ECpublKeyFullCheck`	Check preliminary input parameters, verify that the EC public-key point is on the curve, and verify that EC_GeneratorOrder*PubKey = 0
`EC_PublKeyCheckModeLast`	Reserved.

1.5.8 CryptoCell ChaCha APIs

Contains all CryptoCell ChaCha APIs.

Modules

- *Specific errors of the CryptoCell ChaCha APIs*

Contains the CryptoCell ChaCha-API error definitions. Data structures

- struct *mbedtls_chacha_user_context*

The context prototype of the user. Macros

- #define *CC_CHACHA_USER_CTX_SIZE_IN_WORDS* 17
- #define *CC_CHACHA_BLOCK_SIZE_IN_WORDS* 16
- #define *CC_CHACHA_BLOCK_SIZE_IN_BYTES* (*CC_CHACHA_BLOCK_SIZE_IN_WORDS* * sizeof(uint32_t))
- #define *CC_CHACHA_NONCE_MAX_SIZE_IN_WORDS* 3
- #define *CC_CHACHA_NONCE_MAX_SIZE_IN_BYTES* (*CC_CHACHA_NONCE_MAX_SIZE_IN_WORDS* * sizeof(uint32_t))
- #define *CC_CHACHA_KEY_MAX_SIZE_IN_WORDS* 8
- #define *CC_CHACHA_KEY_MAX_SIZE_IN_BYTES* (*CC_CHACHA_KEY_MAX_SIZE_IN_WORDS* * sizeof(uint32_t))

Typedefs

- typedef uint8_t *mbedtls_chacha_nonce*[*CC_CHACHA_NONCE_MAX_SIZE_IN_BYTES*]
- typedef uint8_t *mbedtls_chacha_key*[*CC_CHACHA_KEY_MAX_SIZE_IN_BYTES*]
- typedef struct *mbedtls_chacha_user_context* *mbedtls_chacha_user_context*

The context prototype of the user.

Enumerations

- enum *mbedtls_chacha_encrypt_mode_t* { *CC_CHACHA_Encrypt* = 0, *CC_CHACHA_Decrypt* = 1, *CC_CHACHA_EncryptNumOfOptions*, *CC_CHACHA_EncryptModeLast* = 0x7FFFFFFF }
- enum *mbedtls_chacha_nonce_size_t* { *CC_CHACHA_Nonce64BitSize* = 0, *CC_CHACHA_Nonce96BitSize* = 1, *CC_CHACHA_NonceSizeNumOfOptions*, *CC_CHACHA_NonceSizeLast* = 0x7FFFFFFF }

Functions

- CIMPORT_C CCError_t *mbedtls_chacha_init* (*mbedtls_chacha_user_context* *pContextID, *mbedtls_chacha_nonce* pNonce, *mbedtls_chacha_nonce_size_t* nonceSize, *mbedtls_chacha_key* pKey, uint32_t initialCounter, *mbedtls_chacha_encrypt_mode_t* EncryptDecryptFlag)

  This function initializes the context for ChaCha-engine operations.
- CIMPORT_C CCError_t *mbedtls_chacha_block* (*mbedtls_chacha_user_context* *pContextID, uint8_t *pDataIn, size_t dataInSize, uint8_t *pDataOut)

  This function processes aligned blocks of the ChaCha engine.
- CIMPORT_C CCError_t *mbedtls_chacha_finish* (*mbedtls_chacha_user_context* *pContextID, uint8_t *pDataIn, size_t dataInSize, uint8_t *pDataOut)

  This function processes the remaining ChaCha data.
- CIMPORT_C CCError_t *mbedtls_chacha_free* (*mbedtls_chacha_user_context* *pContextID)

  This function frees the context used for ChaCha operations.
- CIMPORT_C CCError_t *mbedtls_chacha* (*mbedtls_chacha_nonce* pNonce, *mbedtls_chacha_nonce_size_t* nonceSize, *mbedtls_chacha_key* pKey, uint32_t initialCounter, *mbedtls_chacha_encrypt_mode_t* encryptDecryptFlag, uint8_t *pDataIn, size_t dataInSize, uint8_t *pDataOut)

  This function performs the ChaCha operation in one integrated process.

Detailed description

Contains all CryptoCell ChaCha APIs.

Macro definition documentation

#define CC_CHACHA_BLOCK_SIZE_IN_BYTES (*CC_CHACHA_BLOCK_SIZE_IN_WORDS* * sizeof(uint32_t))

The size of the ChaCha block in Bytes.

#define CC_CHACHA_BLOCK_SIZE_IN_WORDS 16

The size of the ChaCha block in words.

#define CC_CHACHA_KEY_MAX_SIZE_IN_BYTES (*CC_CHACHA_KEY_MAX_SIZE_IN_WORDS* * sizeof(uint32_t))

The maximal size of the ChaCha key in Bytes.

#define CC_CHACHA_KEY_MAX_SIZE_IN_WORDS 8

The maximal size of the ChaCha key in words.

#define CC_CHACHA_NONCE_MAX_SIZE_IN_BYTES (CC_CHACHA_NONCE_MAX_SIZE_IN_WORDS * sizeof(uint32_t))

The maximal size of the nonce buffer in Bytes.

#define CC_CHACHA_NONCE_MAX_SIZE_IN_WORDS 3

The maximal size of the nonce buffer in words.

#define CC_CHACHA_USER_CTX_SIZE_IN_WORDS 17

The size of the ChaCha user-context in words.

Typedef documentation

typedef uint8_t mbedtls_chacha_key[CC_CHACHA_KEY_MAX_SIZE_IN_BYTES]

The definition of the key buffer of the ChaCha engine.

typedef uint8_t mbedtls_chacha_nonce[CC_CHACHA_NONCE_MAX_SIZE_IN_BYTES]

The definition of the 12-Byte array of the nonce buffer.

typedef struct mbedtls_chacha_user_context mbedtls_chacha_user_context

The context prototype of the user.

The argument type that is passed by the user to the ChaCha API.

The context saves the state of the operation. It must be saved by the user until the end of the API flow, for example, until *mbedtls_chacha_free* is called.

Enumeration type documentation

enum mbedtls_chacha_encrypt_mode_t

The ChaCha operation:

- Encrypt
- Decrypt

.

Enumerator:

Enum	Description
`CC_CHACHA_Encrypt`	A ChaCha encrypt operation.
`CC_CHACHA_Decrypt`	A ChaCha decrypt operation.
`CC_CHACHA_EncryptNumOfOptions`	The maximal number of encrypt or decrypt operations for the ChaCha engine.
`CC_CHACHA_EncryptModeLast`	Reserved.

enum mbedtls_chacha_nonce_size_t

The allowed nonce-size values of the ChaCha engine in bits.

Enumerator:

Enum	Description
`CC_CHACHA_Nonce64BitSize`	A 64-bit nonce size.
`CC_CHACHA_Nonce96BitSize`	A 96-bit nonce size.
`CC_CHACHA_NonceSizeNumOfOptions`	The maximal number of nonce sizes for the ChaCha engine.
`CC_CHACHA_NonceSizeLast`	Reserved.

Function documentation

CIMPORT_C CCError_t mbedtls_chacha (mbedtls_chacha_nonce pNonce, mbedtls_chacha_nonce_size_t nonceSize, mbedtls_chacha_key pKey, uint32_t initialCounter, mbedtls_chacha_encrypt_mode_t encryptDecryptFlag, uint8_t * pDataIn, size_t dataInSize, uint8_t * pDataOut)

This function performs the ChaCha operation in one integrated process.

Returns:

CC_OK on success.

A non-zero value on failure as defined in *mbedtls_cc_chacha_error.h*.

Parameters:

I/O	Parameter	Description
in	`pNonce`	A buffer containing a nonce.
in	`nonceSize`	An enumerator defining the size of the nonce. Valid values are: • 64bit • 96bit
in	`pKey`	A pointer to the key buffer of the user.
in	`initialCounter`	An initial counter.
in	`encryptDecryptFlag`	A flag specifying which operation the ChaCha engine should perform: encrypt or decrypt.
in	`pDataIn`	A pointer to the buffer of the input-data to the ChaCha engine. The pointer does not need to be aligned. Must not be null.
in	`dataInSize`	The size of the input data. Must not be zero.
out	`pDataOut`	A pointer to the buffer of the output data from the ChaCha. The pointer does not need to be aligned. Must not be null.

CIMPORT_C CCError_t mbedtls_chacha_block (mbedtls_chacha_user_context * pContextID, uint8_t * pDataIn, size_t dataInSize, uint8_t * pDataOut)

This function processes aligned blocks of the ChaCha engine.

The data-in size should be a multiple of the ChaCha block size.

Returns:

CC_OK on success.

A non-zero value on failure as defined in *mbedtls_cc_chacha_error.h*.

Parameters:

I/O	Parameter	Description
in	`pContextID`	A pointer to the context buffer.
in	`pDataIn`	A pointer to the buffer of the input data to the ChaCha engine. The pointer does not need to be aligned. Must not be null.
in	`dataInSize`	The size of the input data. Must be a multiple of *CC_CHACHA_BLOCK_SIZE_IN_BYTES* Bytes, and must not be zero.
out	`pDataOut`	A pointer to the buffer of the output data from the ChaCha engine. The pointer does not need to be aligned. Must not be null.

CIMPORT_C CCError_t mbedtls_chacha_finish (*mbedtls_chacha_user_context* * pContextID, uint8_t * pDataIn, size_t dataInSize, uint8_t * pDataOut)

This function processes the remaining ChaCha data.

The data-in size should be smaller than the ChaCha block size.

Returns:

CC_OK on success.

A non-zero value on failure as defined in *mbedtls_cc_chacha_error.h*.

Parameters:

I/O	Parameter	Description
in	`pContextID`	A pointer to the context buffer.
in	`pDataIn`	A pointer to the buffer of the input data to the ChaCha engine. The pointer does not need to be aligned. If dataInSize = 0, an input buffer is not required.
in	`dataInSize`	The size of the input data. Valid values are: • Zero • Values that are not multiples of *CC_CHACHA_BLOCK_SIZE_IN_BYTES*.
out	`pDataOut`	A pointer to the buffer of the output data from the ChaCha engine. The pointer does not need to be aligned. If dataInSize = 0, an output buffer is not required.

CIMPORT_C CCError_t mbedtls_chacha_free (*mbedtls_chacha_user_context* * pContextID)

This function frees the context used for ChaCha operations.

Returns:

CC_OK on success.

A non-zero value on failure as defined in *mbedtls_cc_chacha_error.h*.

Parameters:

I/O	Parameter	Description
in	`pContextID`	A pointer to the context buffer.

CIMPORT_C CCError_t mbedtls_chacha_init (*mbedtls_chacha_user_context* * pContextID, *mbedtls_chacha_nonce* pNonce, *mbedtls_chacha_nonce_size_t* nonceSize, *mbedtls_chacha_key* pKey, uint32_t initialCounter, *mbedtls_chacha_encrypt_mode_t* EncryptDecryptFlag)

This function initializes the context for ChaCha-engine operations.

Returns:

CC_OK on success.

A non-zero value on failure as defined in *mbedtls_cc_chacha_error.h*.

Parameters:

I/O	Parameter	Description
in	`pContextID`	A pointer to the ChaCha context buffer that is allocated by the user and used for the ChaCha operation.
in	`pNonce`	A buffer containing a nonce.
in	`nonceSize`	An enumerator defining the nonce size. Valid values are: ● 64bit ● 96bit
in	`pKey`	A pointer to the key buffer of the user.
in	`initialCounter`	An initial counter.
in	`EncryptDecryptFlag`	A flag specifying whether the ChaCha engine should perform an Encrypt operation or a Decrypt operation.

Specific errors of the CryptoCell ChaCha APIs

Contains the CryptoCell ChaCha-API error definitions.

Macros

- #define *CC_CHACHA_INVALID_NONCE_ERROR* (*CC_CHACHA_MODULE_ERROR_BASE* + 0x01UL)
- #define *CC_CHACHA_ILLEGAL_KEY_SIZE_ERROR* (*CC_CHACHA_MODULE_ERROR_BASE* + 0x02UL)
- #define *CC_CHACHA_INVALID_KEY_POINTER_ERROR* (*CC_CHACHA_MODULE_ERROR_BASE* + 0x03UL)
- #define *CC_CHACHA_INVALID_ENCRYPT_MODE_ERROR* (*CC_CHACHA_MODULE_ERROR_BASE* + 0x04UL)
- #define *CC_CHACHA_DATA_IN_POINTER_INVALID_ERROR* (*CC_CHACHA_MODULE_ERROR_BASE* + 0x05UL)
- #define *CC_CHACHA_DATA_OUT_POINTER_INVALID_ERROR* (*CC_CHACHA_MODULE_ERROR_BASE* + 0x06UL)

- #define CC_CHACHA_INVALID_USER_CONTEXT_POINTER_ERROR (CC_CHACHA_MODULE_ERROR_BASE + 0x07UL)
- #define CC_CHACHA_CTX_SIZES_ERROR (CC_CHACHA_MODULE_ERROR_BASE + 0x08UL)
- #define CC_CHACHA_INVALID_NONCE_PTR_ERROR (CC_CHACHA_MODULE_ERROR_BASE + 0x09UL)
- #define CC_CHACHA_DATA_IN_SIZE_ILLEGAL (CC_CHACHA_MODULE_ERROR_BASE + 0x0AUL)
- #define CC_CHACHA_GENERAL_ERROR (CC_CHACHA_MODULE_ERROR_BASE + 0x0BUL)
- #define CC_CHACHA_IS_NOT_SUPPORTED (CC_CHACHA_MODULE_ERROR_BASE + 0xFFUL)

Detailed description

Contains the CryptoCell ChaCha-API error definitions.

Macro definition documentation

#define CC_CHACHA_CTX_SIZES_ERROR (CC_CHACHA_MODULE_ERROR_BASE + 0x08UL)

Illegal user context size.

#define CC_CHACHA_DATA_IN_POINTER_INVALID_ERROR (CC_CHACHA_MODULE_ERROR_BASE + 0x05UL)

Illegal data-in pointer.

#define CC_CHACHA_DATA_IN_SIZE_ILLEGAL (CC_CHACHA_MODULE_ERROR_BASE + 0x0AUL)

Illegal data-in size.

#define CC_CHACHA_DATA_OUT_POINTER_INVALID_ERROR (CC_CHACHA_MODULE_ERROR_BASE + 0x06UL)

Illegal data-out pointer.

#define CC_CHACHA_GENERAL_ERROR (CC_CHACHA_MODULE_ERROR_BASE + 0x0BUL)

General error.

#define CC_CHACHA_ILLEGAL_KEY_SIZE_ERROR (CC_CHACHA_MODULE_ERROR_BASE + 0x02UL)

Illegal key size.

#define CC_CHACHA_INVALID_ENCRYPT_MODE_ERROR (CC_CHACHA_MODULE_ERROR_BASE + 0x04UL)

Illegal operation mode.

#define CC_CHACHA_INVALID_KEY_POINTER_ERROR (*CC_CHACHA_MODULE_ERROR_BASE* + 0x03UL)

Illegal key pointer.

#define CC_CHACHA_INVALID_NONCE_ERROR (*CC_CHACHA_MODULE_ERROR_BASE* + 0x01UL)

Illegal Nonce.

#define CC_CHACHA_INVALID_NONCE_PTR_ERROR (*CC_CHACHA_MODULE_ERROR_BASE* + 0x09UL)

Illegal nonce pointer.

#define CC_CHACHA_INVALID_USER_CONTEXT_POINTER_ERROR (*CC_CHACHA_MODULE_ERROR_BASE* + 0x07UL)

Illegal user context.

#define CC_CHACHA_IS_NOT_SUPPORTED (*CC_CHACHA_MODULE_ERROR_BASE* + 0xFFUL)

ChaCha is not supported.

1.5.9 CryptoCell ChaCha-POLY APIs

Contains CryptoCell ChaCha-POLY APIs.

Modules

- *Specific errors of the CryptoCell ChaCha-POLY APIs*

Contains the CryptoCell ChaCha-POLY-API errors definitions. Functions

- CIMPORT_C CCError_t *mbedtls_chacha_poly* (*mbedtls_chacha_nonce* pNonce, *mbedtls_chacha_key* pKey, *mbedtls_chacha_encrypt_mode_t* encryptDecryptFlag, uint8_t *pAddData, size_t addDataSize, uint8_t *pDataIn, size_t dataInSize, uint8_t *pDataOut, *mbedtls_poly_mac* macRes)

  This function performs the ChaCha-POLY encryption and authentication operation.

Detailed description

Contains CryptoCell ChaCha-POLY APIs.

Function documentation

CIMPORT_C CCError_t mbedtls_chacha_poly (*mbedtls_chacha_nonce* pNonce, *mbedtls_chacha_key* pKey, *mbedtls_chacha_encrypt_mode_t* encryptDecryptFlag, uint8_t * pAddData, size_t addDataSize, uint8_t * pDataIn, size_t dataInSize, uint8_t * pDataOut, *mbedtls_poly_mac* macRes)

This function performs the ChaCha-POLY encryption and authentication operation.

Returns:

CC_OK on success.

A non-zero value on failure as defined in *mbedtls_cc_chacha_poly_error.h*.

Parameters:

I/O	Parameter	Description
in	`pNonce`	A pointer to a buffer containing the nonce value.
in	`pKey`	A pointer to the key buffer of the user.
in	`encryptDecryptFlag`	A flag specifying which operation the ChaCha-POLY module should perform: encrypt or decrypt.
in	`pAddData`	A pointer to the additional data input buffer to the POLY module. This pointer does not need to be aligned. Must not be null.
in	`addDataSize`	The size of the input data. Must not be zero.
in	`pDataIn`	A pointer to the input-data buffer to the ChaCha engine. This pointer does not need to be aligned. Must not be null.
in	`dataInSize`	The size of the input data. Must not be zero.
out	`pDataOut`	A pointer to the output-data buffer from the ChaCha engine. This pointer does not need to be aligned. Must not be null.
in,out	`macRes`	A pointer to the MAC result buffer.

Specific errors of the CryptoCell ChaCha-POLY APIs

Contains the CryptoCell ChaCha-POLY-API errors definitions.

Macros

- #define *CC_CHACHA_POLY_ADATA_INVALID_ERROR* (*CC_CHACHA_POLY_MODULE_ERROR_BASE* + 0x01UL)
- #define *CC_CHACHA_POLY_DATA_INVALID_ERROR* (*CC_CHACHA_POLY_MODULE_ERROR_BASE* + 0x02UL)
- #define *CC_CHACHA_POLY_ENC_MODE_INVALID_ERROR* (*CC_CHACHA_POLY_MODULE_ERROR_BASE* + 0x03UL)
- #define *CC_CHACHA_POLY_DATA_SIZE_INVALID_ERROR* (*CC_CHACHA_POLY_MODULE_ERROR_BASE* + 0x04UL)
- #define *CC_CHACHA_POLY_GEN_KEY_ERROR* (*CC_CHACHA_POLY_MODULE_ERROR_BASE* + 0x05UL)
- #define *CC_CHACHA_POLY_ENCRYPTION_ERROR* (*CC_CHACHA_POLY_MODULE_ERROR_BASE* + 0x06UL)
- #define *CC_CHACHA_POLY_AUTH_ERROR* (*CC_CHACHA_POLY_MODULE_ERROR_BASE* + 0x07UL)
- #define *CC_CHACHA_POLY_MAC_ERROR* (*CC_CHACHA_POLY_MODULE_ERROR_BASE* + 0x08UL)

Detailed description

Contains the CryptoCell ChaCha-POLY-API errors definitions.

Macro definition documentation

#define CC_CHACHA_POLY_ADATA_INVALID_ERROR (CC_CHACHA_POLY_MODULE_ERROR_BASE + 0x01UL)

Invalid additional data.

#define CC_CHACHA_POLY_AUTH_ERROR (CC_CHACHA_POLY_MODULE_ERROR_BASE + 0x07UL)

Authentication error.

#define CC_CHACHA_POLY_DATA_INVALID_ERROR (CC_CHACHA_POLY_MODULE_ERROR_BASE + 0x02UL)

Invalid input data.

#define CC_CHACHA_POLY_DATA_SIZE_INVALID_ERROR (CC_CHACHA_POLY_MODULE_ERROR_BASE + 0x04UL)

Illegal data size.

#define CC_CHACHA_POLY_ENC_MODE_INVALID_ERROR (CC_CHACHA_POLY_MODULE_ERROR_BASE + 0x03UL)

Illegal encryption mode.

#define CC_CHACHA_POLY_ENCRYPTION_ERROR (CC_CHACHA_POLY_MODULE_ERROR_BASE + 0x06UL)

ChaCha key-generation error.

#define CC_CHACHA_POLY_GEN_KEY_ERROR (CC_CHACHA_POLY_MODULE_ERROR_BASE + 0x05UL)

Key-generation error.

#define CC_CHACHA_POLY_MAC_ERROR (CC_CHACHA_POLY_MODULE_ERROR_BASE + 0x08UL)

MAC comparison error.

1.5.10 CryptoCell DHM APIs

Diffie-Hellman-Merkle (DHM) is used to securely exchange cryptographic keys over a public channel.

CryptoCell-312 hardware limitations for DHM

To support the accelerated algorithms, the following conditions must be met:

- The contexts must be DMA-able, as they might be used for some symmetric operations.

Typical usage of DHM in CryptoCell-312

The following is a typical DHM flow for one party:

1. *mbedtls_dhm_init()*.
2. mbedtls_mpi_read_string().
3. mbedtls_mpi_read_string().
4. *mbedtls_dhm_make_params()*.
5. *mbedtls_dhm_read_public()*.
6. *mbedtls_dhm_calc_secret()*.

Modules

- *CryptoCell-312 hardware limitations for DHM*
- *Typical usage of DHM in CryptoCell-312*

Detailed description

Diffie-Hellman-Merkle (DHM) is used to securely exchange cryptographic keys over a public channel.

As described in Public-Key Cryptography Standards (PKCS) #3: Diffie Hellman Key Agreement Standard : "[T]wo parties, without any prior arrangements, can agree upon a secret key that is known only to them...This secret key can then be used, for example, to encrypt further communications between the parties."

The DHM module is implemented based on the definitions in the following standards:

- RFC-3526: More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE) : defines a number of standardized Diffie-Hellman groups for IKE.
- RFC-5114: Additional Diffie-Hellman Groups for Use with IETF Standards : defines a number of standardized Diffie-Hellman groups that can be used.

For the implementation of DHM, see *dhm.h*.

1.5.11 CryptoCell Elliptic Curve APIs

Contains all CryptoCell Elliptic Curve APIs.

Modules

- *CryptoCell ECIES APIs*

  Contains the CryptoCell Elliptic Curve Integrated Encryption Scheme (ECIES) APIs.
- *CryptoCell ECPKI APIs*

  Contains all CryptoCell ECPKI APIs.
- *ECDH module overview*

  Elliptic-curve Diffie–Hellman (ECDH) is an anonymous key agreement protocol. It allows two parties to establish a shared secret over an insecure channel. Each party must have an elliptic-curve public–private key pair.
- *ECDSA module overview*

The Elliptic Curve Digital Signature Algorithm (ECDSA) is a used for generating and validating digital signatures.

Detailed description

Contains all CryptoCell Elliptic Curve APIs.

CryptoCell ECIES APIs

Contains the CryptoCell Elliptic Curve Integrated Encryption Scheme (ECIES) APIs.

Macros

- #define *MBEDTLS_ECIES_MAX_CIPHER_LEN_BYTES* ((2**CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS* + 1) * sizeof(int))
- #define *MBEDTLS_ECIES_MIN_BUFF_LEN_BYTES* (sizeof(*CCEciesTempData_t*))
- #define *mbedtls_ecies_kem_encrypt*(pGrp, pRecipPublKey, kdfDerivMode, kdfHashMode, isSingleHashMode, pSecrKey, secrKeySize, pCipherData, pCipherDataSize, pBuff, buffLen, f_rng, p_rng)

  A macro for creating and encrypting a secret key.

Functions

- CCError_t *mbedtls_ecies_kem_encrypt_full* (*mbedtls_ecp_group* *pGrp, *mbedtls_ecp_point* *pRecipUzPublKey, CCKdfDerivFuncMode_t kdfDerivMode, *mbedtls_hkdf_hashmode_t* kdfHashMode, uint32_t isSingleHashMode, *mbedtls_ecp_point* *pExtEphUzPublicKey, mbedtls_mpi *pExtEphUzPrivateKey, uint8_t *pSecrKey, size_t secrKeySize, uint8_t *pCipherData, size_t *pCipherDataSize, void *pBuff, size_t buffLen, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  This function creates and encrypts (encapsulates) the secret key of required size, according to ISO/IEC 18033-2:2006: Information technology – Security techniques – Encryption algorithms – Part 2: Asymmetric ciphers , ECIES-KEM Encryption.

- CCError_t *mbedtls_ecies_kem_decrypt* (*mbedtls_ecp_group* *pGrp, mbedtls_mpi *pRecipUzPrivKey, CCKdfDerivFuncMode_t kdfDerivMode, *mbedtls_hkdf_hashmode_t* kdfHashMode, uint32_t isSingleHashMode, uint8_t *pCipherData, size_t cipherDataSize, uint8_t *pSecrKey, size_t secrKeySize, void *pBuff, size_t buffLen)

  This function decrypts the encapsulated secret key passed by the sender, according to ISO/IEC 18033-2:2006: Information technology – Security techniques – Encryption algorithms – Part 2: Asymmetric ciphers , sec. 10.2.4 - ECIES-KEM Decryption.

Detailed description

Contains the CryptoCell Elliptic Curve Integrated Encryption Scheme (ECIES) APIs.

Macro definition documentation

#define mbedtls_ecies_kem_encrypt(pGrp, pRecipPublKey, kdfDerivMode, kdfHashMode, isSingleHashMode, pSecrKey, secrKeySize, pCipherData, pCipherDataSize, pBuff, buffLen, f_rng, p_rng)

```
Value:mbedtls_ecies_kem_encrypt_full((pGrp), (pRecipPublKey),
(kdfDerivMode), (kdfHashMode), \
                                   (isSingleHashMode), NULL, NULL,
(pSecrKey), (secrKeySize), \
                                        (pCipherData), (pCipherDataSize),
(pBuff), (buffLen), \
                                        f_rng, p_rng)
```

A macro for creating and encrypting a secret key.

For a description of the parameters see *mbedtls_ecies_kem_encrypt_full*.

#define MBEDTLS_ECIES_MAX_CIPHER_LEN_BYTES ((2*CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS + 1) * sizeof(int))

The maximal length of the ECIES cipher in Bytes.

#define MBEDTLS_ECIES_MIN_BUFF_LEN_BYTES (sizeof(CCEciesTempData_t))

The minimal length of the ECIES buffer in Bytes.

Function documentation

CCError_t mbedtls_ecies_kem_decrypt (mbedtls_ecp_group * pGrp, mbedtls_mpi * pRecipUzPrivKey, CCKdfDerivFuncMode_t kdfDerivMode, mbedtls_hkdf_hashmode_t kdfHashMode, uint32_t isSingleHashMode, uint8_t * pCipherData, size_t cipherDataSize, uint8_t * pSecrKey, size_t secrKeySize, void * pBuff, size_t buffLen)

This function decrypts the encapsulated secret key passed by the sender, according to ISO/IEC 18033-2:2006: Information technology – Security techniques – Encryption algorithms – Part 2: Asymmetric ciphers , sec. 10.2.4 - ECIES-KEM Decryption.

Note

The KDF2 function mode must be used for compliance with X9.63-2011: Public Key Cryptography for the Financial Services Industry – Key Agreement and Key Transport Using Elliptic Curve Cryptograph.

- The term "sender" indicates an entity that creates and encapsulates the secret key using this function. The term "recipient" indicates another entity which receives and decrypts the secret key.
- All public and private keys that are used must relate to the same EC Domain.

Returns:

CCError_t 0 if successful.

Parameters:

I/O	Parameter	Description
in	`pGrp`	The ECP group to use.
in	`pRecipUzPrivKey`	A pointer to the private key of the recipient.
in	`kdfDerivMode`	The KDF function mode to use: KDF1 or KDF2. For more information, see CCKdfDerivFuncMode_t() in cc_kdf.h.
in	`kdfHashMode`	The used hash function.
in	`isSingleHashMode`	The specific ECIES mode definition: 0,1, according to ISO/IEC 18033-2:2006: Information technology – Security techniques – Encryption algorithms – Part 2: Asymmetric ciphers - section 10.2.
in	`pCipherData`	A pointer to the received encrypted cipher data.
in	`cipherDataSize`	The size of the cipher data in Bytes.

I/O	Parameter	Description
in	`pSecrKey`	A pointer to the buffer for the secret-key data to be generated.
in	`secrKeySize`	The size of the secret-key data in Bytes.
in	`pBuff`	A pointer to the temporary buffer.
in	`buffLen`	The size of the buffer pointed by pBuff. Must not be less than *MBEDTLS_ECIES_MIN_BUFF_LEN_BYTES*.

CCError_t mbedtls_ecies_kem_encrypt_full (*mbedtls_ecp_group* * pGrp, *mbedtls_ecp_point* * pRecipUzPublKey, CCKdfDerivFuncMode_t kdfDerivMode, *mbedtls_hkdf_hashmode_t* kdfHashMode, uint32_t isSingleHashMode, *mbedtls_ecp_point* * pExtEphUzPublicKey, mbedtls_mpi * pExtEphUzPrivateKey, uint8_t * pSecrKey, size_t secrKeySize, uint8_t * pCipherData, size_t * pCipherDataSize, void * pBuff, size_t buffLen, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

This function creates and encrypts (encapsulates) the secret key of required size, according to ISO/IEC 18033-2:2006: Information technology – Security techniques – Encryption algorithms – Part 2: Asymmetric ciphers , ECIES-KEM Encryption.

To call this function in applications, the *mbedtls_ecies_kem_encrypt* macro definition must be used. The function itself has the additional input of the external ephemeral key pair, used only for testing purposes.

Note

- Use KDF2 function mode for compliance with X9.63-2011: Public Key Cryptography for the Financial Services Industry – Key Agreement and Key Transport Using Elliptic Curve Cryptography.
- The term "sender" indicates an entity that creates and encapsulates the secret key using this function. The term "recipient" indicates another entity which receives and decrypts the secret key.
- All public and private keys that are used must relate to the same EC Domain.
- The user must verify that the public key of the recipient is on the elliptic curve before it is used in this function.

Returns:

CCError_t 0 if successful.

Parameters:

I/O	Parameter	Description
in	`pGrp`	The ECP group to use.
in	`pRecipUzPublKey`	A pointer to the public key of the recipient.
in	`kdfDerivMode`	The KDF function mode to use: KDF1 or KDF2. For more information, see CCKdfDerivFuncMode_t() in cc_kdf.h.
in	`kdfHashMode`	The used hash function.
in	`isSingleHashMode`	The specific ECIES mode, according to ISO/IEC 18033-2:2006: Information technology – Security techniques – Encryption algorithms – Part 2: Asymmetric ciphers - section 10.2: ● 0: Not-single hash. ● 1: Single hash.

I/O	Parameter	Description
in	`pExtEphUzPublicKey`	A pointer to the ephemeral public key related to the private key. Must be set to NULL if pExtEphUzPrivateKey = NULL.
in	`pExtEphUzPrivateKey`	The pointer to the external ephemeral private key. This key is used only for testing the function. In regular use, the pointer should be set to NULL and then the random key-pair should be generated internally.
in	`pSecrKey`	A pointer to the buffer for the secret-key data to be generated.
in	`secrKeySize`	The size of the secret-key data in Bytes.
in	`pCipherData`	A pointer to the encrypted cipher text.
in,out	`pCipherDataSize`	• In: A pointer to the size of the buffer for output of the CipherData. • Out: The size of the buffer for output of the CipherData in Bytes.
in	`pBuff`	A pointer to the temporary buffer.
in	`buffLen`	The size of the buffer pointed by pBuff. Must not be less than *MBEDTLS_ECIES_MIN_BUFF_LEN_BYTES*.
in	`f_rng`	The RNG function required for generating a key pair when pExtEphUzPublicKey and pExtEphUzPrivateKey are NULL
in	`p_rng`	The RNG parameter.

CryptoCell ECPKI APIs

Contains all CryptoCell ECPKI APIs.

Modules

- *CryptoCell ECPKI type definitions*
- Contains CryptoCell ECPKI API type definitions. *CryptoCell ECPKI supported domains*

Contains CryptoCell ECPKI domains supported by the project.

Detailed description

Contains all CryptoCell ECPKI APIs.

CryptoCell ECPKI supported domains

Contains CryptoCell ECPKI domains supported by the project.

Typedefs

- typedef const *CCEcpkiDomain_t* *(* *getDomainFuncP*) (void)

Detailed description

Contains CryptoCell ECPKI domains supported by the project.

Typedef documentation

typedef const *CCEcpkiDomain_t**(* getDomainFuncP) (void)

Definition of the domain-retrieval function.

ECDH module overview

Elliptic-curve Diffie–Hellman (ECDH) is an anonymous key agreement protocol. It allows two parties to establish a shared secret over an insecure channel. Each party must have an elliptic-curve public–private key pair.

CryptoCell-312 hardware limitations for ECDH

CryotoCell-312 does not support Brainpool curves.

Typical usage of ECDH in CryptoCell-312

The following is a typical ECDH operation flow:

1. *mbedtls_ecp_group_init()*.
2. mbedtls_mpi_init() for each group parameter.
3. *mbedtls_ecdh_gen_public()*.

Modules

- *CryptoCell-312 hardware limitations for ECDH*
- *Typical usage of ECDH in CryptoCell-312*

Detailed description

Elliptic-curve Diffie–Hellman (ECDH) is an anonymous key agreement protocol. It allows two parties to establish a shared secret over an insecure channel. Each party must have an elliptic-curve public–private key pair.

For more information, see NIST SP 800-56A Rev. 2: Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography.

For the implementation of ECDH, see *ecdh.h*.

ECDSA module overview

The Elliptic Curve Digital Signature Algorithm (ECDSA) is a used for generating and validating digital signatures.

CryptoCell-312 hardware limitations for ECDSA

CryotoCell-312 does not support Brainpool curves.

Note

Using hash functions with hash size greater than the EC modulus size is not recommended.

Typical usage of ECDSA in CryptoCell-312

The following is a typical ECDSA operation flow:

1. *mbedtls_ecp_group_init()*.
2. mbedtls_mpi_init() for each group parameter.
3. *mbedtls_ecp_gen_keypair()*.
4. *mbedtls_ecdsa_sign()* or *mbedtls_ecdsa_verify()*.

Modules

- *CryptoCell-312 hardware limitations for ECDSA*
- *Typical usage of ECDSA in CryptoCell-312*

Detailed description

The Elliptic Curve Digital Signature Algorithm (ECDSA) is a used for generating and validating digital signatures.

For the definition of ECDSA, see Standards for Efficient Cryptography Group (SECG): SEC1 Elliptic Curve Cryptography.

For the use of ECDSA for TLS, see RFC-4492: Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS).

For the implementation of ECDSA, see *ecdsa.h*.

1.5.12 CryptoCell external DMA APIs

Contains all CryptoCell external DMA API definitions.

Modules

- *CryptoCell AES external DMA APIs*
- Contains CryptoCell AES external DMA API definitions. *CryptoCell ChaCha external DMA APIs*
- Contains CryptoCell ChaCha external DMA APIs. *CryptoCell hash external DMA APIs*
- Contains CryptoCell hash external DMA APIs. *Specific errors of the CryptoCell external DMA APIs*

Contains the CryptoCell external DMA-API error definitions.

Detailed description

Contains all CryptoCell external DMA API definitions.

CryptoCell AES external DMA APIs

Contains CryptoCell AES external DMA API definitions.

Functions

- int *mbedtls_aes_ext_dma_init* (unsigned int keybits, int encryptDecryptFlag, *CCAesOperationMode_t* operationMode, size_t data_size)

  This function initializes the external DMA Control. It configures the AES mode, the direction(encryption or decryption), and the data size.

- int *mbedtls_aes_ext_dma_set_key* (const unsigned char *key, unsigned int keybits)

  This function configures the key.

- int *mbedtls_aes_ext_dma_set_iv* (*CCAesOperationMode_t* operationMode, unsigned char *iv, unsigned int iv_size)

  This function configures the IV.

- int *mbedtls_aes_ext_dma_finish* (*CCAesOperationMode_t* operationMode, unsigned char *iv, unsigned int iv_size)

  This function returns the IV after an AES CMAC or a CBCMAC operation.

Detailed description

Contains CryptoCell AES external DMA API definitions.

The supported AES modes for external DMA mode are:

- ECB
- CBC
- CTR
- CBC_MAC
- CMAC
- OFB

Function documentation

int mbedtls_aes_ext_dma_finish (CCAesOperationMode_t operationMode, unsigned char * iv, unsigned int iv_size)

This function returns the IV after an AES CMAC or a CBCMAC operation.

Returns:

CC_OK on success.

A non-zero value from cc_aes_error.h on failure.

Parameters:

I/O	Parameter	Description
in	`operationMode`	The AES mode. See the *Detailed description* in mbedtls_aes_ext_dma.h File Reference.
out	`iv`	The AES IV buffer.
in	`iv_size`	The size of the IV. Must be 16 Bytes.

int mbedtls_aes_ext_dma_init (unsigned int keybits, int encryptDecryptFlag, CCAesOperationMode_t operationMode, size_t data_size)

This function initializes the external DMA Control. It configures the AES mode, the direction (encryption or decryption), and the data size.

Returns:

CC_OK on success.

A non-zero value from cc_aes_error.h on failure.

Parameters:

I/O	Parameter	Description
in	`keybits`	AES key size. Valid values are: • 128bits • 192bits • 256bits
in	`encryptDecryptFlag`	• 0: Encrypt. • 1: Decrypt
in	`operationMode`	AES mode. See the *Detailed description* in mbedtls_aes_ext_dma.h File Reference.

I/O	Parameter	Description
in	`data_size`	Data size to encrypt or decrypt.

int mbedtls_aes_ext_dma_set_iv (*CCAesOperationMode_t* operationMode, unsigned char * iv, unsigned int iv_size)

This function configures the IV.

Returns:

CC_OK on success.

A non-zero value from cc_aes_error.h on failure.

Parameters:

I/O	Parameter	Description
in	`operationMode`	AES mode. See the *Detailed description* in *mbedtls_aes_ext_dma.h File Reference*.
in	`iv`	The AES IV buffer.
in	`iv_size`	The size of the IV. Must be 16 Bytes.

int mbedtls_aes_ext_dma_set_key (const unsigned char * key, unsigned int keybits)

This function configures the key.

Returns:

CC_OK on success.

A non-zero value from cc_aes_error.h on failure.

Parameters:

I/O	Parameter	Description
in	`key`	The AES key buffer.
in	`keybits`	The size of the AES Key. Valid values are: • 128bits • 192bits • 256bits

CryptoCell ChaCha external DMA APIs

Contains CryptoCell ChaCha external DMA APIs.

Functions

- int *mbedtls_ext_dma_chacha_init* (uint8_t *pNonce, *mbedtls_chacha_nonce_size_t* nonceSizeFlag, uint8_t *pKey, uint32_t keySizeBytes, uint32_t initialCounter, *mbedtls_chacha_encrypt_mode_t* EncryptDecryptFlag)

  This function initializes the external DMA control. It configures the ChaCha mode, the initial hash value, and other configurations in the ChaCha engine.

- int *mbedtls_chacha_ext_dma_finish* (void)

This function frees used resources.

Detailed description

Contains CryptoCell ChaCha external DMA APIs.

Function documentation

int mbedtls_chacha_ext_dma_finish (void)

This function frees used resources.

Returns:

CC_OK on success.

A non-zero value from *mbedtls_ext_dma_error.h* on failure.

int mbedtls_ext_dma_chacha_init (uint8_t * pNonce, *mbedtls_chacha_nonce_size_t* nonceSizeFlag, uint8_t * pKey, uint32_t keySizeBytes, uint32_t initialCounter, *mbedtls_chacha_encrypt_mode_t* EncryptDecryptFlag)

This function initializes the external DMA control. It configures the ChaCha mode, the initial hash value, and other configurations in the ChaCha engine.

Returns:

0 on success.

A non-zero value from *mbedtls_ext_dma_error.h* on failure.

Parameters:

I/O	Parameter	Description
in	`pNonce`	The nonce buffer.
in	`nonceSizeFlag`	The nonce size flag.
in	`pKey`	The key buffer.
in	`keySizeBytes`	The size of the key buffer. Must be 32 Bytes.
in	`initialCounter`	Intial counter value.
in	`EncryptDecryptFlag`	The ChaCha operation: • Encrypt • Decrypt

CryptoCell hash external DMA APIs

Contains CryptoCell hash external DMA APIs.

Functions

- int *mbedtls_hash_ext_dma_init* (*CCHashOperationMode_t* operationMode)

  This function initializes the External DMA Control.

- int *mbedtls_hash_ext_dma_finish* (*CCHashOperationMode_t* operationMode, uint32_t digestBufferSize, uint32_t *digestBuffer)

  This function returns the digest after the hash operation, and frees used resources.

Detailed description

Contains CryptoCell hash external DMA APIs.

Function documentation

int mbedtls_hash_ext_dma_finish (CCHashOperationMode_t operationMode, uint32_t digestBufferSize, uint32_t * digestBuffer)

This function returns the digest after the hash operation, and frees used resources.

Returns:

CC_OK on success.

A non-zero value from cc_hash_error.h on failure.

Parameters:

I/O	Parameter	Description
in	`operationMode`	The hash mode. Supported modes are: • SHA1 • SHA224 • SHA256
in	`digestBufferSize`	The size of the hash digest in Bytes.
out	`digestBuffer`	The output digest buffer.

int mbedtls_hash_ext_dma_init (CCHashOperationMode_t operationMode)

This function initializes the External DMA Control.

It configures the hash mode, hash initial value and other configurations.

Returns:

CC_OK on success.

A non-zero value from cc_hash_error.h on failure.

Parameters:

I/O	Parameter	Description
in	`operationMode`	The hash mode. Supported modes are: • SHA1 • SHA224 • SHA256

Specific errors of the CryptoCell external DMA APIs

Contains the CryptoCell external DMA-API error definitions.

Macros

- #define *EXT_DMA_AES_ILLEGAL_OPERATION_MODE_ERROR* (*CC_EXT_DMA_MODULE_ERROR_BASE* + 0x00UL)

- #define *EXT_DMA_AES_INVALID_ENCRYPT_MODE_ERROR* (*CC_EXT_DMA_MODULE_ERROR_BASE* + 0x01UL)
- #define *EXT_DMA_AES_DECRYPTION_NOT_ALLOWED_ON_THIS_MODE* (*CC_EXT_DMA_MODULE_ERROR_BASE* + 0x02UL)
- #define *EXT_DMA_AES_ILLEGAL_KEY_SIZE_ERROR* (*CC_EXT_DMA_MODULE_ERROR_BASE* + 0x03UL)
- #define *EXT_DMA_AES_INVALID_IV_OR_TWEAK_PTR_ERROR* (*CC_EXT_DMA_MODULE_ERROR_BASE* + 0x04UL)
- #define *EXT_DMA_HASH_ILLEGAL_OPERATION_MODE_ERROR* (*CC_EXT_DMA_MODULE_ERROR_BASE* + 0x05UL)
- #define *EXT_DMA_HASH_INVALID_RESULT_BUFFER_POINTER_ERROR* (*CC_EXT_DMA_MODULE_ERROR_BASE* + 0x06UL)
- #define *EXT_DMA_HASH_ILLEGAL_PARAMS_ERROR* (*CC_EXT_DMA_MODULE_ERROR_BASE* + 0x07UL)
- #define *EXT_DMA_CHACHA_INVALID_NONCE_PTR_ERROR* (*CC_EXT_DMA_MODULE_ERROR_BASE* + 0x08UL)
- #define *EXT_DMA_CHACHA_INVALID_ENCRYPT_MODE_ERROR* (*CC_EXT_DMA_MODULE_ERROR_BASE* + 0x09UL)
- #define *EXT_DMA_CHACHA_INVALID_KEY_POINTER_ERROR* (*CC_EXT_DMA_MODULE_ERROR_BASE* + 0xAUL)
- #define *EXT_DMA_CHACHA_ILLEGAL_KEY_SIZE_ERROR* (*CC_EXT_DMA_MODULE_ERROR_BASE* + 0xBUL)
- #define *EXT_DMA_CHACHA_INVALID_NONCE_ERROR* (*CC_EXT_DMA_MODULE_ERROR_BASE* + 0xCUL)

Detailed description

Contains the CryptoCell external DMA-API error definitions.

Macro definition documentation

#define EXT_DMA_AES_DECRYPTION_NOT_ALLOWED_ON_THIS_MODE (*CC_EXT_DMA_MODULE_ERROR_BASE* + 0x02UL)

Illegal decryption mode.

#define EXT_DMA_AES_ILLEGAL_KEY_SIZE_ERROR (*CC_EXT_DMA_MODULE_ERROR_BASE* + 0x03UL)

Illegal key size.

#define EXT_DMA_AES_ILLEGAL_OPERATION_MODE_ERROR (CC_EXT_DMA_MODULE_ERROR_BASE + 0x00UL)

Illegal mode.

#define EXT_DMA_AES_INVALID_ENCRYPT_MODE_ERROR (CC_EXT_DMA_MODULE_ERROR_BASE + 0x01UL)

Illegal encryption mode.

#define EXT_DMA_AES_INVALID_IV_OR_TWEAK_PTR_ERROR (CC_EXT_DMA_MODULE_ERROR_BASE + 0x04UL)

Illegal IV.

#define EXT_DMA_CHACHA_ILLEGAL_KEY_SIZE_ERROR (CC_EXT_DMA_MODULE_ERROR_BASE + 0xBUL)

Invalid key size.

#define EXT_DMA_CHACHA_INVALID_ENCRYPT_MODE_ERROR (CC_EXT_DMA_MODULE_ERROR_BASE + 0x09UL)

Invalid encrypt or decrypt mode.

#define EXT_DMA_CHACHA_INVALID_KEY_POINTER_ERROR (CC_EXT_DMA_MODULE_ERROR_BASE + 0xAUL)

Invalid key pointer.

#define EXT_DMA_CHACHA_INVALID_NONCE_ERROR (CC_EXT_DMA_MODULE_ERROR_BASE + 0xCUL)

Invalid nonce size flag.

#define EXT_DMA_CHACHA_INVALID_NONCE_PTR_ERROR (CC_EXT_DMA_MODULE_ERROR_BASE + 0x08UL)

Invalid nonce.

#define EXT_DMA_HASH_ILLEGAL_OPERATION_MODE_ERROR (CC_EXT_DMA_MODULE_ERROR_BASE + 0x05UL)

Illegal hash operation mode.

#define EXT_DMA_HASH_ILLEGAL_PARAMS_ERROR (CC_EXT_DMA_MODULE_ERROR_BASE + 0x07UL)

Illegal parameters.

#define EXT_DMA_HASH_INVALID_RESULT_BUFFER_POINTER_ERROR (CC_EXT_DMA_MODULE_ERROR_BASE + 0x06UL)

Illegal result buffer.

1.5.13 CryptoCell hash APIs

Contains all CryptoCell hash APIs and definitions.

CryptoCell-312 hardware limitations for hash

The CryptoCell-312 hardware supports accelerated hash operations for the following modes:

- SHA-1
- SHA-224
- SHA-256

SHA-384 and SHA-512 operations are only supported in a non-accelerated software mode.

To support the accelerated algorithms, the following conditions must be met:

- The input buffer must be DMA-able.
- The input buffer must be physically contingous block in memory.
- Buffer size must be up to 64KB.
- The context must also be DMA-able, as partial and final results are written to the context.

Typical usage of hash in CryptoCell-312

The following is a typical hash Block operation flow directly using the SHA module:

1. *mbedtls_sha1_init()*.
2. *mbedtls_sha1_starts_ret()*.
3. *mbedtls_sha1_update_ret()*.
4. *mbedtls_sha1_finish_ret()*.

The following is a typical HMAC Block operation flow using the MD module:

1. *mbedtls_md_setup()*.
2. *mbedtls_md_hmac_starts()*.
3. *mbedtls_md_hmac_update()*.
4. *mbedtls_md_hmac_finish()*.

Modules

- *CryptoCell hash API definitions*
- Contains CryptoCell hash-API definitions. *CryptoCell hash-API project-specific definitions*
- Contains the project-specific hash-API definitions. *CryptoCell SHA-512 truncated APIs*
- Contains all CryptoCell SHA-512 truncated APIs. *CryptoCell-312 hardware limitations for hash*
- *Typical usage of hash in CryptoCell-312*

Detailed description

Contains all CryptoCell hash APIs and definitions.

The hash or Message Digest (MD) module allows you to calculate hash digests from data, and create signatures based on those hash digests.

HMAC is a wrapping algorithm that uses one of the supported hash algorithms and a key, to generate a unique authentication code over the input data.

All hash algorithms can be accessed via the generic MD layer. For more information, see *mbedtls_md_setup()*.

For more information on supported hash algorithms,

See also:

CryptoCell-312 hardware limitations for hash.

For the implementation of hash and HMAC, see *md.h*.

CryptoCell hash API definitions

Contains CryptoCell hash-API definitions.

Data structures

- struct *CCHashUserContext_t*

The context prototype of the user. Macros

- #define *CC_HASH_RESULT_SIZE_IN_WORDS* 16
- #define *CC_HASH_MD5_DIGEST_SIZE_IN_BYTES* 16
- #define *CC_HASH_MD5_DIGEST_SIZE_IN_WORDS* 4
- #define *CC_HASH_SHA1_DIGEST_SIZE_IN_BYTES* 20
- #define *CC_HASH_SHA1_DIGEST_SIZE_IN_WORDS* 5
- #define *CC_HASH_SHA224_DIGEST_SIZE_IN_WORDS* 7
- #define *CC_HASH_SHA256_DIGEST_SIZE_IN_WORDS* 8
- #define *CC_HASH_SHA384_DIGEST_SIZE_IN_WORDS* 12
- #define *CC_HASH_SHA512_DIGEST_SIZE_IN_WORDS* 16
- #define *CC_HASH_SHA224_DIGEST_SIZE_IN_BYTES* 28
- #define *CC_HASH_SHA256_DIGEST_SIZE_IN_BYTES* 32
- #define *CC_HASH_SHA384_DIGEST_SIZE_IN_BYTES* 48
- #define *CC_HASH_SHA512_DIGEST_SIZE_IN_BYTES* 64
- #define *CC_HASH_BLOCK_SIZE_IN_WORDS* 16
- #define *CC_HASH_BLOCK_SIZE_IN_BYTES* 64
- #define *CC_HASH_SHA512_BLOCK_SIZE_IN_WORDS* 32
- #define *CC_HASH_SHA512_BLOCK_SIZE_IN_BYTES* 128
- #define *CC_HASH_UPDATE_DATA_MAX_SIZE_IN_BYTES* (1 << 29)

Typedefs

- typedef uint32_t *CCHashResultBuf_t*[*CC_HASH_RESULT_SIZE_IN_WORDS*]
- typedef struct *CCHashUserContext_t* *CCHashUserContext_t*

  The context prototype of the user.

Enumerations

- enum *CCHashOperationMode_t* { *CC_HASH_SHA1_mode* = 0, *CC_HASH_SHA224_mode* = 1, *CC_HASH_SHA256_mode* = 2, *CC_HASH_SHA384_mode* = 3, *CC_HASH_SHA512_mode* = 4, *CC_HASH_MD5_mode* = 5, *CC_HASH_NumOfModes*, *CC_HASH_OperationModeLast* = 0x7FFFFFFF }

Detailed description

Contains CryptoCell hash-API definitions.

Macro definition documentation

#define CC_HASH_BLOCK_SIZE_IN_BYTES 64

The size of the SHA-1 hash block in Bytes.

#define CC_HASH_BLOCK_SIZE_IN_WORDS 16

The size of the SHA-1 hash block in words.

#define CC_HASH_MD5_DIGEST_SIZE_IN_BYTES 16

The size of the MD5 digest result in Bytes.

#define CC_HASH_MD5_DIGEST_SIZE_IN_WORDS 4

The size of the MD5 digest result in words.

#define CC_HASH_RESULT_SIZE_IN_WORDS 16

The size of the hash result in words. The maximal size for SHA-512 is 512 bits.

#define CC_HASH_SHA1_DIGEST_SIZE_IN_BYTES 20

The size of the SHA-1 digest result in Bytes.

#define CC_HASH_SHA1_DIGEST_SIZE_IN_WORDS 5

The size of the SHA-1 digest result in words.

#define CC_HASH_SHA224_DIGEST_SIZE_IN_BYTES 28

The size of the SHA-256 digest result in Bytes.

#define CC_HASH_SHA224_DIGEST_SIZE_IN_WORDS 7

The size of the SHA-224 digest result in words.

#define CC_HASH_SHA256_DIGEST_SIZE_IN_BYTES 32

The size of the SHA-256 digest result in Bytes.

#define CC_HASH_SHA256_DIGEST_SIZE_IN_WORDS 8

The size of the SHA-256 digest result in words.

#define CC_HASH_SHA384_DIGEST_SIZE_IN_BYTES 48

The size of the SHA-384 digest result in Bytes.

#define CC_HASH_SHA384_DIGEST_SIZE_IN_WORDS 12

The size of the SHA-384 digest result in words.

#define CC_HASH_SHA512_BLOCK_SIZE_IN_BYTES 128

The size of the SHA-2 hash block in Bytes.

#define CC_HASH_SHA512_BLOCK_SIZE_IN_WORDS 32

The size of the SHA-2 hash block in words.

#define CC_HASH_SHA512_DIGEST_SIZE_IN_BYTES 64

The size of the SHA-512 digest result in Bytes.

#define CC_HASH_SHA512_DIGEST_SIZE_IN_WORDS 16

The size of the SHA-512 digest result in words.

#define CC_HASH_UPDATE_DATA_MAX_SIZE_IN_BYTES (1 << 29)

The maximal data size for the update operation.

Typedef documentation

typedef uint32_t CCHashResultBuf_t[CC_HASH_RESULT_SIZE_IN_WORDS]

The hash result buffer.

typedef struct CCHashUserContext_t CCHashUserContext_t

The context prototype of the user.

The argument type that is passed by the user to the hash APIs. The context saves the state of the operation, and must be saved by the user until the end of the API flow.

Enumeration type documentation

enum CCHashOperationMode_t

The hash operation mode.

Enumerator:

Enum	Description
`CC_HASH_SHA1_mode`	SHA-1.
`CC_HASH_SHA224_mode`	SHA-224.
`CC_HASH_SHA256_mode`	SHA-256.
`CC_HASH_SHA384_mode`	SHA-384.
`CC_HASH_SHA512_mode`	SHA-512.
`CC_HASH_MD5_mode`	MD5.
`CC_HASH_NumOfModes`	The number of hash modes.
`CC_HASH_OperationModeLast`	Reserved.

CryptoCell hash-API project-specific definitions

Contains the project-specific hash-API definitions.

Macros

- #define CC_HASH_USER_CTX_SIZE_IN_WORDS 60

Detailed description

Contains the project-specific hash-API definitions.

Macro definition documentation

#define CC_HASH_USER_CTX_SIZE_IN_WORDS 60

The size of the context prototype of the user in words. See *CCHashUserContext_t*.

CryptoCell SHA-512 truncated APIs

Contains all CryptoCell SHA-512 truncated APIs.

Functions

- void *mbedtls_sha512_t_init* (*mbedtls_sha512_context* *ctx)

  This function initializes the SHA-512_t context.
- void *mbedtls_sha512_t_free* (*mbedtls_sha512_context* *ctx)

  This function clears the SHA-512_t context.
- void *mbedtls_sha512_t_starts* (*mbedtls_sha512_context* *ctx, int is224)

  This function starts a SHA-512_t checksum calculation.
- void *mbedtls_sha512_t_update* (*mbedtls_sha512_context* *ctx, const unsigned char *input, size_t ilen)

  This function feeds an input buffer into an ongoing SHA-512_t checksum calculation.
- void *mbedtls_sha512_t_finish* (*mbedtls_sha512_context* *ctx, unsigned char output[32], int is224)

  This function finishes the SHA-512_t operation, and writes the result to the output buffer.
- void *mbedtls_sha512_t* (const unsigned char *input, size_t ilen, unsigned char output[32], int is224)

  This function calculates the SHA-512 checksum of a buffer.

Detailed description

Contains all CryptoCell SHA-512 truncated APIs.

Function documentation

void mbedtls_sha512_t (const unsigned char * input, size_t ilen, unsigned char output[32], int is224)

This function calculates the SHA-512 checksum of a buffer.

The function allocates the context, performs the calculation, and frees the context.

The SHA-512 result is calculated as output = SHA-512(input buffer).

Parameters:

Parameter	Description
`input`	The buffer holding the input data.
`ilen`	The length of the input data.
`output`	The SHA-512/256 or SHA-512/224 checksum result.
`is224`	Determines which function to use. • 0: Use SHA-512/256. • 1: Use SHA-512/224.

void mbedtls_sha512_t_finish (*mbedtls_sha512_context* * ctx, unsigned char output[32], int is224)

This function finishes the SHA-512_t operation, and writes the result to the output buffer.

- For SHA512/224, the output buffer will include the 28 leftmost Bytes of the SHA-512 digest.
- For SHA512/256, the output buffer will include the 32 leftmost bytes of the SHA-512 digest.

Parameters:

Parameter	Description
`ctx`	The SHA-512_t context.
`output`	The SHA-512/256 or SHA-512/224 checksum result.
`is224`	Determines which function to use. • 0: Use SHA-512/256. • 1: Use SHA-512/224.

void mbedtls_sha512_t_free (*mbedtls_sha512_context* * ctx)

This function clears the SHA-512_t context.

Parameters:

Parameter	Description
`ctx`	The SHA-512_t context to clear.

void mbedtls_sha512_t_init (*mbedtls_sha512_context* * ctx)

This function initializes the SHA-512_t context.

Parameters:

Parameter	Description
`ctx`	The SHA-512_t context to initialize.

void mbedtls_sha512_t_starts (*mbedtls_sha512_context* * ctx, int is224)

This function starts a SHA-512_t checksum calculation.

Parameters:

Parameter	Description
`ctx`	The context to initialize.
`is224`	Determines which function to use. • 0: Use SHA-512/256. • 1: Use SHA-512/224.

void mbedtls_sha512_t_update (*mbedtls_sha512_context* * ctx, const unsigned char * input, size_t ilen)

This function feeds an input buffer into an ongoing SHA-512_t checksum calculation.

Parameters:

Parameter	Description
`ctx`	The SHA-512_t context.
`input`	The buffer holding the input data.
`ilen`	The length of the input data.

1.5.14 CryptoCell HKDF key-derivation function API

Contains the CryptoCell HMAC key-derivation function API.

Modules

- *Specific errors of the HMAC key-derivation APIs*

Contains the CryptoCell HKDF-API error definitions. Macros

- #define *CC_HKDF_MAX_HASH_KEY_SIZE_IN_BYTES* 512
- #define *CC_HKDF_MAX_HASH_DIGEST_SIZE_IN_BYTES* *CC_HASH_SHA512_DIGEST_SIZE_IN_BYTES*

Enumerations

- enum *mbedtls_hkdf_hashmode_t* { *CC_HKDF_HASH_SHA1_mode* = 0, *CC_HKDF_HASH_SHA224_mode* = 1, *CC_HKDF_HASH_SHA256_mode* = 2, *CC_HKDF_HASH_SHA384_mode* = 3, *CC_HKDF_HASH_SHA512_mode* = 4, *CC_HKDF_HASH_NumOfModes*, *CC_HKDF_HASH_OpModeLast* = 0x7FFFFFFF }

Functions

- CCError_t *mbedtls_hkdf_key_derivation* (*mbedtls_hkdf_hashmode_t* HKDFhashMode, uint8_t *Salt_ptr, size_t SaltLen, uint8_t *Ikm_ptr, uint32_t IkmLen, uint8_t *Info, uint32_t InfoLen, uint8_t *Okm, uint32_t OkmLen, *CCBool* IsStrongKey)

  mbedtls_hkdf_key_derivation() performs the HMAC-based key derivation, as define by RFC-5869: HMAC-based Extract-and-Expand Key Derivation Function (HKDF).

Detailed description

Contains the CryptoCell HMAC key-derivation function API.

Macro definition documentation

#define CC_HKDF_MAX_HASH_DIGEST_SIZE_IN_BYTES *CC_HASH_SHA512_DIGEST_SIZE_IN_BYTES*

The maximal size of the HKDF hash-digest in Bytes.

#define CC_HKDF_MAX_HASH_KEY_SIZE_IN_BYTES 512

The maximal size of the HKDF key in words.

Enumeration type documentation

enum *mbedtls_hkdf_hashmode_t*

Supported HKDF hash modes.

Enumerator:

Enum	Description
`CC_HKDF_HASH_SHA1_mode`	SHA-1 mode.
`CC_HKDF_HASH_SHA224_mode`	SHA-224 mode.
`CC_HKDF_HASH_SHA256_mode`	SHA-256 mode.
`CC_HKDF_HASH_SHA384_mode`	SHA-384 mode.
`CC_HKDF_HASH_SHA512_mode`	SHA-512 mode.
`CC_HKDF_HASH_NumOfModes`	The maximal number of hash modes.

Enum	Description
`CC_HKDF_HASH_OpModeLast`	Reserved.

Function documentation

CCError_t mbedtls_hkdf_key_derivation (*mbedtls_hkdf_hashmode_t* HKDFhashMode, uint8_t * Salt_ptr, size_t SaltLen, uint8_t * Ikm_ptr, uint32_t IkmLen, uint8_t * Info, uint32_t InfoLen, uint8_t * Okm, uint32_t OkmLen, *CCBool* IsStrongKey)

mbedtls_hkdf_key_derivation() performs the HMAC-based key derivation, as define by RFC-5869: HMAC-based Extract-and-Expand Key Derivation Function (HKDF).

Returns:

CC_OK on success.

A non-zero value on failure as defined in cc_kdf_error.h, or in *md.h*.

Parameters:

I/O	Parameter	Description
in	`HKDFhashMode`	The HKDF identifier of the hash function to be used.
in	`Salt_ptr`	A pointer to a non-secret random value. Can be NULL.
in	`SaltLen`	The size of the Salt_ptr.
in	`Ikm_ptr`	A pointer to an input key message.
in	`IkmLen`	The size of the input key message
in	`Info`	A pointer to an optional context and application-specific information. Can be NULL
in	`InfoLen`	The size of the application-specific information.
in	`Okm`	A pointer to an output key material.
in	`OkmLen`	The size of the output key material.
in	`IsStrongKey`	If TRUE, no need to perform the extraction phase.

Specific errors of the HMAC key-derivation APIs

Contains the CryptoCell HKDF-API error definitions.

Macros

- #define *CC_HKDF_INVALID_ARGUMENT_POINTER_ERROR* (*CC_HKDF_MODULE_ERROR_BASE* + 0x0UL)
- #define *CC_HKDF_INVALID_ARGUMENT_SIZE_ERROR* (*CC_HKDF_MODULE_ERROR_BASE* + 0x1UL)
- #define *CC_HKDF_INVALID_ARGUMENT_HASH_MODE_ERROR* (*CC_HKDF_MODULE_ERROR_BASE* + 0x3UL)

- #define *CC_HKDF_IS_NOT_SUPPORTED* (*CC_HKDF_MODULE_ERROR_BASE* + 0xFFUL)

Detailed description

Contains the CryptoCell HKDF-API error definitions.

Macro definition documentation

#define CC_HKDF_INVALID_ARGUMENT_HASH_MODE_ERROR (*CC_HKDF_MODULE_ERROR_BASE* + 0x3UL)

Illegal hash mode.

#define CC_HKDF_INVALID_ARGUMENT_POINTER_ERROR (*CC_HKDF_MODULE_ERROR_BASE* + 0x0UL)

Invalid argument.

#define CC_HKDF_INVALID_ARGUMENT_SIZE_ERROR (*CC_HKDF_MODULE_ERROR_BASE* + 0x1UL)

Invalid argument size.

#define CC_HKDF_IS_NOT_SUPPORTED (*CC_HKDF_MODULE_ERROR_BASE* + 0xFFUL)

HKDF not supported.

1.5.15 CryptoCell management APIs

Contains CryptoCell Management APIs.

Modules

- *Specific errors of the CryptoCell Management APIs*

Contains the CryptoCell management-API error definitions. Data structures

- union *mbedtls_mng_apbcconfig*

Macros

- #define *CC_MNG_LCS_CM* 0x0
- #define *CC_MNG_LCS_DM* 0x1
- #define *CC_MNG_LCS_SEC_ENABLED* 0x5
- #define *CC_MNG_LCS_RMA* 0x7

Typedefs

- typedef union *mbedtls_mng_apbcconfig* *mbedtls_mng_apbcconfig*

Enumerations

- enum *mbedtls_mng_rmastatus* { CC_MNG_NON_RMA = 0, *CC_MNG_PENDING_RMA* = 1, *CC_MNG_ILLEGAL_STATE* = 2, *CC_MNG_RMA* = 3 }

- enum *mbedtls_mng_keytype* { CC_MNG_HUK_KEY = 0, *CC_MNG_RTL_KEY* = 1, *CC_MNG_PROV_KEY* = 2, *CC_MNG_CE_KEY* = 3, *CC_MNG_ICV_PROV_KEY* = 4, *CC_MNG_ICV_CE_KEY* = 5, *CC_MNG_TOTAL_HW_KEYS* = 6, *CC_MNG_END_OF_KEY_TYPE* = 0x7FFFFFFF }

Functions

- int *mbedtls_mng_pending_rma_status_get* (uint32_t *rmaStatus)

  This function reads the OTP word of the OEM flags, and returns the OEM RMA flag status: TRUE or FALSE.

- int *mbedtls_mng_hw_version_get* (uint32_t *partNumber, uint32_t *revision)

  This function verifies and returns the CryptoCell HW version.

- int *mbedtls_mng_cc_sec_mode_set* (CCBool_t isSecAccessMode, CCBool_t isSecModeLock)

  This function sets CryptoCell to Secure mode.

- int *mbedtls_mng_cc_priv_mode_set* (CCBool_t isPrivAccessMode, CCBool_t isPrivModeLock)

  This function sets CryptoCell to Privilege mode.

- int *mbedtls_mng_debug_key_set* (*mbedtls_mng_keytype* keyType, uint32_t *pHwKey, size_t keySize)

  This function sets the shadow register of one of the HW Keys when the device is in CM LCS or DM LCS.

- int *mbedtls_mng_gen_config_get* (uint32_t *pOtpWord)

  This function retrieves the general configuration from the OTP. See Arm TrustZone CryptoCell-312 Software Integrators Manual.

- int *mbedtls_mng_oem_key_lock* (CCBool_t kcpLock, CCBool_t kceLock)

  This function locks the usage of either Kcp, Kce, or both during runtime, in either Secure LCS or RMA LCS.

- int *mbedtls_mng_apbc_config_set* (*mbedtls_mng_apbcconfig* apbcConfig)

  This function sets the CryptoCell APB-C into one of the following modes:

- int *mbedtls_mng_apbc_access* (CCBool_t isApbcAccessUsed)

  This function requests usage of or releases the APB-C.

- int *mbedtls_mng_suspend* (uint8_t *pBackupBuffer, size_t backupSize)

  This function is called once the external PMU decides to power-down CryptoCell.

- int *mbedtls_mng_resume* (uint8_t *pBackupBuffer, size_t backupSize)

  This function is called once the external PMU decides to power-up CryptoCell.

Detailed description

Contains CryptoCell Management APIs.

Macro definition documentation

#define CC_MNG_LCS_CM   0x0

Chip manufacturer (CM LCS).

#define CC_MNG_LCS_DM 0x1

Device manufacturer (DM LCS).

#define CC_MNG_LCS_RMA 0x7

RMA (RMA LCS).

#define CC_MNG_LCS_SEC_ENABLED 0x5

Security enabled (Secure LCS).

Typedef documentation

typedef union *mbedtls_mng_apbcconfig* *mbedtls_mng_apbcconfig*

Input to the *mbedtls_mng_apbc_config_set()* function.

Enumeration type documentation

enum *mbedtls_mng_keytype*

AES HW key types.

Enumerator:

Enum	Description
`CC_MNG_RTL_KEY`	Device root key (HUK).
`CC_MNG_PROV_KEY`	Platform key (Krtl).
`CC_MNG_CE_KEY`	ICV provisioning key (Kcp).
`CC_MNG_ICV_PROV_KEY`	OEM code-encryption key (Kce).
`CC_MNG_ICV_CE_KEY`	OEM provisioning key (Kpicv).
`CC_MNG_TOTAL_HW_KEYS`	ICV code-encryption key (Kceicv).
`CC_MNG_END_OF_KEY_TYPE`	Total number of HW Keys.

enum *mbedtls_mng_rmastatus*

RMA statuses.

Enumerator:

Enum	Description
`CC_MNG_PENDING_RMA`	Non-RMA: bit [30] = 0, bit [31] = 0.
`CC_MNG_ILLEGAL_STATE`	Pending RMA: bit [30] = 1, bit [31] = 0.
`CC_MNG_RMA`	Illegal state: bit [30] = 0, bit [31] = 1.

Function documentation

int mbedtls_mng_apbc_access (CCBool_t isApbcAccessUsed)

This function requests usage of or releases the APB-C.

Note

This function must be called before and after each use of APB-C.

Returns:

CC_OK on success.

A non-zero value from *mbedtls_cc_mng_error.h* on failure.

Parameters:

I/O	Parameter	Description
in	`isApbcAccessUsed`	• TRUE: Request usage of APB-C • FALSE: Free APB-C.

int mbedtls_mng_apbc_config_set (mbedtls_mng_apbcconfig apbcConfig)

This function sets the CryptoCell APB-C into one of the following modes:

- Secured access mode.
- Privileged access mode.
- Instruction access.

Returns:

CC_OK on success.

A non-zero value from *mbedtls_cc_mng_error.h* on failure.

Parameters:

Parameter	Description
`apbcConfig`	APB-C mode.

int mbedtls_mng_cc_priv_mode_set (CCBool_t isPrivAccessMode, CCBool_t isPrivModeLock)

This function sets CryptoCell to Privilege mode.

It is done only while CryptoCell is idle.

Returns:

CC_OK on success.

A non-zero value from *mbedtls_cc_mng_error.h* on failure.

Parameters:

I/O	Parameter	Description
in	`isPrivAccessMode`	• True: Set CryptoCell to privileged mode. • False: Set CryptoCell to unprivileged mode.
in	`isPrivModeLock`	• True: Lock CryptoCell to current mode. • False: Do not lock CryptoCell to current mode. Allows calling this function again.

int mbedtls_mng_cc_sec_mode_set (CCBool_t isSecAccessMode, CCBool_t isSecModeLock)

This function sets CryptoCell to Secure mode.

It is done only while CryptoCell is idle.

Returns:

CC_OK on success.

A non-zero value from *mbedtls_cc_mng_error.h* on failure.

Parameters:

I/O	Parameter	Description
in	`isSecAccessMode`	• True: Set CryptoCell to Secure mode. • False: Set CryptoCell to non-Secure mode.
in	`isSecModeLock`	• True: Lock CryptoCell to current mode. • False: Do not lock CryptoCell to current mode. Allows calling this function again.

int mbedtls_mng_debug_key_set (*mbedtls_mng_keytype* keyType, uint32_t * pHwKey, size_t keySize)

This function sets the shadow register of one of the HW Keys when the device is in CM LCS or DM LCS.

Returns:

CC_OK on success.

A non-zero value from *mbedtls_cc_mng_error.h* on failure.

Parameters:

I/O	Parameter	Description
in	`keyType`	The HW-key type: • HUK • Kcp • Kce • Kpicv • Kceicv
in	`pHwKey`	A pointer to the HW-key buffer.
in	`keySize`	The size of the HW key in Bytes.

int mbedtls_mng_gen_config_get (uint32_t * pOtpWord)

This function retrieves the general configuration from the OTP. See Arm TrustZone CryptoCell-312 Software Integrators Manual.

Returns:

CC_OK on success.

A non-zero value from *mbedtls_cc_mng_error.h* on failure.

Parameters:

I/O	Parameter	Description
out	`pOtpWord`	The OTP configuration word.

int mbedtls_mng_hw_version_get (uint32_t * partNumber, uint32_t * revision)

This function verifies and returns the CryptoCell HW version.

Returns:

CC_OK on success.

A non-zero value from *mbedtls_cc_mng_error.h* on failure.

Parameters:

I/O	Parameter	Description
out	`partNumber`	The part number.
out	`revision`	The HW revision.

int mbedtls_mng_oem_key_lock (CCBool_t kcpLock, CCBool_t kceLock)

This function locks the usage of either Kcp, Kce, or both during runtime, in either Secure LCS or RMA LCS.

Returns:

CC_OK on success.

A non-zero value from *mbedtls_cc_mng_error.h* on failure.

Parameters:

I/O	Parameter	Description
in	`kcpLock`	The flag for locking Kcp usage.
in	`kceLock`	The flag for locking Kce usage.

int mbedtls_mng_pending_rma_status_get (uint32_t * rmaStatus)

This function reads the OTP word of the OEM flags, and returns the OEM RMA flag status: TRUE or FALSE.

The function returns the value only in DM LCS or Secure LCS. It validates the device RoT configuration, and returns the value only if both HBK0 and HBK1 are supported. Otherwise, it returns FALSE regardless to the OTP status.

Returns:

CC_OK on success.

A non-zero value from *mbedtls_cc_mng_error.h* on failure.

Parameters:

I/O	Parameter	Description
out	`rmaStatus`	The RMA status.

int mbedtls_mng_resume (uint8_t * pBackupBuffer, size_t backupSize)

This function is called once the external PMU decides to power-up CryptoCell.

Returns:

CC_OK on success.

A non-zero value from *mbedtls_cc_mng_error.h* on failure.

Parameters:

I/O	Parameter	Description
in	`pBackupBuffer`	A pointer to a buffer that can be used for backup.
in	`backupSize`	The size of the backup buffer. Must be at least CC_MNG_MIN_BACKUP_SIZE_IN_BYTES.

int mbedtls_mng_suspend (uint8_t * pBackupBuffer, size_t backupSize)

This function is called once the external PMU decides to power-down CryptoCell.

Returns:

CC_OK on success.

A non-zero value from *mbedtls_cc_mng_error.h* on failure.

Parameters:

I/O	Parameter	Description
in	`pBackupBuffer`	A pointer to a buffer that can be used for backup.
in	`backupSize`	The size of the backup buffer. Must be at least CC_MNG_MIN_BACKUP_SIZE_IN_BYTES.

Specific errors of the CryptoCell Management APIs

Contains the CryptoCell management-API error definitions.

Macros

- #define *CC_MNG_ILLEGAL_INPUT_PARAM_ERR* (*CC_MNG_MODULE_ERROR_BASE* + 0x00UL)
- #define *CC_MNG_ILLEGAL_OPERATION_ERR* (*CC_MNG_MODULE_ERROR_BASE* + 0x01UL)
- #define *CC_MNG_ILLEGAL_PIDR_ERR* (*CC_MNG_MODULE_ERROR_BASE* + 0x02UL)
- #define *CC_MNG_ILLEGAL_CIDR_ERR* (*CC_MNG_MODULE_ERROR_BASE* + 0x03UL)
- #define *CC_MNG_APB_SECURE_IS_LOCKED_ERR* (*CC_MNG_MODULE_ERROR_BASE* + 0x04UL)
- #define *CC_MNG_APB_PRIVILEGE_IS_LOCKED_ERR* (*CC_MNG_MODULE_ERROR_BASE* + 0x05UL)
- #define *CC_MNG_APBC_SECURE_IS_LOCKED_ERR* (*CC_MNG_MODULE_ERROR_BASE* + 0x06UL)
- #define *CC_MNG_APBC_PRIVILEGE_IS_LOCKED_ERR* (*CC_MNG_MODULE_ERROR_BASE* + 0x07UL)
- #define *CC_MNG_APBC_INSTRUCTION_IS_LOCKED_ERR* (*CC_MNG_MODULE_ERROR_BASE* + 0x08UL)
- #define *CC_MNG_INVALID_KEY_TYPE_ERROR* (*CC_MNG_MODULE_ERROR_BASE* + 0x09UL)
- #define *CC_MNG_ILLEGAL_HUK_SIZE_ERR* (*CC_MNG_MODULE_ERROR_BASE* + 0x0AUL)

- #define *CC_MNG_ILLEGAL_HW_KEY_SIZE_ERR* (*CC_MNG_MODULE_ERROR_BASE* + 0x0BUL)
- #define *CC_MNG_HW_KEY_IS_LOCKED_ERR* (*CC_MNG_MODULE_ERROR_BASE* + 0x0CUL)
- #define *CC_MNG_KCP_IS_LOCKED_ERR* (*CC_MNG_MODULE_ERROR_BASE* + 0x0DUL)
- #define *CC_MNG_KCE_IS_LOCKED_ERR* (*CC_MNG_MODULE_ERROR_BASE* + 0x0EUL)
- #define *CC_MNG_RMA_ILLEGAL_STATE_ERR* (*CC_MNG_MODULE_ERROR_BASE* + 0x0FUL)
- #define *CC_MNG_AO_WRITE_FAILED_ERR* (*CC_MNG_MODULE_ERROR_BASE* + 0x10UL)
- #define *CC_MNG_APBC_ACCESS_FAILED_ERR* (*CC_MNG_MODULE_ERROR_BASE* + 0x11UL)
- #define *CC_MNG_PM_SUSPEND_RESUME_FAILED_ERR* (*CC_MNG_MODULE_ERROR_BASE* + 0x12UL)
- #define *CC_MNG_ILLEGAL_SW_VERSION_ERR* (*CC_MNG_MODULE_ERROR_BASE* + 0x13UL)
- #define *CC_MNG_HASH_NOT_PROGRAMMED_ERR* (*CC_MNG_MODULE_ERROR_BASE* + 0x14UL)
- #define *CC_MNG_HBK_ZERO_COUNT_ERR* (*CC_MNG_MODULE_ERROR_BASE* + 0x15UL)

Detailed description

Contains the CryptoCell management-API error definitions.

Macro definition documentation

#define CC_MNG_AO_WRITE_FAILED_ERR (*CC_MNG_MODULE_ERROR_BASE* + 0x10UL)

Error returned from AO write operation.

#define CC_MNG_APB_PRIVILEGE_IS_LOCKED_ERR (*CC_MNG_MODULE_ERROR_BASE* + 0x05UL)

APB Privilege is locked.

#define CC_MNG_APB_SECURE_IS_LOCKED_ERR (*CC_MNG_MODULE_ERROR_BASE* + 0x04UL)

APB Secure is locked.

#define CC_MNG_APBC_ACCESS_FAILED_ERR (*CC_MNG_MODULE_ERROR_BASE* + 0x11UL)

APBC access failure.

#define CC_MNG_APBC_INSTRUCTION_IS_LOCKED_ERR (*CC_MNG_MODULE_ERROR_BASE* + 0x08UL)

APBC Instruction is locked.

#define CC_MNG_APBC_PRIVILEGE_IS_LOCKED_ERR (CC_MNG_MODULE_ERROR_BASE + 0x07UL)

APBC Privilege is locked.

#define CC_MNG_APBC_SECURE_IS_LOCKED_ERR (CC_MNG_MODULE_ERROR_BASE + 0x06UL)

APBC Secure is locked.

#define CC_MNG_HASH_NOT_PROGRAMMED_ERR (CC_MNG_MODULE_ERROR_BASE + 0x14UL)

Hash Public Key NA.

#define CC_MNG_HBK_ZERO_COUNT_ERR (CC_MNG_MODULE_ERROR_BASE + 0x15UL)

Illegal hash boot key zero count in the OTP error.

#define CC_MNG_HW_KEY_IS_LOCKED_ERR (CC_MNG_MODULE_ERROR_BASE + 0x0CUL)

HW key is locked.

#define CC_MNG_ILLEGAL_CIDR_ERR (CC_MNG_MODULE_ERROR_BASE + 0x03UL)

Illegal Component ID.

#define CC_MNG_ILLEGAL_HUK_SIZE_ERR (CC_MNG_MODULE_ERROR_BASE + 0x0AUL)

Illegal size of HUK.

#define CC_MNG_ILLEGAL_HW_KEY_SIZE_ERR (CC_MNG_MODULE_ERROR_BASE + 0x0BUL)

Illegal size for any HW key other than HUK.

#define CC_MNG_ILLEGAL_INPUT_PARAM_ERR (CC_MNG_MODULE_ERROR_BASE + 0x00UL)

Illegal input parameter.

#define CC_MNG_ILLEGAL_OPERATION_ERR (CC_MNG_MODULE_ERROR_BASE + 0x01UL)

Illegal operation.

#define CC_MNG_ILLEGAL_PIDR_ERR (CC_MNG_MODULE_ERROR_BASE + 0x02UL)

Illegal Peripheral ID.

#define CC_MNG_ILLEGAL_SW_VERSION_ERR (CC_MNG_MODULE_ERROR_BASE + 0x13UL)

SW version failure.

#define CC_MNG_INVALID_KEY_TYPE_ERROR (CC_MNG_MODULE_ERROR_BASE + 0x09UL)

Invalid Key type.

#define CC_MNG_KCE_IS_LOCKED_ERR (CC_MNG_MODULE_ERROR_BASE + 0x0EUL)

Kce is locked.

#define CC_MNG_KCP_IS_LOCKED_ERR (CC_MNG_MODULE_ERROR_BASE + 0x0DUL)

Kcp is locked.

#define CC_MNG_PM_SUSPEND_RESUME_FAILED_ERR (CC_MNG_MODULE_ERROR_BASE + 0x12UL)

PM SUSPEND/RESUME failure.

#define CC_MNG_RMA_ILLEGAL_STATE_ERR (CC_MNG_MODULE_ERROR_BASE + 0x0FUL)

RMA Illegal state.

1.5.16 CryptoCell PAL APIs

Groups all PAL APIs and definitions.

Modules

- *CryptoCell PAL abort operations*
- *CryptoCell PAL memory Barrier APIs*
- Contains memory-barrier implementation definitions and APIs. *CryptoCell PAL platform-dependent compiler-specific definitions*
- Contains CryptoCell PAL platform-dependent compiler-related definitions. *Specific errors of the CryptoCell PAL APIs*
- Contains platform-dependent PAL-API error definitions. *CryptoCell PAL entry or exit point APIs*
- Contains PAL initialization and termination APIs. *CryptoCell PAL logging APIs and definitions*
- Contains CryptoCell PAL layer log definitions. *CryptoCell PAL memory operations*
- Contains memory-operation functions. *CryptoCell PAL memory mapping APIs*
- Contains memory mapping functions. *CryptoCell PAL mutex APIs*
- Contains resource management functions. *CryptoCell PAL platform-dependent definitions and types*
- Contains CryptoCell PAL platform-dependent definitions and types. *CryptoCell PAL definitions for Boot Services*
- Contains CryptoCell PAL Secure Boot definitions. *CryptoCell PAL power-management APIs*
- Contains PAL power-management APIs. *CryptoCell PAL TRNG APIs*

Contains APIs for retrieving TRNG user parameters.

Detailed description

Groups all PAL APIs and definitions.

CryptoCell PAL abort operations

Functions

- void *CC_PalAbort* (const char *exp)

  This function performs the "Abort" operation.

Detailed description

Function documentation

void CC_PalAbort (const char * exp)

  This function performs the "Abort" operation.

Must be implemented according to platform and OS.

CryptoCell PAL memory Barrier APIs

Contains memory-barrier implementation definitions and APIs.

Functions

- void *CC_PalWmb* (void)
- void *CC_PalRmb* (void)

Detailed description

Contains memory-barrier implementation definitions and APIs.

Function documentation

void CC_PalRmb (void)

This macro is puts the memory barrier before the read operation.

Returns:

None

void CC_PalWmb (void)

This macro is puts the memory barrier after the write operation.

Returns:

None

CryptoCell PAL platform-dependent compiler-specific definitions

Contains CryptoCell PAL platform-dependent compiler-related definitions.

Macros

- #define *CC_PAL_COMPILER_SECTION*(sectionName) __attribute__((section(sectionName)))
- #define *CC_PAL_COMPILER_KEEP_SYMBOL* __attribute__((used))
- #define *CC_PAL_COMPILER_ALIGN*(alignement) __attribute__((aligned(alignement)))
- #define *CC_PAL_COMPILER_FUNC_NEVER_RETURNS* __attribute__((noreturn))

- #define *CC_PAL_COMPILER_FUNC_DONT_INLINE* __attribute__((noinline))
- #define *CC_PAL_COMPILER_TYPE_MAY_ALIAS* __attribute__((__may_alias__))
- #define *CC_PAL_COMPILER_SIZEOF_STRUCT_MEMBER*(type_name, member_name) sizeof(((type_name *)0)->member_name)
- #define *CC_ASSERT_CONCAT_*(a, b) a##b
- #define *CC_ASSERT_CONCAT*(a, b) *CC_ASSERT_CONCAT_*(a, b)
- #define *CC_PAL_COMPILER_ASSERT*(cond, message) enum { *CC_ASSERT_CONCAT*(assert_line_, __LINE__) = 1/(!!(cond)) }

Detailed description

Contains CryptoCell PAL platform-dependent compiler-related definitions.

Macro definition documentation

#define CC_ASSERT_CONCAT(a, b) *CC_ASSERT_CONCAT_*(a, b)

Definition of assertion.

#define CC_ASSERT_CONCAT_(a, b) a##b

Definition of assertion.

#define CC_PAL_COMPILER_ALIGN(alignement) __attribute__((aligned(alignement)))

Make a given data item aligned (alignment in Bytes).

#define CC_PAL_COMPILER_ASSERT(cond, message) enum { *CC_ASSERT_CONCAT*(assert_line_, __LINE__) = 1/(!!(cond)) }

Definition of assertion.

#define CC_PAL_COMPILER_FUNC_DONT_INLINE __attribute__((noinline))

Prevent a function from being inlined.

#define CC_PAL_COMPILER_FUNC_NEVER_RETURNS __attribute__((noreturn))

Mark a function that never returns.

#define CC_PAL_COMPILER_KEEP_SYMBOL __attribute__((used))

Mark symbol as used, that is, prevent the garbage collector from dropping it.

#define CC_PAL_COMPILER_SECTION(sectionName) __attribute__((section(sectionName)))

Associate a symbol with a link section.

#define CC_PAL_COMPILER_SIZEOF_STRUCT_MEMBER(type_name, member_name) sizeof(((type_name *)0)->member_name)

Get the size of a structure-type member.

#define CC_PAL_COMPILER_TYPE_MAY_ALIAS __attribute__((__may_alias__))

Given data type might serve as an alias for another data-type pointer.

CryptoCell PAL entry or exit point APIs

Contains PAL initialization and termination APIs.

Functions

- int *CC_PalInit* (void)

This function performs all initializations that may be required by your PAL implementation, specifically by the DMA-able buffer scheme.

- void *CC_PalTerminate* (void)

  This function terminates the PAL implementation and frees the resources that were allocated by *CC_PalInit*.

Detailed description

Contains PAL initialization and termination APIs.

Function documentation

int CC_PalInit (void)

This function performs all initializations that may be required by your PAL implementation, specifically by the DMA-able buffer scheme.

The existing implementation allocates a contiguous memory pool that is later used by the CryptoCell implementation. If no initializations are needed in your environment, the function can be minimized to return OK. It is called by *CC_LibInit*.

Returns:

A non-zero value in case of failure.

void CC_PalTerminate (void)

This function terminates the PAL implementation and frees the resources that were allocated by *CC_PalInit*.

Returns:

Void.

CryptoCell PAL logging APIs and definitions

Contains CryptoCell PAL layer log definitions.

Macros

- #define *CC_PAL_LOG_LEVEL_NULL* (-1)
- #define *CC_PAL_LOG_LEVEL_ERR* 0
- #define *CC_PAL_LOG_LEVEL_WARN* 1
- #define *CC_PAL_LOG_LEVEL_INFO* 2
- #define *CC_PAL_LOG_LEVEL_DEBUG* 3
- #define *CC_PAL_LOG_LEVEL_TRACE* 4
- #define *CC_PAL_LOG_LEVEL_DATA* 5
- #define *CC_PAL_LOG_CUR_COMPONENT* 0xFFFFFFFF
- #define *CC_PAL_LOG_CUR_COMPONENT_NAME* "CC"
- #define *CC_PAL_MAX_LOG_LEVEL* *CC_PAL_LOG_LEVEL_ERR* /**CC_PAL_LOG_LEVEL_DEBUG**/
- #define *__CC_PAL_LOG_LEVEL_EVAL*(level) level
- #define *_CC_PAL_MAX_LOG_LEVEL* *__CC_PAL_LOG_LEVEL_EVAL*(*CC_PAL_MAX_LOG_LEVEL*)
- #define inline __inline

- #define CC_PalLogInit() do {} while (0)
- #define CC_PalLogLevelSet(setLevel) do {} while (0)
- #define CC_PalLogMaskSet(setMask) do {} while (0)
- #define _CC_PAL_LOG(level, format, ...)
- #define CC_PAL_LOG_ERR(format, ...) _CC_PAL_LOG(ERR, format, ##__VA_ARGS__)
- #define CC_PAL_LOG_WARN(...) do {} while (0)
- #define CC_PAL_LOG_INFO(...) do {} while (0)
- #define CC_PAL_LOG_DEBUG(...) do {} while (0)
- #define CC_PAL_LOG_DUMP_BUF(msg, buf, size) do {} while (0)
- #define CC_PAL_LOG_TRACE(...) do {} while (0)
- #define CC_PAL_LOG_DATA(...) do {} while (0)

Detailed description

Contains CryptoCell PAL layer log definitions.

Macro definition documentation

#define __CC_PAL_LOG_LEVEL_EVAL(level) level

Evaluate CC_PAL_MAX_LOG_LEVEL in case provided by caller.

#define _CC_PAL_LOG(level, format, ...)

```
Value:if (CC_PAL_logMask & CC_PAL_LOG_CUR_COMPONENT) \
        CC_PalLog(CC_PAL_LOG_LEVEL_ ## level, "%s:%s: " format,
CC_PAL_LOG_CUR_COMPONENT_NAME, __func__, ##__VA_ARGS__)
```

Filter logging based on logMask , and dispatch to platform-specific logging mechanism.

#define _CC_PAL_MAX_LOG_LEVEL __CC_PAL_LOG_LEVEL_EVAL(CC_PAL_MAX_LOG_LEVEL)

The maximal log-level definition.

#define CC_PAL_LOG_CUR_COMPONENT 0xFFFFFFFF

Default log debugged component.

#define CC_PAL_LOG_CUR_COMPONENT_NAME "CC"

Default log debugged component.

#define CC_PAL_LOG_DATA(...) do {} while (0)

Log debug data.

#define CC_PAL_LOG_DEBUG(...) do {} while (0)

Log debug messages.

#define CC_PAL_LOG_DUMP_BUF(msg, buf, size) do {} while (0)

Log debug buffer.

#define CC_PAL_LOG_ERR(format, ...) _CC_PAL_LOG(ERR, format, ##__VA_ARGS__)

Log messages according to log level.

#define CC_PAL_LOG_INFO(...) do {} while (0)

Log messages according to log level.

#define CC_PAL_LOG_LEVEL_DATA   5

PAL log level - data.

#define CC_PAL_LOG_LEVEL_DEBUG   3

PAL log level - debug.

#define CC_PAL_LOG_LEVEL_ERR   0

PAL log level - error.

#define CC_PAL_LOG_LEVEL_INFO   2

PAL log level - info.

#define CC_PAL_LOG_LEVEL_NULL   (-1)

PAL log level - disabled.

#define CC_PAL_LOG_LEVEL_TRACE   4

PAL log level - trace.

#define CC_PAL_LOG_LEVEL_WARN   1

PAL log level - warning.

#define CC_PAL_LOG_TRACE(...)   do {} while (0)

Log debug trace.

#define CC_PAL_LOG_WARN(...)   do {} while (0)

Log messages according to log level.

#define CC_PAL_MAX_LOG_LEVEL   *CC_PAL_LOG_LEVEL_ERR* /**CC_PAL_LOG_LEVEL_DEBUG**/

Default debug log level, when debug is set to on.

#define CC_PalLogInit()   do {} while (0)

Log initialization function.

#define CC_PalLogLevelSet(setLevel)   do {} while (0)

Log set-level function - sets the level of logging in case of debug.

#define CC_PalLogMaskSet(setMask)   do {} while (0)

Log set-mask function - sets the component-masking in case of debug.

CryptoCell PAL memory operations

Contains memory-operation functions.

Macros

- #define *CC_PalMemCmp*(aTarget,   aSource,   aSize)   CC_PalMemCmpPlat(aTarget, aSource, aSize)

  This function compares between two given buffers, according to the given size.

- #define *CC_PalMemCopy*(aDestination,   aSource,
  aSize)   CC_PalMemCopyPlat(aDestination, aSource, aSize)

  This function copies aSize   Bytes from the source buffer to the destination buffer.

- #define *CC_PalMemMove*(aDestination, aSource, aSize) CC_PalMemMovePlat(aDestination, aSource, aSize)

  This function moves aSize Bytes from the source buffer to the destination buffer. This function supports overlapped buffers.
- #define *CC_PalMemSet*(aTarget, aChar, aSize) CC_PalMemSetPlat(aTarget, aChar, aSize)

  This function sets aSize Bytes of aChar in the given buffer.
- #define *CC_PalMemSetZero*(aTarget, aSize) CC_PalMemSetZeroPlat(aTarget, aSize)

  This function sets aSize Bytes in the given buffer with zeroes.
- #define *CC_PalMemMalloc*(aSize) CC_PalMemMallocPlat(aSize)

  This function allocates a memory buffer according to aSize.
- #define *CC_PalMemRealloc*(aBuffer, aNewSize) CC_PalMemReallocPlat(aBuffer, aNewSize)

  This function reallocates a memory buffer according to aNewSize. The contents of the old buffer is moved to the new location.
- #define *CC_PalMemFree*(aBuffer) CC_PalMemFreePlat(aBuffer)

  This function frees a previously-allocated buffer.

Detailed description

Contains memory-operation functions.

Macro definition documentation

#define CC_PalMemCmp(aTarget, aSource, aSize) CC_PalMemCmpPlat(aTarget, aSource, aSize)

This function compares between two given buffers, according to the given size.

Returns:

The return values are according to operating-system return values.

#define CC_PalMemCopy(aDestination, aSource, aSize) CC_PalMemCopyPlat(aDestination, aSource, aSize)

This function copies aSize Bytes from the source buffer to the destination buffer.

Returns:

void.

#define CC_PalMemFree(aBuffer) CC_PalMemFreePlat(aBuffer)

This function frees a previously-allocated buffer.

Returns:

void.

#define CC_PalMemMalloc(aSize) CC_PalMemMallocPlat(aSize)

This function allocates a memory buffer according to aSize.

Returns:

A pointer to the allocated buffer if successful.

NULL if failed.

#define CC_PalMemMove(aDestination, aSource, aSize) CC_PalMemMovePlat(aDestination, aSource, aSize)

This function moves aSize Bytes from the source buffer to the destination buffer. This function supports overlapped buffers.

Returns:

void.

#define CC_PalMemRealloc(aBuffer, aNewSize) CC_PalMemReallocPlat(aBuffer, aNewSize)

This function reallocates a memory buffer according to aNewSize. The contents of the old buffer is moved to the new location.

Returns:

A pointer to the newly-allocated buffer if successful.

NULL if failed.

#define CC_PalMemSet(aTarget, aChar, aSize) CC_PalMemSetPlat(aTarget, aChar, aSize)

This function sets aSize Bytes of aChar in the given buffer.

Returns:

void.

#define CC_PalMemSetZero(aTarget, aSize) CC_PalMemSetZeroPlat(aTarget, aSize)

This function sets aSize Bytes in the given buffer with zeroes.

Returns:

void.

CryptoCell PAL memory mapping APIs

Contains memory mapping functions.

Functions

- uint32_t *CC_PalMemMap* (*CCDmaAddr_t* physicalAddress, uint32_t mapSize, uint32_t **ppVirtBuffAddr)

  This function returns the base virtual address that maps the base physical address.

- uint32_t *CC_PalMemUnMap* (uint32_t *pVirtBuffAddr, uint32_t mapSize)

  This function unmaps a specified address range that was previously mapped by *CC_PalMemMap*.

Detailed description

Contains memory mapping functions.

Function documentation

uint32_t CC_PalMemMap (*CCDmaAddr_t* physicalAddress, uint32_t mapSize, uint32_t ** ppVirtBuffAddr)

This function returns the base virtual address that maps the base physical address.

Returns:

0 on success.

A non-zero value in case of failure.

Parameters:

I/O	Parameter	Description
in	`physicalAddress`	The starting physical address of the I/O range to be mapped.
in	`mapSize`	The number of Bytes that were mapped.
out	`ppVirtBuffAddr`	A pointer to the base virtual address to which the physical pages were mapped.

uint32_t CC_PalMemUnMap (uint32_t * pVirtBuffAddr, uint32_t mapSize)

This function unmaps a specified address range that was previously mapped by *CC_PalMemMap*.

Returns:

0 on success.

A non-zero value in case of failure.

Parameters:

I/O	Parameter	Description
in	`pVirtBuffAddr`	A pointer to the base virtual address to which the physical pages were mapped.
in	`mapSize`	The number of Bytes that were mapped.

CryptoCell PAL mutex APIs

Contains resource management functions.

Functions

- CCError_t *CC_PalMutexCreate* (CC_PalMutex *pMutexId)

  This function creates a mutex.

- CCError_t *CC_PalMutexDestroy* (CC_PalMutex *pMutexId)

  This function destroys a mutex.

- CCError_t *CC_PalMutexLock* (CC_PalMutex *pMutexId, uint32_t aTimeOut)

  This function waits for a mutex with aTimeOut. aTimeOut is specified in milliseconds. A value of aTimeOut=CC_INFINITE means that the function will not return.

- CCError_t *CC_PalMutexUnlock* (CC_PalMutex *pMutexId)

  This function releases the mutex.

Detailed description

Contains resource management functions.

Function documentation

CCError_t CC_PalMutexCreate (CC_PalMutex *   pMutexId)

This function creates a mutex.

Returns:

0   on success.

A non-zero value on failure.

Parameters:

I/O	Parameter	Description
out	`pMutexId`	A pointer to the handle of the created mutex.

CCError_t CC_PalMutexDestroy (CC_PalMutex *   pMutexId)

This function destroys a mutex.

Returns:

0   on success.

A non-zero value on failure.

Parameters:

I/O	Parameter	Description
in	`pMutexId`	A pointer to handle of the mutex to destroy.

CCError_t CC_PalMutexLock (CC_PalMutex *   pMutexId, uint32_t   aTimeOut)

This function waits for a mutex with aTimeOut. aTimeOut   is specified in milliseconds. A value of aTimeOut=CC_INFINITE   means that the function will not return.

Returns:

0   on success.

A non-zero value on failure.

Parameters:

I/O	Parameter	Description
in	`pMutexId`	A pointer to handle of the mutex.
in	`aTimeOut`	The timeout in mSec, or CC_INFINITE.

CCError_t CC_PalMutexUnlock (CC_PalMutex *   pMutexId)

This function releases the mutex.

Returns:

0   on success.

A non-zero value on failure.

Parameters:

I/O	Parameter	Description
in	`pMutexId`	A pointer to the handle of the mutex.

CryptoCell PAL platform-dependent definitions and types

Contains CryptoCell PAL platform-dependent definitions and types.

Macros

- #define *CC_SUCCESS* 0UL
- #define *CC_FAIL* 1UL
- #define *CC_OK* 0
- #define *CC_UNUSED_PARAM*(prm) ((void)prm)
- #define *CC_MAX_UINT32_VAL* (0xFFFFFFFF)
- #define *CC_MIN*(a, b) min(a , b)
- #define *CC_MAX*(a, b) max(a , b)
- #define *CALC_FULL_BYTES*(numBits) ((numBits)/*CC_BITS_IN_BYTE* + (((numBits) & (*CC_BITS_IN_BYTE*-1)) > 0))
- #define *CALC_FULL_32BIT_WORDS*(numBits) ((numBits)/*CC_BITS_IN_32BIT_WORD* + (((numBits) & (*CC_BITS_IN_32BIT_WORD*-1)) > 0))
- #define *CALC_32BIT_WORDS_FROM_BYTES*(sizeBytes) ((sizeBytes)/*CC_32BIT_WORD_SIZE* + (((sizeBytes) & (*CC_32BIT_WORD_SIZE*-1)) > 0))
- #define *CALC_32BIT_WORDS_FROM_64BIT_DWORD*(sizeWords) (sizeWords * *CC_32BIT_WORD_IN_64BIT_DWORD*)
- #define *ROUNDUP_BITS_TO_32BIT_WORD*(numBits) (*CALC_FULL_32BIT_WORDS*(numBits) * *CC_BITS_IN_32BIT_WORD*)
- #define *ROUNDUP_BITS_TO_BYTES*(numBits) (*CALC_FULL_BYTES*(numBits) * *CC_BITS_IN_BYTE*)
- #define *ROUNDUP_BYTES_TO_32BIT_WORD*(sizeBytes) (*CALC_32BIT_WORDS_FROM_BYTES*(sizeBytes) * *CC_32BIT_WORD_SIZE*)
- #define *CC_1K_SIZE_IN_BYTES* 1024
- #define *CC_BITS_IN_BYTE* 8
- #define *CC_BITS_IN_32BIT_WORD* 32
- #define *CC_32BIT_WORD_SIZE* 4
- #define *CC_32BIT_WORD_IN_64BIT_DWORD* 2

Enumerations

- enum *CCBool* { *CC_FALSE* = 0, *CC_TRUE* = 1 }

Detailed description

Contains CryptoCell PAL platform-dependent definitions and types.

Macro definition documentation

#define CALC_32BIT_WORDS_FROM_64BIT_DWORD(sizeWords) (sizeWords * *CC_32BIT_WORD_IN_64BIT_DWORD*)

This macro calculates the number of full 32-bit words from 64-bits dwords.

#define CALC_32BIT_WORDS_FROM_BYTES(sizeBytes) ((sizeBytes)/CC_32BIT_WORD_SIZE + (((sizeBytes) & (CC_32BIT_WORD_SIZE-1)) > 0))

This macro calculates the number of full 32-bit words from Bytes where three Bytes are one word.

#define CALC_FULL_32BIT_WORDS(numBits) ((numBits)/CC_BITS_IN_32BIT_WORD + (((numBits) & (CC_BITS_IN_32BIT_WORD-1)) > 0))

This macro calculates the number of full 32-bit words from bits where 31 bits are one word.

#define CALC_FULL_BYTES(numBits) ((numBits)/CC_BITS_IN_BYTE + (((numBits) & (CC_BITS_IN_BYTE-1)) > 0))

This macro calculates the number of full Bytes from bits, where seven bits are one Byte.

#define CC_1K_SIZE_IN_BYTES 1024

Definition of 1 KB in Bytes.

#define CC_32BIT_WORD_IN_64BIT_DWORD 2

Definition of number of 32-bits words in a 64-bits dword.

#define CC_32BIT_WORD_SIZE 4

Definition of number of Bytes in a 32-bits word.

#define CC_BITS_IN_32BIT_WORD 32

Definition of number of bits in a 32-bits word.

#define CC_BITS_IN_BYTE 8

Definition of number of bits in a Byte.

#define CC_FAIL 1UL

Failure definition.

#define CC_MAX(a, b) max(a , b)

Definition for maximal calculation.

#define CC_MAX_UINT32_VAL (0xFFFFFFFF)

The maximal uint32 value.

#define CC_MIN(a, b) min(a , b)

Definition for minimal calculation.

#define CC_OK 0

Success (OK) definition.

#define CC_SUCCESS 0UL

Success definition.

#define CC_UNUSED_PARAM(prm) ((void)prm)

Handles unused parameters in the code, to avoid compilation warnings.

#define ROUNDUP_BITS_TO_32BIT_WORD(numBits) (CALC_FULL_32BIT_WORDS(numBits) * CC_BITS_IN_32BIT_WORD)

This macro rounds up bits to 32-bit words.

#define ROUNDUP_BITS_TO_BYTES(numBits)   (CALC_FULL_BYTES(numBits) * CC_BITS_IN_BYTE)

This macro rounds up bits to Bytes.

#define ROUNDUP_BYTES_TO_32BIT_WORD(sizeBytes)   (CALC_32BIT_WORDS_FROM_BYTES(sizeBytes) * CC_32BIT_WORD_SIZE)

This macro rounds up Bytes to 32-bit words.

Enumeration type documentation

enum CCBool

Boolean types.

Enumerator:

Enum	Description
CC_FALSE	Boolean false definition.
CC_TRUE	Boolean true definition.

CryptoCell PAL definitions for Boot Services

Contains CryptoCell PAL Secure Boot definitions.

Typedefs

- typedef uint32_t *CCDmaAddr_t*
- typedef uint32_t *CCAddr_t*

Detailed description

Contains CryptoCell PAL Secure Boot definitions.

Typedef documentation

typedef uint32_t CCAddr_t

CryptocCell address types: 32 bits or 64 bits, according to platform.

typedef uint32_t CCDmaAddr_t

DMA address types: 32 bits or 64 bits, according to platform.

CryptoCell PAL power-management APIs

Contains PAL power-management APIs.

Functions

- void *CC_PalPowerSaveModeInit* (void)

  This function initiates an atomic counter.

- int32_t *CC_PalPowerSaveModeStatus* (void)

  This function returns the number of active registered CryptoCell operations.

- CCError_t *CC_PalPowerSaveModeSelect* (*CCBool* isPowerSaveMode)

  This function updates the atomic counter on each call to CryptoCell.

Detailed description

Contains PAL power-management APIs.

Function documentation

void CC_PalPowerSaveModeInit (void)

This function initiates an atomic counter.

Returns:

Void.

CCError_t CC_PalPowerSaveModeSelect (*CCBool* isPowerSaveMode)

This function updates the atomic counter on each call to CryptoCell.

On each call to CryptoCell, the counter is increased. At the end of each operation the counter is decreased. Once the counter is zero, an external callback is called.

Returns:

0 on success.

A non-zero value on failure.

Parameters:

I/O	Parameter	Description
in	`isPowerSaveMode`	● TRUE: CryptoCell is active. ● FALSE: CryptoCell is idle.

int32_t CC_PalPowerSaveModeStatus (void)

This function returns the number of active registered CryptoCell operations.

Returns:

The value of the atomic counter.

CryptoCell PAL TRNG APIs

Contains APIs for retrieving TRNG user parameters.

Data structures

- struct *CC_PalTrngParams_t*

Typedefs

- typedef struct *CC_PalTrngParams_t* *CC_PalTrngParams_t*

Functions

- CCError_t *CC_PalTrngParamGet* (*CC_PalTrngParams_t* *pTrngParams, size_t *pParamsSize)

  This function returns the TRNG user parameters.

Detailed description

Contains APIs for retrieving TRNG user parameters.

Typedef documentation

typedef struct *CC_PalTrngParams_t* *CC_PalTrngParams_t*

Definition for the structure of the random-generator parameters of CryptoCell, containing the user-given parameters.

Function documentation

CCError_t CC_PalTrngParamGet (*CC_PalTrngParams_t* * pTrngParams, size_t * pParamsSize)

This function returns the TRNG user parameters.

Returns:

0 on success.

A non-zero value on failure.

Parameters:

I/O	Parameter	Description
out	`pTrngParams`	A pointer to the TRNG user parameters.
in,out	`pParamsSize`	A pointer to the size of the TRNG-user-parameters structure used. • Input: the function must verify its size is the same as *CC_PalTrngParams_t*. • Output: the function returns the size of *CC_PalTrngParams_t* for library-size verification.

Specific errors of the CryptoCell PAL APIs

Contains platform-dependent PAL-API error definitions.

Macros

- #define *CC_PAL_BASE_ERROR* 0x0F000000
- #define *CC_PAL_MEM_BUF1_GREATER* *CC_PAL_BASE_ERROR* + 0x01UL
- #define *CC_PAL_MEM_BUF2_GREATER* *CC_PAL_BASE_ERROR* + 0x02UL
- #define *CC_PAL_SEM_CREATE_FAILED* *CC_PAL_BASE_ERROR* + 0x03UL
- #define *CC_PAL_SEM_DELETE_FAILED* *CC_PAL_BASE_ERROR* + 0x04UL
- #define *CC_PAL_SEM_WAIT_TIMEOUT* *CC_PAL_BASE_ERROR* + 0x05UL
- #define *CC_PAL_SEM_WAIT_FAILED* *CC_PAL_BASE_ERROR* + 0x06UL
- #define *CC_PAL_SEM_RELEASE_FAILED* *CC_PAL_BASE_ERROR* + 0x07UL
- #define *CC_PAL_ILLEGAL_ADDRESS* *CC_PAL_BASE_ERROR* + 0x08UL

Detailed description

Contains platform-dependent PAL-API error definitions.

Macro definition documentation

#define CC_PAL_BASE_ERROR 0x0F000000

The PAL error base.

#define CC_PAL_ILLEGAL_ADDRESS *CC_PAL_BASE_ERROR* + 0x08UL

Illegal PAL address.

#define CC_PAL_MEM_BUF1_GREATER _CC_PAL_BASE_ERROR + 0x01UL

Buffer 1 is greater than buffer 2 error.

#define CC_PAL_MEM_BUF2_GREATER _CC_PAL_BASE_ERROR + 0x02UL

Buffer 2 is greater than buffer 1 error.

#define CC_PAL_SEM_CREATE_FAILED _CC_PAL_BASE_ERROR + 0x03UL

Semaphore creation failed.

#define CC_PAL_SEM_DELETE_FAILED _CC_PAL_BASE_ERROR + 0x04UL

Semaphore deletion failed.

#define CC_PAL_SEM_RELEASE_FAILED _CC_PAL_BASE_ERROR + 0x07UL

Semaphore release failed.

#define CC_PAL_SEM_WAIT_FAILED _CC_PAL_BASE_ERROR + 0x06UL

Semaphore wait failed.

#define CC_PAL_SEM_WAIT_TIMEOUT _CC_PAL_BASE_ERROR + 0x05UL

Semaphore reached timeout.

1.5.17 CryptoCell PKA APIs

Contains all CryptoCell PKA APIs.

Modules

- *CryptoCell PKA-specific definitions*
- Contains the CryptoCell PKA API definitions. *CryptoCell PKA-API platform-dependent types and definitions*

Contains the platform-dependent definitions of the CryptoCell PKA APIs.

Detailed description

Contains all CryptoCell PKA APIs.

CryptoCell PKA-specific definitions

Contains the CryptoCell PKA API definitions.

Macros

- #define *CC_RSA_MAXIMUM_MOD_BUFFER_SIZE_IN_WORDS* ((*CC_RSA_MAX_VALID_KEY_SIZE_VALUE_IN_BITS* + *CC_PKA_WORD_SIZE_IN_BITS*) / *CC_BITS_IN_32BIT_WORD*)
- #define *CC_ECPKI_MODUL_MAX_LENGTH_IN_BITS* 521
- #define *CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS* 5
- #define *CC_PKA_ECPKI_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS* *CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS*

- #define *CC_PKA_MAXIMUM_MOD_BUFFER_SIZE_IN_WORDS* *CC_RSA_MAXIMUM_MOD_BUFFER_SIZE_IN_WORDS*
- #define *CC_PKA_PUB_KEY_BUFF_SIZE_IN_WORDS* (2**CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS*)
- #define *CC_PKA_PRIV_KEY_BUFF_SIZE_IN_WORDS* (2**CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS*)
- #define *CC_PKA_KGDATA_BUFF_SIZE_IN_WORDS* (3**CC_PKA_MAXIMUM_MOD_BUFFER_SIZE_IN_WORDS* + 3**CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS*)
- #define *CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS* 18
- #define *CC_ECPKI_ORDER_MAX_LENGTH_IN_WORDS* (*CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS* + 1)
- #define *CC_PKA_DOMAIN_BUFF_SIZE_IN_WORDS* (2**CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS*)
- #define *COUNT_NAF_WORDS_PER_KEY_WORD* 8
- #define *CC_PKA_ECDSA_NAF_BUFF_MAX_LENGTH_IN_WORDS* (*COUNT_NAF_WORDS_PER_KEY_WORD***CC_ECPKI_ORDER_MAX_LENGTH_IN_WORDS* + 1)
- #define *CC_PKA_ECPKI_SCALAR_MUL_BUFF_MAX_LENGTH_IN_WORDS* (*CC_PKA_ECDSA_NAF_BUFF_MAX_LENGTH_IN_WORDS*+*CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS*+2)
- #define *CC_PKA_ECPKI_BUILD_TMP_BUFF_MAX_LENGTH_IN_WORDS* (3**CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS*+*CC_PKA_ECPKI_SCALAR_MUL_BUFF_MAX_LENGTH_IN_WORDS*)
- #define *CC_PKA_ECDSA_SIGN_BUFF_MAX_LENGTH_IN_WORDS* (6**CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS*+*CC_PKA_ECPKI_SCALAR_MUL_BUFF_MAX_LENGTH_IN_WORDS*)
- #define *CC_PKA_ECDH_BUFF_MAX_LENGTH_IN_WORDS* (2**CC_ECPKI_ORDER_MAX_LENGTH_IN_WORDS* + *CC_PKA_ECPKI_SCALAR_MUL_BUFF_MAX_LENGTH_IN_WORDS*)
- #define *CC_PKA_KG_BUFF_MAX_LENGTH_IN_WORDS* (2**CC_ECPKI_ORDER_MAX_LENGTH_IN_WORDS* + *CC_PKA_ECPKI_SCALAR_MUL_BUFF_MAX_LENGTH_IN_WORDS*)
- #define *CC_PKA_ECDSA_VERIFY_BUFF_MAX_LENGTH_IN_WORDS* (3**CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS*)
- #define *CC_EC_MONT_EDW_MODULUS_MAX_SIZE_IN_BYTES* 32U
- #define *CC_EC_MONT_EDW_MODULUS_MAX_SIZE_IN_WORDS* 8U
- #define *CC_EC_MONT_TEMP_BUFF_SIZE_IN_32BIT_WORDS* (8 * *CC_EC_MONT_EDW_MODULUS_MAX_SIZE_IN_WORDS*)
- #define *CC_EC_EDW_TEMP_BUFF_SIZE_IN_32BIT_WORDS* (8**CC_EC_MONT_EDW_MODULUS_MAX_SIZE_IN_WORDS* + (sizeof(*CCHashUserContext_t*)+*CC_32BIT_WORD_SIZE*-1)/*CC_32BIT_WORD_SIZE*)

Detailed description

Contains the CryptoCell PKA API definitions.

Macro definition documentation

#define CC_EC_EDW_TEMP_BUFF_SIZE_IN_32BIT_WORDS (8*CC_EC_MONT_EDW_MODULUS_MAX_SIZE_IN_WORDS + (sizeof(CCHashUserContext_t)+CC_32BIT_WORD_SIZE-1)/CC_32BIT_WORD_SIZE)

The size of the ECC Edwards temporary buffer in words.

#define CC_EC_MONT_EDW_MODULUS_MAX_SIZE_IN_BYTES 32U

The maximal size of the modulus buffers for CC_EC_MONT and EC_EDW in Bytes.

#define CC_EC_MONT_EDW_MODULUS_MAX_SIZE_IN_WORDS 8U

The maximal size of the modulus buffers for CC_EC_MONT and EC_EDW in words.

#define CC_EC_MONT_TEMP_BUFF_SIZE_IN_32BIT_WORDS (8 * CC_EC_MONT_EDW_MODULUS_MAX_SIZE_IN_WORDS)

The size of the ECC Montgomery temporary buffer in words.

#define CC_ECPKI_MODUL_MAX_LENGTH_IN_BITS 521

The maximal EC modulus size.

#define CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS 18

The maximal size of the EC modulus in words.

#define CC_ECPKI_ORDER_MAX_LENGTH_IN_WORDS (CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS + 1)

The maximal size of the EC order in words.

#define CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS 5

The size of the buffers for Barrett modulus tag NP, used in PKI algorithms.

#define CC_PKA_DOMAIN_BUFF_SIZE_IN_WORDS (2*CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS)

The maximal size of the EC domain in words.

#define CC_PKA_ECDH_BUFF_MAX_LENGTH_IN_WORDS (2*CC_ECPKI_ORDER_MAX_LENGTH_IN_WORDS + CC_PKA_ECPKI_SCALAR_MUL_BUFF_MAX_LENGTH_IN_WORDS)

The size of the ECC ECDH temporary-buffer in words.

#define CC_PKA_ECDSA_NAF_BUFF_MAX_LENGTH_IN_WORDS (COUNT_NAF_WORDS_PER_KEY_WORD*CC_ECPKI_ORDER_MAX_LENGTH_IN_WORDS + 1)

The maximal length of the ECC NAF buffer.

#define CC_PKA_ECDSA_SIGN_BUFF_MAX_LENGTH_IN_WORDS (6*CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS+CC_PKA_ECPKI_SCALAR_MUL_BUFF_MAX_LENGTH_IN_WORDS)

The size of the ECC sign temporary buffer in words.

#define CC_PKA_ECDSA_VERIFY_BUFF_MAX_LENGTH_IN_WORDS (3*CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS)

The size of the ECC verify temporary-buffer in words.

#define CC_PKA_ECPKI_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS

The size of the buffers for Barrett modulus tag NP, used in ECC.

#define CC_PKA_ECPKI_BUILD_TMP_BUFF_MAX_LENGTH_IN_WORDS (3*CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS+CC_PKA_ECPKI_SCALAR_MUL_BUFF_MAX_LENGTH_IN_WORDS)

The size of the ECC temporary buffer in words.

#define CC_PKA_ECPKI_SCALAR_MUL_BUFF_MAX_LENGTH_IN_WORDS (CC_PKA_ECDSA_NAF_BUFF_MAX_LENGTH_IN_WORDS+CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS+2)

The size of the Scalar buffer in words.

#define CC_PKA_KG_BUFF_MAX_LENGTH_IN_WORDS (2*CC_ECPKI_ORDER_MAX_LENGTH_IN_WORDS + CC_PKA_ECPKI_SCALAR_MUL_BUFF_MAX_LENGTH_IN_WORDS)

The size of the PKA KG temporary-buffer in words.

#define CC_PKA_KGDATA_BUFF_SIZE_IN_WORDS (3*CC_PKA_MAXIMUM_MOD_BUFFER_SIZE_IN_WORDS + 3*CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS)

The maximal size of the PKA KG buffer in words

#define CC_PKA_MAXIMUM_MOD_BUFFER_SIZE_IN_WORDS CC_RSA_MAXIMUM_MOD_BUFFER_SIZE_IN_WORDS

The maximal size of the PKA modulus.

#define CC_PKA_PRIV_KEY_BUFF_SIZE_IN_WORDS (2*CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS)

The maximal size of the PKA private-key in words.

#define CC_PKA_PUB_KEY_BUFF_SIZE_IN_WORDS (2*CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS)

The maximal size of the PKA public-key in words.

#define CC_RSA_MAXIMUM_MOD_BUFFER_SIZE_IN_WORDS ((CC_RSA_MAX_VALID_KEY_SIZE_VALUE_IN_BITS + CC_PKA_WORD_SIZE_IN_BITS) / CC_BITS_IN_32BIT_WORD)

The maximal RSA modulus size.

#define COUNT_NAF_WORDS_PER_KEY_WORD 8

The ECC NAF buffer definitions.

CryptoCell PKA-API platform-dependent types and definitions

Contains the platform-dependent definitions of the CryptoCell PKA APIs.

Macros

- #define CC_PKA_WORD_SIZE_IN_BITS 64
- #define CC_SRP_MAX_MODULUS_SIZE_IN_BITS 3072
- #define CC_RSA_MAX_VALID_KEY_SIZE_VALUE_IN_BITS 4096
- #define CC_RSA_MAX_KEY_GENERATION_HW_SIZE_BITS 3072
- #define CC_RSA_MAX_VALID_KEY_SIZE_VALUE_IN_WORDS CC_RSA_MAX_VALID_KEY_SIZE_VALUE_IN_BITS / CC_BITS_IN_32BIT_WORD
- #define SB_CERT_RSA_KEY_SIZE_IN_BITS 3072UL
- #define SB_CERT_RSA_KEY_SIZE_IN_BYTES (SB_CERT_RSA_KEY_SIZE_IN_BITS/CC_BITS_IN_BYTE)
- #define SB_CERT_RSA_KEY_SIZE_IN_WORDS (SB_CERT_RSA_KEY_SIZE_IN_BITS/CC_BITS_IN_32BIT_WORD)
- #define PKA_EXTRA_BITS 8
- #define PKA_MAX_COUNT_OF_PHYS_MEM_REGS 32

Detailed description

Contains the platform-dependent definitions of the CryptoCell PKA APIs.

Macro definition documentation

#define CC_PKA_WORD_SIZE_IN_BITS 64

The size of the PKA engine word.

#define CC_RSA_MAX_KEY_GENERATION_HW_SIZE_BITS 3072

The maximal supported size of key-generation in RSA in bits.

#define CC_RSA_MAX_VALID_KEY_SIZE_VALUE_IN_BITS 4096

The maximal supported size of modulus in RSA in bits.

#define CC_RSA_MAX_VALID_KEY_SIZE_VALUE_IN_WORDS CC_RSA_MAX_VALID_KEY_SIZE_VALUE_IN_BITS / CC_BITS_IN_32BIT_WORD

The maximal supported size of modulus in RSA in words.

#define CC_SRP_MAX_MODULUS_SIZE_IN_BITS 3072

The maximal supported size of modulus in bits.

#define PKA_EXTRA_BITS 8

The maximal count of extra bits in PKA operations.

#define PKA_MAX_COUNT_OF_PHYS_MEM_REGS 32

The number of memory registers in PKA operations.

#define SB_CERT_RSA_KEY_SIZE_IN_BITS 3072UL

The size of the Secure Boot or Secure Debug certificate RSA public modulus key in bits.

#define SB_CERT_RSA_KEY_SIZE_IN_BYTES (*SB_CERT_RSA_KEY_SIZE_IN_BITS*/*CC_BITS_IN_BYTE*)

The size of the Secure Boot or Secure Debug certificate RSA public modulus key in Bytes.

#define SB_CERT_RSA_KEY_SIZE_IN_WORDS (*SB_CERT_RSA_KEY_SIZE_IN_BITS*/*CC_BITS_IN_32BIT_WORD*)

The size of the Secure Boot or Secure Debug certificate RSA public modulus key in words.

1.5.18 CryptoCell POLY APIs

Contains all CryptoCell POLY APIs.

Modules

- *Specific errors of the CryptoCell POLY APIs*

Contains the CryptoCell POLY-API error definitions. Macros

- #define *CC_POLY_KEY_SIZE_IN_WORDS* 8
- #define *CC_POLY_KEY_SIZE_IN_BYTES* (*CC_POLY_KEY_SIZE_IN_WORDS***CC_32BIT_WORD_SIZE*)
- #define *CC_POLY_MAC_SIZE_IN_WORDS* 4
- #define *CC_POLY_MAC_SIZE_IN_BYTES* (*CC_POLY_MAC_SIZE_IN_WORDS***CC_32BIT_WORD_SIZE*)

Typedefs

- typedef uint32_t *mbedtls_poly_mac*[*CC_POLY_MAC_SIZE_IN_WORDS*]
- typedef uint32_t *mbedtls_poly_key*[*CC_POLY_KEY_SIZE_IN_WORDS*]

Functions

- CIMPORT_C CCError_t *mbedtls_poly* (*mbedtls_poly_key* pKey, uint8_t *pDataIn, size_t dataInSize, *mbedtls_poly_mac* macRes)

  This function performs the POLY MAC Calculation.

Detailed description

Contains all CryptoCell POLY APIs.

Macro definition documentation

#define CC_POLY_KEY_SIZE_IN_BYTES (*CC_POLY_KEY_SIZE_IN_WORDS*CC_32BIT_WORD_SIZE*)**

The size of the POLY key in Bytes.

#define CC_POLY_KEY_SIZE_IN_WORDS 8

The size of the POLY key in words.

#define CC_POLY_MAC_SIZE_IN_BYTES (*CC_POLY_MAC_SIZE_IN_WORDS*CC_32BIT_WORD_SIZE*)**

The size of the POLY MAC in Bytes.

#define CC_POLY_MAC_SIZE_IN_WORDS 4

The size of the POLY MAC in words.

Typedef documentation

typedef uint32_t mbedtls_poly_key[*CC_POLY_KEY_SIZE_IN_WORDS*]

The definition of the ChaCha-key buffer.

typedef uint32_t mbedtls_poly_mac[*CC_POLY_MAC_SIZE_IN_WORDS*]

The definition of the ChaCha-MAC buffer.

Function documentation

CIMPORT_C CCError_t mbedtls_poly (*mbedtls_poly_key* pKey, uint8_t * pDataIn, size_t dataInSize, *mbedtls_poly_mac* macRes)

This function performs the POLY MAC Calculation.

Returns:

CC_OK on success.

A non-zero value on failure, as defined in *mbedtls_cc_poly_error.h*.

Parameters:

I/O	Parameter	Description
in	`pKey`	A pointer to the key buffer of the user.
in	`pDataIn`	A pointer to the buffer of the input data to the ChaCha. Must not be null.
in	`dataInSize`	The size of the input data. Must not be zero.
in,out	`macRes`	A pointer to the MAC-result buffer.

Specific errors of the CryptoCell POLY APIs

Contains the CryptoCell POLY-API error definitions.

Macros

- #define *CC_POLY_KEY_INVALID_ERROR* (*CC_POLY_MODULE_ERROR_BASE* + 0x01UL)
- #define *CC_POLY_DATA_INVALID_ERROR* (*CC_POLY_MODULE_ERROR_BASE* + 0x02UL)
- #define *CC_POLY_DATA_SIZE_INVALID_ERROR* (*CC_POLY_MODULE_ERROR_BASE* + 0x03UL)
- #define *CC_POLY_RESOURCES_ERROR* (*CC_POLY_MODULE_ERROR_BASE* + 0x04UL)

Detailed description

Contains the CryptoCell POLY-API error definitions.

Macro definition documentation

#define CC_POLY_DATA_INVALID_ERROR (*CC_POLY_MODULE_ERROR_BASE* + 0x02UL)

Invalid input data.

#define CC_POLY_DATA_SIZE_INVALID_ERROR (*CC_POLY_MODULE_ERROR_BASE* + 0x03UL)

Illegal input data size.

#define CC_POLY_KEY_INVALID_ERROR (*CC_POLY_MODULE_ERROR_BASE* + 0x01UL)

Invalid key.

#define CC_POLY_RESOURCES_ERROR (*CC_POLY_MODULE_ERROR_BASE* + 0x04UL)

MAC calculation error.

1.5.19 CryptoCell production-library APIs

Contains CryptoCell production-library APIs.

Modules

- *CryptoCell production-library definitions*
- *CryptoCell ICV production library APIs*

  Contains CryptoCell ICV production library APIs.
- *CryptoCell OEM production library APIs*

  Contains CryptoCell OEM production library APIs.
- *Specific errors of the CryptoCell production-library APIs*

  Contains the CryptoCell production-library-API error definitions.

Detailed description

Contains CryptoCell production-library APIs.

CryptoCell production-library definitions

Data structures

- union *CCAssetBuff_t*

The asset buffer. Macros

- #define *CC_PROD_32BIT_WORD_SIZE* sizeof(uint32_t)
- #define *PROD_ASSET_SIZE* 16
- #define *PROD_ASSET_PKG_SIZE* 64
- #define *PROD_ASSET_PKG_WORD_SIZE* (*PROD_ASSET_PKG_SIZE*/*CC_PROD_32BIT_WORD_SIZE*)
- #define *PROD_DCU_LOCK_WORD_SIZE* 4

Typedefs

- typedef uint8_t *CCPlainAsset_t*[*PROD_ASSET_SIZE*]
- typedef uint32_t *CCAssetPkg_t*[*PROD_ASSET_PKG_WORD_SIZE*]

Enumerations

- enum *CCAssetType_t* { *ASSET_NO_KEY* = 0, *ASSET_PLAIN_KEY* = 1, *ASSET_PKG_KEY* = 2, *ASSET_TYPE_RESERVED* = 0x7FFFFFFF }

Detailed description

Macro definition documentation

#define CC_PROD_32BIT_WORD_SIZE   sizeof(uint32_t)

The definition of the number of Bytes in a word.

#define PROD_ASSET_PKG_SIZE   64

The size of the asset-package in Bytes.

#define PROD_ASSET_PKG_WORD_SIZE   (*PROD_ASSET_PKG_SIZE*/*CC_PROD_32BIT_WORD_SIZE*)

The size of the asset-package in words.

#define PROD_ASSET_SIZE   16

The size of the plain-asset in Bytes.

#define PROD_DCU_LOCK_WORD_SIZE   4

The number of words of the DCU LOCK.

Typedef documentation

typedef uint32_t CCAssetPkg_t[*PROD_ASSET_PKG_WORD_SIZE*]

Defines the buffer of the asset-package. A 64-Byte array.

typedef uint8_t CCPlainAsset_t[*PROD_ASSET_SIZE*]

Defines the buffer of the plain asset. A 16-Byte array.

Enumeration type documentation

enum *CCAssetType_t*

The type of the provided asset.

Enumerator:

Enum	Description
ASSET_NO_KEY	The asset is not provided.
ASSET_PLAIN_KEY	The asset is provided as plain, not in a package.
ASSET_PKG_KEY	The asset is provided as a package.
ASSET_TYPE_RESERVED	Reserved.

CryptoCell ICV production library APIs

Contains CryptoCell ICV production library APIs.

Data structures

- union *CCCmpuUniqueBuff_t*
- The device use of the unique buffer. struct *CCCmpuData_t*

Macros

- #define *CMPU_WORKSPACE_MINIMUM_SIZE*   4096
- #define *PROD_UNIQUE_BUFF_SIZE*   16

Enumerations

- enum *CCCmpuUniqueDataType_t* { *CMPU_UNIQUE_IS_HBK0* = 1, *CMPU_UNIQUE_IS_USER_DATA* = 2, *CMPU_UNIQUE_RESERVED* = 0x7FFFFFFF }

Functions

- CIMPORT_C CCError_t *CCProd_Cmpu* (unsigned long ccHwRegBaseAddr, *CCCmpuData_t* *pCmpuData, unsigned long workspaceBaseAddr, uint32_t workspaceSize)

  This function burns all ICV assets into the OTP of the device.

Detailed description

Contains CryptoCell ICV production library APIs.

Macro definition documentation

#define CMPU_WORKSPACE_MINIMUM_SIZE   4096

The size of the ICV production library workspace in Bytes, needed by the library for internal use.

#define PROD_UNIQUE_BUFF_SIZE   16

The size of the ICV production library unique buffer in Bytes: Hbk0 or user data.

Enumeration type documentation

enum *CCCmpuUniqueDataType_t*

The unique data type.

Enumerator:

Enum	Description
`CMPU_UNIQUE_IS_HBK0`	The device uses the unique data as Hbk0.
`CMPU_UNIQUE_IS_USER_DATA`	The device uses the unique data as a random value. Hbk0 is not used for the device.
`CMPU_UNIQUE_RESERVED`	Reserved.

Function documentation

CIMPORT_C CCError_t CCProd_Cmpu (unsigned long   ccHwRegBaseAddr, *CCCmpuData_t* *   pCmpuData, unsigned long   workspaceBaseAddr, uint32_t workspaceSize)

This function burns all ICV assets into the OTP of the device.

The user must perform a power-on-reset (PoR) to trigger LCS change to DM LCS.

Returns:

CC_OK   on success.

A non-zero value from *cc_prod_error.h* on failure.

Parameters:

I/O	Parameter	Description
in	`ccHwRegBaseAddr`	The base address of CrytoCell HW register.
in	`pCmpuData`	A pointer to the ICV defines structure.
in	`workspaceBaseAddr`	The base address of the workspace for internal use.
in	`workspaceSize`	The size of the provided workspace. Must be at least *CMPU_WORKSPACE_MINIMUM_SIZE*.

CryptoCell OEM production library APIs

Contains CryptoCell OEM production library APIs.

Data structures

- union *CCDmpuHbkBuff_t*
- The device use of the Hbk buffer. struct *CCDmpuData_t*

Macros

- #define *DMPU_WORKSPACE_MINIMUM_SIZE* 1536
- #define *DMPU_HBK1_SIZE_IN_WORDS* 4
- #define *DMPU_HBK_SIZE_IN_WORDS* 8

Enumerations

- enum *CCDmpuHBKType_t* { *DMPU_HBK_TYPE_HBK1* = 1, *DMPU_HBK_TYPE_HBK* = 2, *DMPU_HBK_TYPE_RESERVED* = 0x7FFFFFFF }

Functions

- CIMPORT_C CCError_t *CCProd_Dmpu* (unsigned long ccHwRegBaseAddr, *CCDmpuData_t* *pDmpuData, unsigned long workspaceBaseAddr, uint32_t workspaceSize)

  This function burns all OEM assets into the OTP of the device.

Detailed description

Contains CryptoCell OEM production library APIs.

Macro definition documentation

#define DMPU_HBK1_SIZE_IN_WORDS 4

The size of the Hbk1 buffer in words.

#define DMPU_HBK_SIZE_IN_WORDS 8

The size of the Hbk buffer in words.

#define DMPU_WORKSPACE_MINIMUM_SIZE 1536

The size of the OEM production library workspace in Bytes, needed by the library for internal use.

Enumeration type documentation

enum *CCDmpuHBKType_t*

The type of the unique data.

Enumerator:

Enum	Description
DMPU_HBK_TYPE_HBK1	The device uses Hbk1.
DMPU_HBK_TYPE_HBK	The device uses a full Hbk.
DMPU_HBK_TYPE_RESERVED	Reserved.

Function documentation

CIMPORT_C CCError_t CCProd_Dmpu (unsigned long ccHwRegBaseAddr, *CCDmpuData_t* * pDmpuData, unsigned long workspaceBaseAddr, uint32_t workspaceSize)

This function burns all OEM assets into the OTP of the device.

The user must perform a power-on-reset (PoR) to trigger LCS change to Secure.

Returns:

CC_OK on success.

A non-zero value from *cc_prod_error.h* on failure.

Parameters:

I/O	Parameter	Description
in	`ccHwRegBaseAddr`	The base address of CrytoCell HW registers.
in	`pDmpuData`	A pointer to the defines structure of the OEM.
in	`workspaceBaseAddr`	The base address of the workspace for internal use.
in	`workspaceSize`	The size of provided workspace. Must be at least DMPU_WORKSPACE_MINIMUM_SIZE.

Specific errors of the CryptoCell production-library APIs

Contains the CryptoCell production-library-API error definitions.

Macros

- #define *CC_PROD_INIT_ERR* (*CC_PROD_MODULE_ERROR_BASE* + 0x01UL)
- #define *CC_PROD_INVALID_PARAM_ERR* (*CC_PROD_MODULE_ERROR_BASE* + 0x02UL)
- #define *CC_PROD_ILLEGAL_ZERO_COUNT_ERR* (*CC_PROD_MODULE_ERROR_BASE* + 0x03UL)
- #define *CC_PROD_ILLEGAL_LCS_ERR* (*CC_PROD_MODULE_ERROR_BASE* + 0x04UL)
- #define *CC_PROD_ASSET_PKG_PARAM_ERR* (*CC_PROD_MODULE_ERROR_BASE* + 0x05UL)
- #define *CC_PROD_ASSET_PKG_VERIFY_ERR* (*CC_PROD_MODULE_ERROR_BASE* + 0x06UL)
- #define *CC_PROD_HAL_FATAL_ERR* (*CC_PROD_MODULE_ERROR_BASE* + 0x07UL)

Detailed description

Contains the CryptoCell production-library-API error definitions.

Macro definition documentation

#define CC_PROD_ASSET_PKG_PARAM_ERR (*CC_PROD_MODULE_ERROR_BASE* + 0x05UL)

Invalid asset-package fields.

#define CC_PROD_ASSET_PKG_VERIFY_ERR (*CC_PROD_MODULE_ERROR_BASE* + 0x06UL)

Failed to validate the asset package.

#define CC_PROD_HAL_FATAL_ERR (*CC_PROD_MODULE_ERROR_BASE* + 0x07UL)

HAL Fatal error occured.

#define CC_PROD_ILLEGAL_LCS_ERR (*CC_PROD_MODULE_ERROR_BASE* + 0x04UL)

LCS is invalid for the operation.

#define CC_PROD_ILLEGAL_ZERO_COUNT_ERR (*CC_PROD_MODULE_ERROR_BASE* + 0x03UL)

Invalid number of zeroes calculated.

#define CC_PROD_INIT_ERR (*CC_PROD_MODULE_ERROR_BASE* + 0x01UL)

Library initialization failure.

#define CC_PROD_INVALID_PARAM_ERR (*CC_PROD_MODULE_ERROR_BASE* + 0x02UL)

Illegal parameter.

1.5.20 CryptoCell random-number generation APIs.

Contains the CryptoCell random-number generation APIs.

Data structures

- struct *CCRndWorkBuff_t*
- struct *CCRndState_t*
- The structure for the RND state. struct *CCRndContext_t*

Macros

- #define *CC_RND_SEED_MAX_SIZE_WORDS* 12
- #define *CC_RND_MAX_GEN_VECTOR_SIZE_BITS* 0x7FFFF
- #define *CC_RND_MAX_GEN_VECTOR_SIZE_BYTES* 0xFFFF
- #define *CC_RND_REQUESTED_SIZE_COUNTER* 0x3FFFF
- #define *CC_RND_WORK_BUFFER_SIZE_WORDS* 1528
- #define *CC_RND_TRNG_SRC_INNER_OFFSET_WORDS* 2
- #define *CC_RND_TRNG_SRC_INNER_OFFSET_BYTES* (*CC_RND_TRNG_SRC_INNER_OFFSET_WORDS**sizeof(uint32_t))

Typedefs

- typedef int(* *CCRndGenerateVectWorkFunc_t*) (void *rndState_ptr, unsigned char *out_ptr, size_t outSizeBytes)

Enumerations

- enum *CCRndMode_t* { *CC_RND_FE* = 1, CC_RND_ModeLast = 0x7FFFFFFF }

Functions

- CCError_t *CC_RndSetGenerateVectorFunc* (*CCRndContext_t* *rndContext_ptr, *CCRndGenerateVectWorkFunc_t* rndGenerateVectFunc)

  This function sets the RND vector-generation function into the RND context.

Detailed description

Contains the CryptoCell random-number generation APIs.

Macro definition documentation

#define CC_RND_MAX_GEN_VECTOR_SIZE_BITS   0x7FFFF

The maximal size of the generated vector in bits.

#define CC_RND_MAX_GEN_VECTOR_SIZE_BYTES   0xFFFF

The maximal size of the generated random vector in Bytes.

#define CC_RND_REQUESTED_SIZE_COUNTER   0x3FFFF

The maximal size of the generated vector in Bytes.

#define CC_RND_SEED_MAX_SIZE_WORDS   12

The maximal size of the random seed in words.

#define CC_RND_TRNG_SRC_INNER_OFFSET_BYTES   (*CC_RND_TRNG_SRC_INNER_OFFSET_WORDS**sizeof(uint32_t))

The definition of the internal offset in Bytes.

#define CC_RND_TRNG_SRC_INNER_OFFSET_WORDS   2

The definition of the internal offset in words.

#define CC_RND_WORK_BUFFER_SIZE_WORDS   1528

The size of the temporary buffer in words.

Typedef documentation

typedef int(* CCRndGenerateVectWorkFunc_t) (void *rndState_ptr, unsigned char *out_ptr, size_t outSizeBytes)

The RND vector-generation function pointer.

Enumeration type documentation

enum *CCRndMode_t*

The definition of the random operation modes.

Enumerator:

Enum	Description
`CC_RND_FE`	HW entropy estimation: 80090B or full.

Function documentation

CCError_t CC_RndSetGenerateVectorFunc (CCRndContext_t * rndContext_ptr, CCRndGenerateVectWorkFunc_t rndGenerateVectFunc)

This function sets the RND vector-generation function into the RND context.

It is called as part of Arm TrustZone CryptoCell library initialization, to set the RND vector generation function into the primary RND context.

Note

It must be called before any other API that requires the RND context as parameter.

Returns:

CC_OK on success.

A non-zero value from cc_rnd_error.h on failure.

Parameters:

I/O	Parameter	Description
in,out	`rndContext_ptr`	A pointer to the RND context buffer allocated by the user, which is used to maintain the RND state, as well as pointers to the functions used for random vector generation.
in	`rndGenerateVectFunc`	A pointer to the CC_RndGenerateVector random-vector-generation function.

1.5.21 CryptoCell RNG APIs

The Random Number Generator (RNG) module supports random number generation, as defined in NIST SP 800-90A: Recommendation for Random Number Generation Using Deterministic Random Bit Generators. See *mbedtls_ctr_drbg_random()*.

The Random Number Generator (RNG) module supports random number generation, as defined in NIST SP 800-90A: Recommendation for Random Number Generation Using Deterministic Random Bit Generators. See *mbedtls_ctr_drbg_random()*.

The block-cipher counter-mode based deterministic random-bit generator (CTR_DBRG). CryptoCell provides the source of entropy.

For the implementation of RNG, see *ctr_drbg.h*.

1.5.22 CryptoCell runtime-library asset-provisioning APIs

Contains CryptoCell runtime-library ICV and OEM asset-provisioning APIs and definitions.

Macros

- #define *CC_ASSET_PROV_MAX_ASSET_PKG_SIZE* 560

Enumerations

- enum *CCAssetProvKeyType_t* { *ASSET_PROV_KEY_TYPE_KPICV* = 1, *ASSET_PROV_KEY_TYPE_KCP* = 2, *ASSET_PROV_KEY_TYPE_RESERVED* = 0x7FFFFFFF }

Functions

- CCError_t *mbedtls_util_asset_pkg_unpack* (*CCAssetProvKeyType_t* keyType, uint32_t assetId, uint32_t *pAssetPackage, size_t assetPackageLen, uint8_t *pAssetData, size_t *pAssetDataLen)

  This API securely provisions ICV or OEM assets to devices, using CryptoCell.

Detailed description

Contains CryptoCell runtime-library ICV and OEM asset-provisioning APIs and definitions.

Macro definition documentation

#define CC_ASSET_PROV_MAX_ASSET_PKG_SIZE 560

The maximal size of an asset package.

Enumeration type documentation

enum *CCAssetProvKeyType_t*

The type of key used to pack the asset.

Enumerator:

Enum	Description
ASSET_PROV_KEY_TYPE_KPICV	ICV: The Kpicv key was used to pack the asset.
ASSET_PROV_KEY_TYPE_KCP	OEM: The Kcp key was used to pack the asset.
ASSET_PROV_KEY_TYPE_RESERVED	Reserved.

Function documentation

CCError_t mbedtls_util_asset_pkg_unpack (*CCAssetProvKeyType_t* keyType, uint32_t assetId, uint32_t * pAssetPackage, size_t assetPackageLen, uint8_t * pAssetData, size_t * pAssetDataLen)

This API securely provisions ICV or OEM assets to devices, using CryptoCell.

It takes an encrypted and authenticated asset package produced by the ICV or OEM asset-packaging offline utility (using AES-CCM with key derived from Kpicv or Kcp respectively, and the asset identifier), authenticates and decrypts it. The decrypted asset data is returned to the caller.

Note

The function is valid in all LCSes. However, an error is returned if the requested key is locked.

Returns:

CC_UTIL_OK on success.

A non-zero value on failure as defined in *cc_util_error.h*.

Parameters:

I/O	Parameter	Description
	keyType	The type of key used to pack the asset.
	assetId	A 32-bit index identifying the asset, in big-endian order.
	pAssetPackage	The encrypted and authenticated asset package.
	assetPackageLen	The length of the asset package. Must not exceed CC_ASSET_PROV_MAX_ASSET_PKG_SIZE.
	pAssetData	The buffer for retrieving the decrypted asset data.
	pAssetDataLen	• In: The size of the available asset-data buffer. Maximal size is 512 bytes. • Out: A pointer to the actual length of the decrypted asset data.

1.5.23 CryptoCell RSA APIs

RSA is an asymmetric algorithm used for secure-data transmittion.

CryptoCell-312 hardware limitations for RSA

CryptoCell-312 supports the following RSA key sizes for private-public operations:

- 256 Bytes (2048 bits).
- 384 Bytes (3071 bits).
- 512 Bytes (4096 bits).

For key-generation, CryptoCell-312 supports the following RSA key sizes:

- 256 Bytes (2048 bits).
- 384 Bytes (3071 bits).

Typical usage of RSA in CryptoCell-312

The following is a typical RSA operation flow:

1. *mbedtls_rsa_gen_key()*.
2. *mbedtls_rsa_pkcs1_encrypt()*.

Modules

- *CryptoCell-312 hardware limitations for RSA*
- *Typical usage of RSA in CryptoCell-312*

Detailed description

RSA is an asymmetric algorithm used for secure-data transmittion.

Note

As it is considered slow, it is mainly used to pass encrypted shared keys for symmetric key cryptography.

The RSA module implements the standards defined in Public-Key Cryptography Standards (PKCS) #1 v1.5: RSA Encryption and Public-Key Cryptography Standards (PKCS) #1 v2.1: RSA Cryptography Specifications.

For the implementation of RSA, see *rsa.h*

1.5.24 CryptoCell Secure Boot certificate-chain-processing APIs.

Contains CryptoCell Secure Boot certificate-chain-processing APIs.

Functions

- CCError_t *mbedtls_sb_cert_chain_cerification_init* (*CCSbCertInfo_t* *certPkgInfo)

  This function initializes the Secure Boot certificate-chain processing.
- CCError_t *mbedtls_sb_cert_verify_single* (*CCSbFlashReadFunc* flashReadFunc, void *userContext, *CCAddr_t* certStoreAddress, *CCSbCertInfo_t* *pCertPkgInfo, uint32_t *pHeader, uint32_t headerSize, uint32_t *pWorkspace, uint32_t workspaceSize)

  This function verifies a single certificate package containing either a key or content certificate.
- CCError_t *mbedtls_sb_sw_image_store_address_change* (uint32_t *pCert, uint32_t maxCertSizeWords, *CCAddr_t* address, uint32_t indexOfAddress)

  This function changes the storage address of a specific SW image in the content certificate.

Detailed description

Contains CryptoCell Secure Boot certificate-chain-processing APIs.

Function documentation

CCError_t mbedtls_sb_cert_chain_cerification_init (*CCSbCertInfo_t* * certPkgInfo)

This function initializes the Secure Boot certificate-chain processing.

It initializes the internal data fields of the certificate package.

Note

This function must be the first API called when processing a Secure Boot certificate chain.

Returns:

CC_OK on success.

A non-zero value from sbrom_bsv_error.h on failure.

Parameters:

I/O	Parameter	Description
in,out	`certPkgInfo`	A pointer to the information about the certificate package.

CCError_t mbedtls_sb_cert_verify_single (*CCSbFlashReadFunc* flashReadFunc, void * userContext, *CCAddr_t* certStoreAddress, *CCSbCertInfo_t* * pCertPkgInfo, uint32_t * pHeader, uint32_t headerSize, uint32_t * pWorkspace, uint32_t workspaceSize)

This function verifies a single certificate package containing either a key or content certificate.

It verifies the following:

- The public key, as saved in the certificate, against its hash, found in either the OTP memory (HBK) or in certPkgInfo.
- The RSA signature of the certificate.
- The SW version in the certificate is higher than or equal to the minimal SW version, as recorded on the device and passed in certPkgInfo.
- For content certificates: Each SW module against its hash in the certificate.

Note

The certificates may reside in the memory or in the flash. flashReadFunc() must be implemented accordingly.

Certificates and images must both be placed either in the memory, or in the flash.

Returns:

CC_OK on success.

A non-zero value from bsv_error.h on failure.

Parameters:

I/O	Parameter	Description
in	`flashReadFunc`	A pointer to the flash-read function.
in	`userContext`	An additional pointer for flashRead() usage. May be NULL.
in	`certStoreAddress`	The address where the certificate is located. This address is provided to flashReadFunc.
in,out	`pCertPkgInfo`	A pointer to the certificate-package information.
in,out	`pHeader`	A pointer to a buffer used for extracting the X.509 TBS Headers. **Note** Must be NULL for proprietary certificates.
in	`headerSize`	The size of pHeader in Bytes. **Note** Must be 0 for proprietary certificates.
in	`pWorkspace`	Buffer for the internal use of the function.

I/O	Parameter	Description
in	`workspaceSize`	The size of the workspace in Bytes. **Note** Must be at least *CC_SB_MIN_WORKSPACE_SIZE_IN_BYTES*.

CCError_t mbedtls_sb_sw_image_store_address_change (uint32_t * pCert, uint32_t maxCertSizeWords, *CCAddr_t* address, uint32_t indexOfAddress)

This function changes the storage address of a specific SW image in the content certificate.

Note

The certificate must be loaded to the RAM before calling this function.

The function does not verify the certificate before the address change.

Returns:

CC_OK on success.

A non-zero value from bsv_error.h on failure.

Parameters:

I/O	Parameter	Description
in	`pCert`	The certificate address, after it has been loaded to memory.
in	`maxCertSizeWords`	The certificate boundaries, that is, the maximal memory size allocated for the certificate in words.
in	`address`	The new storage address to change to.
in	`indexOfAddress`	The index of the SW image in the content certificate, starting from 0.

1.5.25 CryptoCell Secure Boot and Secure Debug APIs.

Contains all Secure Boot and Secure Debug APIs and definitions.

Modules

- *CryptoCell Secure Boot and Secure Debug API definitions*
- Contains definitions used for the Secure Boot and Secure Debug APIs. *CryptoCell Secure Boot basic type definitions*
- Contains CryptoCell Secure Boot basic type definitions. *CryptoCell Secure Boot definitions*
- Contains CryptoCell Secure Boot type definitions. *CryptoCell Secure Boot and Secure Debug definitions and structures*

Contains CryptoCell Secure Boot and Secure Debug definitions and structures.

Detailed description

Contains all Secure Boot and Secure Debug APIs and definitions.

CryptoCell Secure Boot and Secure Debug API definitions

Contains definitions used for the Secure Boot and Secure Debug APIs.

Macros

- #define CC_SB_MAX_NUM_OF_IMAGES 16
- #define CC_SB_MAX_CERT_SIZE_IN_BYTES (0xB10)
- #define CC_SB_MAX_CERT_SIZE_IN_WORDS (CC_SB_MAX_CERT_SIZE_IN_BYTES/CC_32BIT_WORD_SIZE)
- #define CC_SB_MIN_DBG_WORKSPACE_SIZE_IN_BYTES (0x350)
- #define CC_SB_MIN_WORKSPACE_SIZE_IN_BYTES (CC_SB_MAX_CERT_SIZE_IN_BYTES + CC_MAX(CC_SB_MIN_DBG_WORKSPACE_SIZE_IN_BYTES, CC_DOUBLE_BUFFER_MAX_SIZE_IN_BYTES))

  The minimal size of the Secure Boot workspace in Bytes.

Detailed description

Contains definitions used for the Secure Boot and Secure Debug APIs.

Macro definition documentation

#define CC_SB_MAX_CERT_SIZE_IN_BYTES (0xB10)

The maximal size of a certificate in Bytes.

#define CC_SB_MAX_CERT_SIZE_IN_WORDS (CC_SB_MAX_CERT_SIZE_IN_BYTES/CC_32BIT_WORD_SIZE)

The maximal size of a certificate in words.

#define CC_SB_MAX_NUM_OF_IMAGES 16

The maximal number of SW images per content certificate.

#define CC_SB_MIN_DBG_WORKSPACE_SIZE_IN_BYTES (0x350)

The size of the Secure Debug workspace in Bytes. This workspace is used to store the RSA parameters, for example, modulus and signature.

#define CC_SB_MIN_WORKSPACE_SIZE_IN_BYTES (CC_SB_MAX_CERT_SIZE_IN_BYTES + CC_MAX(CC_SB_MIN_DBG_WORKSPACE_SIZE_IN_BYTES, CC_DOUBLE_BUFFER_MAX_SIZE_IN_BYTES))

The minimal size of the Secure Boot workspace in Bytes.

The Secure Boot APIs use a temporary workspace for processing the data that is read from the flash, before loading the SW modules to their designated memory addresses. This workspace must be large enough to accommodate the size of the certificates, and twice the size of the data that is read from flash in each processing round. The definition of CC_SB_MIN_WORKSPACE_SIZE_IN_BYTES is comprised of CC_DOUBLE_BUFFER_MAX_SIZE_IN_BYTES and additional space for the certificate itself, which resides in the workspace at the same time the SW images data is processed.

It is assumed that the optimal size of the data to read in each processing round is 4KB, based on the standard flash-memory page size. Therefore, the size of the double buffer, CC_CONFIG_SB_DOUBLE_BUFFER_MAX_SIZE_IN_BYTES , is defined by default as 8KB in the project configuration file. This can be changed to accommodate the optimal value in different environments. CC_DOUBLE_BUFFER_MAX_SIZE_IN_BYTES is defined by the Boot Services makefile as equal to CC_CONFIG_SB_DOUBLE_BUFFER_MAX_SIZE_IN_BYTES.

Note

When writing code that uses the Secure Boot APIs, and includes the *bootimagesverifier_def.h* file, the value of CC_DOUBLE_BUFFER_MAX_SIZE_IN_BYTES must be defined by your makefile to be exactly the same value as was used when compiling the SBROM library, and CC_SB_X509_CERT_SUPPORTED must be defined in the Makefile, according to the definition of CC_CONFIG_SB_X509_CERT_SUPPORTED.

The size of CC_DOUBLE_BUFFER_MAX_SIZE_IN_BYTES must be a multiple of the hash SHA-256 block size of 64 Bytes.

CryptoCell Secure Boot basic type definitions

Contains CryptoCell Secure Boot basic type definitions.

Contains CryptoCell Secure Boot basic type definitions.

CryptoCell Secure Boot definitions

Contains CryptoCell Secure Boot type definitions.

Data structures

- struct *CCSbCertInfo_t*

Macros

- #define *SW_REC_SIGNED_DATA_SIZE_IN_BYTES* 44
- #define *SW_REC_NONE_SIGNED_DATA_SIZE_IN_BYTES* 8
- #define *SW_REC_NONE_SIGNED_DATA_SIZE_IN_WORDS* *SW_REC_NONE_SIGNED_DATA_SIZE_IN_BYTES*/*CC_32BIT_WORD_SIZE*
- #define *CC_SW_COMP_NO_MEM_LOAD_INDICATION* 0xFFFFFFFFUL

Detailed description

Contains CryptoCell Secure Boot type definitions.

Macro definition documentation

#define CC_SW_COMP_NO_MEM_LOAD_INDICATION 0xFFFFFFFFUL

Indication whether or not to load the SW image to memory.

#define SW_REC_NONE_SIGNED_DATA_SIZE_IN_BYTES 8

The size of the additional-data of the SW-image certificate in Bytes.

#define SW_REC_NONE_SIGNED_DATA_SIZE_IN_WORDS *SW_REC_NONE_SIGNED_DATA_SIZE_IN_BYTES*/*CC_32BIT_WORD_SIZE*

The size of the additional-data of the SW-image certificate in words.

#define SW_REC_SIGNED_DATA_SIZE_IN_BYTES 44

The size of the data of the SW-image certificate.

CryptoCell Secure Boot and Secure Debug definitions and structures

Contains CryptoCell Secure Boot and Secure Debug definitions and structures.

Macros

- #define *CC_SB_MAX_SIZE_ADDITIONAL_DATA_BYTES* 128

Typedefs

- typedef uint32_t *CCSbCertPubKeyHash_t*[HASH_RESULT_SIZE_IN_WORDS]
- typedef uint32_t *CCSbCertSocId_t*[HASH_RESULT_SIZE_IN_WORDS]
- typedef uint32_t(* *CCSbFlashReadFunc*) (*CCAddr_t* flashAddress, uint8_t *memDst, uint32_t sizeToRead, void *context)

  Typedef of the Flash read function pointer, to be implemented by the partner.
- typedef uint32_t(* *CCBsvFlashWriteFunc*) (*CCAddr_t* flashAddress, uint8_t *memSrc, uint32_t sizeToWrite, void *context)

Detailed description

Contains CryptoCell Secure Boot and Secure Debug definitions and structures.

Macro definition documentation

#define CC_SB_MAX_SIZE_ADDITIONAL_DATA_BYTES 128

The maximal size of the additional-data of the Secure Boot in Bytes.

Typedef documentation

typedef uint32_t(* CCBsvFlashWriteFunc) (*CCAddr_t* flashAddress, uint8_t *memSrc, uint32_t sizeToWrite, void *context)

Typedef of the Flash write function pointer, to be implemented by the partner.

Used for writing authenticated and decrypted SW modules to flash memory.

Parameters:

I/O	Parameter	Description
in	`flashAddress`	The address for writing to flash memory.
out	`memSrc`	A pointer to the RAM source to read the data from.
in	`sizeToWrite`	The size to write in Bytes.
in	`context`	For partner use.

typedef uint32_t CCSbCertPubKeyHash_t[HASH_RESULT_SIZE_IN_WORDS]

Definition of public key hash array.

typedef uint32_t CCSbCertSocId_t[HASH_RESULT_SIZE_IN_WORDS]

Definition of SOC ID array.

typedef uint32_t(* CCSbFlashReadFunc) (*CCAddr_t* flashAddress, uint8_t *memDst, uint32_t sizeToRead, void *context)

Typedef of the Flash read function pointer, to be implemented by the partner.

Used for reading the certificates and SW modules from flash memory.

Note

It is your responsibility to verify that this function does not copy data from restricted memory regions.

Parameters:

I/O	Parameter	Description
	flashAddress	The address for reading from flash memory.
	memDst	A pointer to the RAM destination address to write the data to.
	sizeToRead	The size to read in Bytes.
	context	For partner use.

1.5.26 CryptoCell SRAM mapping APIs

Contains internal SRAM mapping APIs.

Macros

- #define *CC_SRAM_PKA_BASE_ADDRESS* 0x0
- #define *CC_PKA_SRAM_SIZE_IN_KBYTES* 6
- #define *CC_SRAM_RND_HW_DMA_ADDRESS* 0x0
- #define *CC_SRAM_RND_MAX_SIZE* 0x800
- #define *CC_SRAM_MAX_SIZE* 0x1000

Detailed description

Contains internal SRAM mapping APIs.

Macro definition documentation

#define CC_PKA_SRAM_SIZE_IN_KBYTES 6

The size of the PKA SRAM in KB.

#define CC_SRAM_MAX_SIZE 0x1000

The maximal size of SRAM.

#define CC_SRAM_PKA_BASE_ADDRESS 0x0

The base address of the PKA in the PKA SRAM.

#define CC_SRAM_RND_HW_DMA_ADDRESS 0x0

The SRAM address of the RND.

#define CC_SRAM_RND_MAX_SIZE 0x800

Addresses 0K-2KB in SRAM. Reserved for RND operations.

1.5.27 CryptoCell SRP APIs

Contains CryptoCell SRP APIs.

Modules

- *Specific errors of the CryptoCell SRP APIs*

Contains the CryptoCell SRP-API error definitions. Data structures

- struct *mbedtls_srp_group_param*
- Group parameters for the SRP. struct *mbedtls_srp_context*

Macros

- #define *CC_SRP_MODULUS_SIZE_1024_BITS* 1024
- #define *CC_SRP_MODULUS_SIZE_1536_BITS* 1536
- #define *CC_SRP_MODULUS_SIZE_2048_BITS* 2048
- #define *CC_SRP_MODULUS_SIZE_3072_BITS* 3072
- #define *CC_SRP_MAX_MODULUS_IN_BITS* *CC_SRP_MODULUS_SIZE_3072_BITS*
- #define *CC_SRP_MAX_MODULUS* (*CC_SRP_MAX_MODULUS_IN_BITS*/*CC_BITS_IN_BYTE*)
- #define *CC_SRP_MAX_MODULUS_IN_WORDS* (*CC_SRP_MAX_MODULUS_IN_BITS*/*CC_BITS_IN_32BIT_WORD*)
- #define *CC_SRP_PRIV_NUM_MIN_SIZE_IN_BITS* (256)
- #define *CC_SRP_PRIV_NUM_MIN_SIZE* (*CC_SRP_PRIV_NUM_MIN_SIZE_IN_BITS*/*CC_BITS_IN_BYTE*)
- #define *CC_SRP_PRIV_NUM_MIN_SIZE_IN_WORDS* (*CC_SRP_PRIV_NUM_MIN_SIZE_IN_BITS*/*CC_BITS_IN_32BIT_WORD*)
- #define *CC_SRP_PRIV_NUM_MAX_SIZE_IN_BITS* (*CC_SRP_MAX_MODULUS_IN_BITS*)
- #define *CC_SRP_PRIV_NUM_MAX_SIZE* (*CC_SRP_PRIV_NUM_MAX_SIZE_IN_BITS*/*CC_BITS_IN_BYTE*)
- #define *CC_SRP_PRIV_NUM_MAX_SIZE_IN_WORDS* (*CC_SRP_PRIV_NUM_MAX_SIZE_IN_BITS*/*CC_BITS_IN_32BIT_WORD*)
- #define *CC_SRP_MAX_DIGEST_IN_WORDS* *CC_HASH_RESULT_SIZE_IN_WORDS*
- #define *CC_SRP_MAX_DIGEST* (*CC_SRP_MAX_DIGEST_IN_WORDS***CC_32BIT_WORD_SIZE*)
- #define *CC_SRP_MIN_SALT_SIZE* (8)
- #define *CC_SRP_MIN_SALT_SIZE_IN_WORDS* (*CC_SRP_MIN_SALT_SIZE*/*CC_32BIT_WORD_SIZE*)
- #define *CC_SRP_MAX_SALT_SIZE* (64)
- #define *CC_SRP_MAX_SALT_SIZE_IN_WORDS* (*CC_SRP_MAX_SALT_SIZE*/*CC_32BIT_WORD_SIZE*)

- #define *CC_SRP_HK_INIT*(srpType, srpModulus, srpGen, modSizeInBits, pUserName, userNameSize, pPwd, pwdSize, pRndCtx, pCtx) *mbedtls_srp_init*(srpType, *CC_SRP_VER_HK*, srpModulus, srpGen, modSizeInBits, *CC_HASH_SHA512_mode*, pUserName, userNameSize, pPwd, pwdSize, pRndCtx, pCtx)

Typedefs

- typedef uint8_t *mbedtls_srp_modulus*[*CC_SRP_MAX_MODULUS*]
- typedef uint8_t *mbedtls_srp_digest*[*CC_SRP_MAX_DIGEST*]
- typedef uint8_t *mbedtls_srp_secret*[2 **CC_SRP_MAX_DIGEST*]
- typedef struct *mbedtls_srp_group_param* *mbedtls_srp_group_param*

  Group parameters for the SRP.

- typedef struct *mbedtls_srp_context* *mbedtls_srp_context*

Enumerations

- enum *mbedtls_srp_version_t* { *CC_SRP_VER_3* = 0, *CC_SRP_VER_6* = 1, *CC_SRP_VER_6A* = 2, *CC_SRP_VER_HK* = 3, CC_SRP_NumOfVersions, *CC_SRP_VersionLast* = 0x7FFFFFFF }
- enum *mbedtls_srp_entity_t* { *CC_SRP_HOST* = 1, *CC_SRP_USER* = 2, CC_SRP_NumOfEntityType, *CC_SRP_EntityLast* = 0x7FFFFFFF }

Functions

- CIMPORT_C CCError_t *mbedtls_srp_init* (*mbedtls_srp_entity_t* srpType, *mbedtls_srp_version_t* srpVer, *mbedtls_srp_modulus* srpModulus, uint8_t srpGen, size_t modSizeInBits, *CCHashOperationMode_t* hashMode, uint8_t *pUserName, size_t userNameSize, uint8_t *pPwd, size_t pwdSize, *CCRndContext_t* *pRndCtx, *mbedtls_srp_context* *pCtx)

  This function initiates the SRP context.

- CIMPORT_C CCError_t *mbedtls_srp_pwd_ver_create* (size_t saltSize, uint8_t *pSalt, *mbedtls_srp_modulus* pwdVerifier, *mbedtls_srp_context* *pCtx)

  This function calculates pSalt and pwdVerifier.

- CIMPORT_C CCError_t *mbedtls_srp_clear* (*mbedtls_srp_context* *pCtx)

  This function clears the SRP context.

- CIMPORT_C CCError_t *mbedtls_srp_host_pub_key_create* (size_t ephemPrivSize, *mbedtls_srp_modulus* pwdVerifier, *mbedtls_srp_modulus* hostPubKeyB, *mbedtls_srp_context* *pCtx)

  This function generates the public and private host ephemeral keys, known as B and b in RFC 5054 Using the Secure Remote Password (SRP) Protocol for TLS Authentication.

- CIMPORT_C CCError_t *mbedtls_srp_host_proof_verify_and_calc* (size_t saltSize, uint8_t *pSalt, *mbedtls_srp_modulus* pwdVerifier, *mbedtls_srp_modulus* userPubKeyA, *mbedtls_srp_modulus* hostPubKeyB, *mbedtls_srp_digest* userProof, *mbedtls_srp_digest* hostProof, *mbedtls_srp_secret* sharedSecret, *mbedtls_srp_context* *pCtx)

  This function verifies the user proof, and calculates the host-message proof.

- CIMPORT_C CCError_t *mbedtls_srp_user_pub_key_create* (size_t ephemPrivSize, *mbedtls_srp_modulus* userPubKeyA, *mbedtls_srp_context* *pCtx)

  This function generates public and private user ephemeral keys, known as A and a in RFC 5054 Using the Secure Remote Password (SRP) Protocol for TLS Authentication.

- CIMPORT_C CCError_t *mbedtls_srp_user_proof_calc* (size_t saltSize, uint8_t *pSalt, *mbedtls_srp_modulus* userPubKeyA, *mbedtls_srp_modulus* hostPubKeyB, *mbedtls_srp_digest* userProof, *mbedtls_srp_secret* sharedSecret, *mbedtls_srp_context* *pCtx)

  This function calculates the user proof.

- CIMPORT_C CCError_t *mbedtls_srp_user_proof_verify* (*mbedtls_srp_secret* sharedSecret, *mbedtls_srp_modulus* userPubKeyA, *mbedtls_srp_digest* userProof, *mbedtls_srp_digest* hostProof, *mbedtls_srp_context* *pCtx)

  This function verifies the host proof.

Detailed description

Contains CryptoCell SRP APIs.

Macro definition documentation

#define CC_SRP_HK_INIT(srpType, srpModulus, srpGen, modSizeInBits, pUserName, userNameSize, pPwd, pwdSize, pRndCtx, pCtx) *mbedtls_srp_init*(srpType, *CC_SRP_VER_HK*, srpModulus, srpGen, modSizeInBits, *CC_HASH_SHA512_mode*, pUserName, userNameSize, pPwd, pwdSize, pRndCtx, pCtx)

Macro definition for a specific SRP-initialization function.

#define CC_SRP_MAX_DIGEST (*CC_SRP_MAX_DIGEST_IN_WORDS*CC_32BIT_WORD_SIZE*)**

The maximal size of the SRP hash digest in Bytes.

#define CC_SRP_MAX_DIGEST_IN_WORDS *CC_HASH_RESULT_SIZE_IN_WORDS*

The maximal size of the SRP hash digest in words.

#define CC_SRP_MAX_MODULUS (*CC_SRP_MAX_MODULUS_IN_BITS*/*CC_BITS_IN_BYTE*)

The maximal size of the SRP modulus in Bytes.

#define CC_SRP_MAX_MODULUS_IN_BITS *CC_SRP_MODULUS_SIZE_3072_BITS*

The maximal size of the SRP modulus in bits.

#define CC_SRP_MAX_MODULUS_IN_WORDS (*CC_SRP_MAX_MODULUS_IN_BITS*/*CC_BITS_IN_32BIT_WORD*)

The maximal size of the SRP modulus in words.

#define CC_SRP_MAX_SALT_SIZE (64)

The maximal size of the salt in Bytes.

#define CC_SRP_MAX_SALT_SIZE_IN_WORDS (*CC_SRP_MAX_SALT_SIZE*/*CC_32BIT_WORD_SIZE*)

The maximal size of the salt in words.

#define CC_SRP_MIN_SALT_SIZE (8)

The minimal size of the salt in Bytes.

#define CC_SRP_MIN_SALT_SIZE_IN_WORDS (CC_SRP_MIN_SALT_SIZE/CC_32BIT_WORD_SIZE)

The minimal size of the salt in words.

#define CC_SRP_MODULUS_SIZE_1024_BITS 1024

SRP modulus size of 1024 bits.

#define CC_SRP_MODULUS_SIZE_1536_BITS 1536

SRP modulus size of 1536 bits.

#define CC_SRP_MODULUS_SIZE_2048_BITS 2048

SRP modulus size of 2048 bits.

#define CC_SRP_MODULUS_SIZE_3072_BITS 3072

SRP modulus size of 3072 bits.

#define CC_SRP_PRIV_NUM_MAX_SIZE (CC_SRP_PRIV_NUM_MAX_SIZE_IN_BITS/CC_BITS_IN_BYTE)

The maximal size of the SRP private number in Bytes.

#define CC_SRP_PRIV_NUM_MAX_SIZE_IN_BITS (CC_SRP_MAX_MODULUS_IN_BITS)

The maximal size of the SRP private number in bits.

#define CC_SRP_PRIV_NUM_MAX_SIZE_IN_WORDS (CC_SRP_PRIV_NUM_MAX_SIZE_IN_BITS/CC_BITS_IN_32BIT_WORD)

The maximal size of the SRP private number in words.

#define CC_SRP_PRIV_NUM_MIN_SIZE (CC_SRP_PRIV_NUM_MIN_SIZE_IN_BITS/CC_BITS_IN_BYTE)

The minimal size of the SRP private number in Bytes.

#define CC_SRP_PRIV_NUM_MIN_SIZE_IN_BITS (256)

The minimal size of the SRP private number in bits.

#define CC_SRP_PRIV_NUM_MIN_SIZE_IN_WORDS (CC_SRP_PRIV_NUM_MIN_SIZE_IN_BITS/CC_BITS_IN_32BIT_WORD)

The minimal size of the SRP private number in words.

Typedef documentation

typedef struct mbedtls_srp_context mbedtls_srp_context

The SRP context prototype

typedef uint8_t mbedtls_srp_digest[CC_SRP_MAX_DIGEST]

The definition of the SRP digest buffer.

typedef struct mbedtls_srp_group_param mbedtls_srp_group_param

Group parameters for the SRP.

Defines the modulus and the generator used.

typedef uint8_t mbedtls_srp_modulus[*CC_SRP_MAX_MODULUS*]

The definition of the SRP modulus buffer.

typedef uint8_t mbedtls_srp_secret[2 **CC_SRP_MAX_DIGEST*]

The definition of the SRP secret buffer.

Enumeration type documentation

enum *mbedtls_srp_entity_t*

SRP entity types.

Enumerator:

Enum	Description
`CC_SRP_HOST`	The host entity, also known as server, verifier, or accessory.
`CC_SRP_USER`	The user entity, also known as client, or device.
`CC_SRP_EntityLast`	The maximal number of entities types.

enum *mbedtls_srp_version_t*

Supported SRP versions.

Enumerator:

Enum	Description
`CC_SRP_VER_3`	SRP version 3.
`CC_SRP_VER_6`	SRP version 6.
`CC_SRP_VER_6A`	SRP version 6A.
`CC_SRP_VER_HK`	SRP version HK.
`CC_SRP_VersionLast`	The maximal number of supported versions.

Function documentation

CIMPORT_C CCError_t mbedtls_srp_clear (*mbedtls_srp_context* * pCtx)

This function clears the SRP context.

Returns:

CC_OK on success.

A non-zero value on failure, as defined in *mbedtls_cc_srp_error.h*.

Parameters:

I/O	Parameter	Description
in,out	`pCtx`	A pointer to the SRP context.

CIMPORT_C CCError_t mbedtls_srp_host_proof_verify_and_calc (size_t saltSize, uint8_t * pSalt, *mbedtls_srp_modulus* pwdVerifier, *mbedtls_srp_modulus* userPubKeyA, *mbedtls_srp_modulus* hostPubKeyB, *mbedtls_srp_digest* userProof, *mbedtls_srp_digest* hostProof, *mbedtls_srp_secret* sharedSecret, *mbedtls_srp_context* * pCtx)

This function verifies the user proof, and calculates the host-message proof.

Returns:

CC_OK on success.

A non-zero value on failure, as defined in *mbedtls_cc_srp_error.h* or cc_hash_error.h.

Parameters:

I/O	Parameter	Description
in	`saltSize`	The size of the random salt. The range is between *CC_SRP_MIN_SALT_SIZE* and *CC_SRP_MAX_SALT_SIZE*.
in	`pSalt`	A pointer to the pSalt number.
in	`pwdVerifier`	A pointer to the password verifier (v).
in	`userPubKeyA`	A pointer to the ephemeral public key of the user (A).
in	`hostPubKeyB`	A pointer to the ephemeral public key of the host (B).
in	`userProof`	A pointer to the SRP user-proof buffer (M1).
out	`hostProof`	A pointer to the SRP host-proof buffer (M2).
out	`sharedSecret`	A pointer to the SRP shared secret (K).
in	`pCtx`	A pointer to the SRP context.

CIMPORT_C CCError_t mbedtls_srp_host_pub_key_create (size_t ephemPrivSize, *mbedtls_srp_modulus* pwdVerifier, *mbedtls_srp_modulus* hostPubKeyB, *mbedtls_srp_context* * pCtx)

This function generates the public and private host ephemeral keys, known as B and b in RFC 5054 Using the Secure Remote Password (SRP) Protocol for TLS Authentication.

Returns:

CC_OK on success.

A non-zero value on failure, as defined in *mbedtls_cc_srp_error.h* or cc_rnd_error.h.

Parameters:

I/O	Parameter	Description
in	`ephemPrivSize`	The size of the generated ephemeral private key (b). The range is between *CC_SRP_PRIV_NUM_MIN_SIZE* and *CC_SRP_PRIV_NUM_MAX_SIZE*
in	`pwdVerifier`	A pointer to the verifier (v).
out	`hostPubKeyB`	A pointer to the host ephemeral public key (B).
in,out	`pCtx`	A pointer to the SRP context.

CIMPORT_C CCError_t mbedtls_srp_init (*mbedtls_srp_entity_t* srpType, *mbedtls_srp_version_t* srpVer, *mbedtls_srp_modulus* srpModulus, uint8_t srpGen, size_t modSizeInBits, *CCHashOperationMode_t* hashMode, uint8_t * pUserName, size_t userNameSize, uint8_t * pPwd, size_t pwdSize, *CCRndContext_t* * pRndCtx, *mbedtls_srp_context* * pCtx)

This function initiates the SRP context.

Returns:

CC_OK on success.

A non-zero value on failure as defined in *mbedtls_cc_srp_error.h* or cc_hash_error.h.

Parameters:

I/O	Parameter	Description
in	`srpType`	The SRP entity type.
in	`srpVer`	The SRP version.
in	`srpModulus`	A pointer to the SRP modulus, BE Byte buffer.
in	`srpGen`	The SRP generator param.
in	`modSizeInBits`	The size of the SRP modulus in bits. Valid values are: ● 1024 ● 1536 ● 2048 ● 3072
in	`hashMode`	The hash mode.
in	`pUserName`	A pointer to the username.
in	`userNameSize`	The size of the username buffer. Must be larger than 0.
in	`pPwd`	A pointer to the user password.
in	`pwdSize`	The size of the user-password buffer. Must be larger than 0 if pPwd is valid.
in	`pRndCtx`	A pointer to the RND context.
out	`pCtx`	A pointer to the SRP host context.

CIMPORT_C CCError_t mbedtls_srp_pwd_ver_create (size_t saltSize, uint8_t * pSalt, *mbedtls_srp_modulus* pwdVerifier, *mbedtls_srp_context* * pCtx)

This function calculates pSalt and pwdVerifier.

Returns:

CC_OK on success.

A non-zero value on failure, as defined in *mbedtls_cc_srp_error.h*, cc_rnd_error.h or cc_hash_error.h.

Parameters:

I/O	Parameter	Description
in	`saltSize`	The size of the random salt to generate. The range is between *CC_SRP_MIN_SALT_SIZE* and *CC_SRP_MAX_SALT_SIZE*.

I/O	Parameter	Description
out	`pSalt`	A pointer to the pSalt number (s).
out	`pwdVerifier`	A pointer to the password verifier (v).
out	`pCtx`	A pointer to the SRP context.

CIMPORT_C CCError_t mbedtls_srp_user_proof_calc (size_t saltSize, uint8_t * pSalt, *mbedtls_srp_modulus* userPubKeyA, *mbedtls_srp_modulus* hostPubKeyB, *mbedtls_srp_digest* userProof, *mbedtls_srp_secret* sharedSecret, *mbedtls_srp_context* * pCtx)

This function calculates the user proof.

Returns:

CC_OK on success.

A non-zero value on failure, as defined in *mbedtls_cc_srp_error.h* or cc_hash_error.h.

Parameters:

I/O	Parameter	Description
in	`saltSize`	The size of the random salt. The range is between *CC_SRP_MIN_SALT_SIZE* and *CC_SRP_MAX_SALT_SIZE*.
in	`pSalt`	A pointer to the pSalt number.
in	`userPubKeyA`	A pointer to the public ephemeral key of the user (A).
in	`hostPubKeyB`	A pointer to the public ephemeral key of the host (B).
out	`userProof`	A pointer to the SRP user proof buffer (M1).
out	`sharedSecret`	A pointer to the SRP shared secret (K).
out	`pCtx`	A pointer to the SRP context.

CIMPORT_C CCError_t mbedtls_srp_user_proof_verify (*mbedtls_srp_secret* sharedSecret, *mbedtls_srp_modulus* userPubKeyA, *mbedtls_srp_digest* userProof, *mbedtls_srp_digest* hostProof, *mbedtls_srp_context* * pCtx)

This function verifies the host proof.

Returns:

CC_OK on success.

A non-zero value on failure, as defined in *mbedtls_cc_srp_error.h* or cc_hash_error.h.

Parameters:

I/O	Parameter	Description
in	`sharedSecret`	A pointer to the SRP shared secret (K).
in	`userPubKeyA`	A pointer to the public ephmeral key of the user (A).
in	`userProof`	A pointer to the SRP user proof buffer (M1).
in	`hostProof`	A pointer to the SRP host proof buffer (M2).
out	`pCtx`	A pointer to the SRP user context.

CIMPORT_C CCError_t mbedtls_srp_user_pub_key_create (size_t ephemPrivSize, *mbedtls_srp_modulus* userPubKeyA, *mbedtls_srp_context* * pCtx)

This function generates public and private user ephemeral keys, known as A and a in RFC 5054 Using the Secure Remote Password (SRP) Protocol for TLS Authentication.

Returns:

CC_OK on success.

A non-zero value on failure, as defined in *mbedtls_cc_srp_error.h* or cc_rnd_error.h.

Parameters:

I/O	Parameter	Description
in	`ephemPrivSize`	The size of the generated ephemeral private key (a). The range is between *CC_SRP_PRIV_NUM_MIN_SIZE* and *CC_SRP_PRIV_NUM_MAX_SIZE*.
out	`userPubKeyA`	A pointer to the user ephemeral public key (A).
in,out	`pCtx`	A pointer to the SRP context.

Specific errors of the CryptoCell SRP APIs

Contains the CryptoCell SRP-API error definitions.

Macros

- #define *CC_SRP_PARAM_INVALID_ERROR* (*CC_SRP_MODULE_ERROR_BASE* + 0x01UL)
- #define *CC_SRP_MOD_SIZE_INVALID_ERROR* (*CC_SRP_MODULE_ERROR_BASE* + 0x02UL)
- #define *CC_SRP_STATE_UNINITIALIZED_ERROR* (*CC_SRP_MODULE_ERROR_BASE* + 0x03UL)
- #define *CC_SRP_RESULT_ERROR* (*CC_SRP_MODULE_ERROR_BASE* + 0x04UL)
- #define *CC_SRP_PARAM_ERROR* (*CC_SRP_MODULE_ERROR_BASE* + 0x05UL)
- #define *CC_SRP_INTERNAL_ERROR* (*CC_SRP_MODULE_ERROR_BASE* + 0x06UL)

Detailed description

Contains the CryptoCell SRP-API error definitions.

Macro definition documentation

#define CC_SRP_INTERNAL_ERROR (*CC_SRP_MODULE_ERROR_BASE* + 0x06UL)

Internal PKI error.

#define CC_SRP_MOD_SIZE_INVALID_ERROR (*CC_SRP_MODULE_ERROR_BASE* + 0x02UL)

Illegal modulus size.

#define CC_SRP_PARAM_ERROR (*CC_SRP_MODULE_ERROR_BASE* + 0x05UL)

Invalid parameter.

#define CC_SRP_PARAM_INVALID_ERROR (*CC_SRP_MODULE_ERROR_BASE* + 0x01UL)

Illegal parameter.

#define CC_SRP_RESULT_ERROR (CC_SRP_MODULE_ERROR_BASE + 0x04UL)

Result validation error.

#define CC_SRP_STATE_UNINITIALIZED_ERROR (CC_SRP_MODULE_ERROR_BASE + 0x03UL)

Illegal state (uninitialized).

1.5.28 CryptoCell utility APIs

This contains all utility APIs.

Modules

- *CryptoCell utility APIs general definitions*
- Contains CryptoCell utility APIs general definitions. *CryptoCell utility key-derivation APIs*
- Contains the CryptoCell utility key-derivation function API. *CryptoCell utils general key definitions*
- Contains KDF API definitions. *Specific errors of the CryptoCell utility module APIs*

Contains utility API error definitions.

Detailed description

This contains all utility APIs.

CryptoCell utility APIs general definitions

Contains CryptoCell utility APIs general definitions.

Data structures

- struct *mbedtls_util_keydata*

Macros

- #define *CC_UTIL_AES_128BIT_SIZE* 16
- #define *CC_UTIL_AES_192BIT_SIZE* 24
- #define *CC_UTIL_AES_256BIT_SIZE* 32
- #define *CC_UTIL_CMAC_DERV_MIN_DATA_IN_SIZE* *CC_UTIL_FIX_DATA_MIN_SIZE_IN_BYTES*+2
- #define *CC_UTIL_CMAC_DERV_MAX_DATA_IN_SIZE* *CC_UTIL_MAX_KDF_SIZE_IN_BYTES*
- #define *CC_UTIL_AES_CMAC_RESULT_SIZE_IN_BYTES* 0x10UL
- #define *CC_UTIL_AES_CMAC_RESULT_SIZE_IN_WORDS* (*CC_UTIL_AES_CMAC_RESULT_SIZE_IN_BYTES*/sizeof(uint32_t))

Typedefs

- typedef uint32_t *CCUtilError_t*
- typedef struct *mbedtls_util_keydata* *mbedtls_util_keydata*

Detailed description

Contains CryptoCell utility APIs general definitions.

Macro definition documentation

#define CC_UTIL_AES_128BIT_SIZE 16

The size of the AES 128-bit key in Bytes.

#define CC_UTIL_AES_192BIT_SIZE 24

The size of the AES 192-bit key in Bytes.

#define CC_UTIL_AES_256BIT_SIZE 32

The size of the AES 256-bit key in Bytes.

#define CC_UTIL_AES_CMAC_RESULT_SIZE_IN_BYTES 0x10UL

The size of the AES CMAC result in Bytes.

#define CC_UTIL_AES_CMAC_RESULT_SIZE_IN_WORDS (CC_UTIL_AES_CMAC_RESULT_SIZE_IN_BYTES/sizeof(uint32_t))

The size of the AES CMAC result in words.

#define CC_UTIL_CMAC_DERV_MAX_DATA_IN_SIZE CC_UTIL_MAX_KDF_SIZE_IN_BYTES

The maximal size of the data for CMAC derivation operation.

#define CC_UTIL_CMAC_DERV_MIN_DATA_IN_SIZE CC_UTIL_FIX_DATA_MIN_SIZE_IN_BYTES+2

The minimal size of the data for the CMAC derivation operation.

Typedef documentation

typedef uint32_t CCUtilError_t

Util error type.

typedef struct mbedtls_util_keydata mbedtls_util_keydata

Key data.

CryptoCell utility key-derivation APIs

Contains the CryptoCell utility key-derivation function API.

Macros

- #define mbedtls_util_key_derivation_cmac(keyType, pUserKey, pLabel, labelSize, pContextData, contextSize, pDerivedKey, derivedKeySize) mbedtls_util_key_derivation(keyType, pUserKey, CC_UTIL_PRF_CMAC, CC_HASH_OperationModeLast, pLabel, labelSize, pContextData, contextSize, pDerivedKey, derivedKeySize)

  This function performs key derivation using CMAC.

- #define mbedtls_util_key_derivation_hmac(keyType, pUserKey, hashMode, pLabel, labelSize, pContextData, contextSize, pDerivedKey, derivedKeySize) mbedtls_util_key_derivation(keyType, pUserKey, CC_UTIL_PRF_HMAC, hashMode, pLabel, labelSize, pContextData, contextSize, pDerivedKey, derivedKeySize)

This function performs key derivation using HMAC.

Enumerations

- enum *mbedtls_util_keytype_t* { *CC_UTIL_USER_KEY* = 0, *CC_UTIL_ROOT_KEY* = 1, *CC_UTIL_TOTAL_KEYS* = 2, *CC_UTIL_END_OF_KEY_TYPE* = 0x7FFFFFFF }
- enum *mbedtls_util_prftype_t* { *CC_UTIL_PRF_CMAC* = 0, *CC_UTIL_PRF_HMAC* = 1, *CC_UTIL_TOTAL_PRFS* = 2, *CC_UTIL_END_OF_PRF_TYPE* = 0x7FFFFFFF }

Functions

- *CCUtilError_t* *mbedtls_util_key_derivation* (*mbedtls_util_keytype_t* keyType, *mbedtls_util_keydata* *pUserKey, *mbedtls_util_prftype_t* prfType, *CCHashOperationMode_t* hashMode, const uint8_t *pLabel, size_t labelSize, const uint8_t *pContextData, size_t contextSize, uint8_t *pDerivedKey, size_t derivedKeySize)

  This function performs key derivation using AES-CMAC.

Detailed description

Contains the CryptoCell utility key-derivation function API.

Macro definition documentation

#define mbedtls_util_key_derivation_cmac(keyType, pUserKey, pLabel, labelSize, pContextData, contextSize, pDerivedKey, derivedKeySize) *mbedtls_util_key_derivation*(keyType, pUserKey, *CC_UTIL_PRF_CMAC*, *CC_HASH_OperationModeLast*, pLabel, labelSize, pContextData, contextSize, pDerivedKey, derivedKeySize)

This function performs key derivation using CMAC.

It is defined as specified in the KDF in Counter Mode section in NIST Special Publication 800-108: Recommendation for Key Derivation Using Pseudorandom Functions.

The derivation is based on length l, label L, context C, and derivation key Ki. In this macro, AES-CMAC is used as the pseudo-random function (PRF).

Returns:

CC_UTIL_OK on success.

A non-zero value from *cc_util_error.h* on failure.

#define mbedtls_util_key_derivation_hmac(keyType, pUserKey, hashMode, pLabel, labelSize, pContextData, contextSize, pDerivedKey, derivedKeySize) *mbedtls_util_key_derivation*(keyType, pUserKey, *CC_UTIL_PRF_HMAC*, hashMode, pLabel, labelSize, pContextData, contextSize, pDerivedKey, derivedKeySize)

This function performs key derivation using HMAC.

It is defined as specified in the KDF in Counter Mode section in NIST Special Publication 800-108: Recommendation for Key Derivation Using Pseudorandom Functions.

The derivation is based on length l, label L, context C, and derivation key Ki. In this macro, HMAC is used as the pseudo-random function (PRF).

Returns:

CC_UTIL_OK on success.

A non-zero value from *cc_util_error.h* on failure.

Enumeration type documentation

enum mbedtls_util_keytype_t

Input key derivation type.

Enumerator:

Enum	Description
`CC_UTIL_USER_KEY`	The user key.
`CC_UTIL_ROOT_KEY`	The device root key (HUK).
`CC_UTIL_TOTAL_KEYS`	Total number of keys.
`CC_UTIL_END_OF_KEY_TYPE`	Reserved.

enum mbedtls_util_prftype_t

Pseudo-random function type for key derivation.

Enumerator:

Enum	Description
`CC_UTIL_PRF_CMAC`	CMAC function.
`CC_UTIL_PRF_HMAC`	HMAC function.
`CC_UTIL_TOTAL_PRFS`	Total number of pseudo-random functions.
`CC_UTIL_END_OF_PRF_TYPE`	Reserved.

Function documentation

CCUtilError_t mbedtls_util_key_derivation (mbedtls_util_keytype_t keyType, mbedtls_util_keydata * pUserKey, mbedtls_util_prftype_t prfType, CCHashOperationMode_t hashMode, const uint8_t * pLabel, size_t labelSize, const uint8_t * pContextData, size_t contextSize, uint8_t * pDerivedKey, size_t derivedKeySize)

This function performs key derivation using AES-CMAC.

It is defined as specified in the KDF in Counter Mode section in NIST Special Publication 800-108: Recommendation for Key Derivation Using Pseudorandom Functions.

The derivation is based on length l, label L, context C, and derivation key Ki. AES-CMAC is used as the pseudo-random function (PRF).

Note

The user must define the label and context for each use-case well, when using this API.

Returns:

CC_UTIL_OK on success.

A non-zero value from cc_util_error.h on failure.

Parameters:

I/O	Parameter	Description
in	`keyType`	The key type that is used as an input to a key-derivation function. Can be one of the following: • CC_UTIL_USER_KEY • CC_UTIL_ROOT_KEY
in	`pUserKey`	A pointer to the key buffer of the user, in case of CC_UTIL_USER_KEY.
in	`prfType`	The PRF type that is used as an input to a key-derivation function. Can be one of the following: • CC_UTIL_PRF_CMAC • CC_UTIL_PRF_HMAC
in	`hashMode`	One of the supported hash modes, as defined in CCHashOperationMode_t.
in	`pLabel`	A string that identifies the purpose for the derived keying material.
in	`labelSize`	The label size should be in range of 1 to 64 Bytes in length.
in	`pContextData`	A binary string containing the information related to the derived keying material.
in	`contextSize`	The context size should be in range of 1 to 64 Bytes in length.
out	`pDerivedKey`	Keying material output. Must be at least the size of derivedKeySize.
in	`derivedKeySize`	The size of the derived keying material in Bytes, up to 4080 Bytes.

CryptoCell utils general key definitions

Contains KDF API definitions.

Macros

- #define CC_UTIL_MAX_LABEL_LENGTH_IN_BYTES 64
- #define CC_UTIL_MAX_CONTEXT_LENGTH_IN_BYTES 64
- #define CC_UTIL_FIX_DATA_MIN_SIZE_IN_BYTES 3
- #define CC_UTIL_FIX_DATA_MAX_SIZE_IN_BYTES 4
- #define CC_UTIL_MAX_KDF_SIZE_IN_BYTES (CC_UTIL_MAX_LABEL_LENGTH_IN_BYTES+CC_UTIL_MAX_CONTEXT_LENGTH_IN_BYTES+CC_UTIL_FIX_DATA_MAX_SIZE_IN_BYTES)
- #define CC_UTIL_MAX_DERIVED_KEY_SIZE_IN_BYTES 4080

Detailed description

Contains KDF API definitions.

Macro definition documentation

#define CC_UTIL_FIX_DATA_MAX_SIZE_IN_BYTES 4

The maximal size of the fixed data in Bytes.

#define CC_UTIL_FIX_DATA_MIN_SIZE_IN_BYTES 3

The minimal size of the fixed data in Bytes.

#define CC_UTIL_MAX_CONTEXT_LENGTH_IN_BYTES 64

The maximal length of the context in Bytes.

#define CC_UTIL_MAX_DERIVED_KEY_SIZE_IN_BYTES 4080

The maximal size of the derived-key in Bytes.

#define CC_UTIL_MAX_KDF_SIZE_IN_BYTES (CC_UTIL_MAX_LABEL_LENGTH_IN_BYTES +CC_UTIL_MAX_CONTEXT_LENGTH_IN_BYTES+CC_UTIL_FIX_DATA_MAX_SIZE_IN_BYTES)

The maximal size of the derived-key material in Bytes.

#define CC_UTIL_MAX_LABEL_LENGTH_IN_BYTES 64

The maximal length of the label in Bytes.

Specific errors of the CryptoCell utility module APIs

Contains utility API error definitions.

Macros

- #define CC_UTIL_OK 0x00UL
- #define CC_UTIL_MODULE_ERROR_BASE 0x80000000
- #define CC_UTIL_INVALID_KEY_TYPE (CC_UTIL_MODULE_ERROR_BASE + 0x00UL)
- #define CC_UTIL_DATA_IN_POINTER_INVALID_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x01UL)
- #define CC_UTIL_DATA_IN_SIZE_INVALID_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x02UL)
- #define CC_UTIL_DATA_OUT_POINTER_INVALID_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x03UL)
- #define CC_UTIL_DATA_OUT_SIZE_INVALID_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x04UL)
- #define CC_UTIL_FATAL_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x05UL)
- #define CC_UTIL_ILLEGAL_PARAMS_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x06UL)
- #define CC_UTIL_BAD_ADDR_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x07UL)
- #define CC_UTIL_EK_DOMAIN_INVALID_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x08UL)
- #define CC_UTIL_KDR_INVALID_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x09UL)
- #define CC_UTIL_LCS_INVALID_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x0AUL)
- #define CC_UTIL_SESSION_KEY_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x0BUL)
- #define CC_UTIL_INVALID_USER_KEY_SIZE (CC_UTIL_MODULE_ERROR_BASE + 0x0DUL)

- #define CC_UTIL_ILLEGAL_LCS_FOR_OPERATION_ERR (CC_UTIL_MODULE_ERROR_BASE + 0x0EUL)
- #define CC_UTIL_INVALID_PRF_TYPE (CC_UTIL_MODULE_ERROR_BASE + 0x0FUL)
- #define CC_UTIL_INVALID_HASH_MODE (CC_UTIL_MODULE_ERROR_BASE + 0x10UL)
- #define CC_UTIL_UNSUPPORTED_HASH_MODE (CC_UTIL_MODULE_ERROR_BASE + 0x11UL)
- #define CC_UTIL_KEY_UNUSABLE_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x12UL)

Detailed description

Contains utility API error definitions.

Macro definition documentation

#define CC_UTIL_BAD_ADDR_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x07UL)

Invalid address given.

#define CC_UTIL_DATA_IN_POINTER_INVALID_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x01UL)

Illegal data-in pointer.

#define CC_UTIL_DATA_IN_SIZE_INVALID_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x02UL)

Illegal data-in size.

#define CC_UTIL_DATA_OUT_POINTER_INVALID_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x03UL)

Illegal data-out pointer.

#define CC_UTIL_DATA_OUT_SIZE_INVALID_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x04UL)

Illegal data-out size.

#define CC_UTIL_EK_DOMAIN_INVALID_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x08UL)

Illegal domain for endorsement key.

#define CC_UTIL_FATAL_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x05UL)

Fatal error.

#define CC_UTIL_ILLEGAL_LCS_FOR_OPERATION_ERR (CC_UTIL_MODULE_ERROR_BASE + 0x0EUL)

Illegal LCS for the required operation.

#define CC_UTIL_ILLEGAL_PARAMS_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x06UL)

Illegal parameters.

#define CC_UTIL_INVALID_HASH_MODE (CC_UTIL_MODULE_ERROR_BASE + 0x10UL)

Invalid hash mode.

#define CC_UTIL_INVALID_KEY_TYPE (CC_UTIL_MODULE_ERROR_BASE + 0x00UL)

Illegal key type.

#define CC_UTIL_INVALID_PRF_TYPE (CC_UTIL_MODULE_ERROR_BASE + 0x0FUL)

Invalid PRF type.

#define CC_UTIL_INVALID_USER_KEY_SIZE (CC_UTIL_MODULE_ERROR_BASE + 0x0DUL)

Illegal user key size.

#define CC_UTIL_KDR_INVALID_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x09UL)

HUK is not valid.

#define CC_UTIL_KEY_UNUSABLE_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x12UL)

Key is unusable

#define CC_UTIL_LCS_INVALID_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x0AUL)

LCS is not valid.

#define CC_UTIL_MODULE_ERROR_BASE 0x80000000

The error base address definition.

#define CC_UTIL_OK 0x00UL

Success definition.

#define CC_UTIL_SESSION_KEY_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x0BUL)

Session key is not valid.

#define CC_UTIL_UNSUPPORTED_HASH_MODE (CC_UTIL_MODULE_ERROR_BASE + 0x11UL)

Unsupported hash mode.

1.6 Data Structure Documentation

1.6.1 CC_PalTrngParams_t Struct Reference

#include <cc_pal_trng.h>

Data Fields

- uint32_t *SubSamplingRatio1*
- uint32_t *SubSamplingRatio2*
- uint32_t *SubSamplingRatio3*
- uint32_t *SubSamplingRatio4*

Detailed description

Definition for the structure of the random-generator parameters of CryptoCell, containing the user-given parameters.

Field Documentation

uint32_t CC_PalTrngParams_t::SubSamplingRatio1

The sampling ratio of ROSC #1.

uint32_t CC_PalTrngParams_t::SubSamplingRatio2

The sampling ratio of ROSC #2.

uint32_t CC_PalTrngParams_t::SubSamplingRatio3

The sampling ratio of ROSC #3.

uint32_t CC_PalTrngParams_t::SubSamplingRatio4

The sampling ratio of ROSC #4.

The documentation for this struct was generated from the following file:

- *cc_pal_trng.h*

1.6.2 CCAesHwKeyData_t Struct Reference

#include <cc_aes_defs.h>

Data Fields

- size_t *slotNumber*

Detailed description

The AES HW key Data.

Field Documentation

size_t CCAesHwKeyData_t::slotNumber

Slot number.

The documentation for this struct was generated from the following file:

- *cc_aes_defs.h*

1.6.3 CCAesUserContext_t Struct Reference

The context prototype of the user.

#include <cc_aes_defs.h>

Data Fields

- uint32_t *buff* [*CC_AES_USER_CTX_SIZE_IN_WORDS*]

Detailed description

The context prototype of the user.

The argument type that is passed by the user to the AES APIs. The context saves the state of the operation, and must be saved by the user till the end of the API flow.

Field Documentation

uint32_t CCAesUserContext_t::buff[*CC_AES_USER_CTX_SIZE_IN_WORDS*]

The context buffer for internal usage.

The documentation for this struct was generated from the following file:

- *cc_aes_defs.h*

1.6.4 CCAesUserKeyData_t Struct Reference

#include <cc_aes_defs.h>

Data Fields

- uint8_t * *pKey*
- size_t *keySize*

Detailed description

The AES key data of the user.

Field Documentation

size_t CCAesUserKeyData_t::keySize

The size of the key in Bytes. Valid values:

- For XTS mode (if supportes): 32 Bytes or 64 Bytes, indicating the full size of the double key (2x128 or 2x256 bit).
- For XCBC-MAC mode: 16 Bytes, as limited by the standard.
- For all other modes: 16 Bytes, 24 Bytes or 32 Bytes.

uint8_t* CCAesUserKeyData_t::pKey

A pointer to the key.

The documentation for this struct was generated from the following file:

- *cc_aes_defs.h*

1.6.5 CCAssetBuff_t Union Reference

The asset buffer.

#include <cc_prod.h>

Data Fields

- *CCPlainAsset_t plainAsset*
- *CCAssetPkg_t pkgAsset*

Detailed description

The asset buffer.

If the asset is provided as plain asset, the plainAsset field is used. Otherwise, the pkgAsset field is used.

Field Documentation

CCAssetPkg_t CCAssetBuff_t::pkgAsset

Asset-package buffer.

CCPlainAsset_t CCAssetBuff_t::plainAsset

Plain asset buffer.

The documentation for this union was generated from the following file:

- *cc_prod.h*

1.6.6 CCCmpuData_t Struct Reference

#include <cc_cmpu.h>

Data Fields

- *CCCmpuUniqueDataType_t uniqueDataType*
- *CCCmpuUniqueBuff_t uniqueBuff*
- *CCAssetType_t kpicvDataType*
- *CCAssetBuff_t kpicv*
- *CCAssetType_t kceicvDataType*
- *CCAssetBuff_t kceicv*
- uint32_t *icvMinVersion*
- uint32_t *icvConfigWord*
- uint32_t *icvDcuDefaultLock* [*PROD_DCU_LOCK_WORD_SIZE*]

Detailed description

The ICV production library input options.

Field Documentation

uint32_t CCCmpuData_t::icvConfigWord

The ICV configuration word.

uint32_t CCCmpuData_t::icvDcuDefaultLock[*PROD_DCU_LOCK_WORD_SIZE*]

The default DCU lock bits of the ICV. Valid only if Hbk0 is used.

uint32_t CCCmpuData_t::icvMinVersion

The minimal SW version of the ICV. Valid only if Hbk0 is used.

CCAssetBuff_t **CCCmpuData_t::kceicv**

The buffer of the Kceicv, if its type is plain asset or package.

CCAssetType_t **CCCmpuData_t::kceicvDataType**

The asset type of the Kceicv. Allowed values are:

- Not used.
- Plain-asset.
- Package.

CCAssetBuff_t **CCCmpuData_t::kpicv**

The buffer of the Kpicv, if its type is plain-asset or package.

CCAssetType_t **CCCmpuData_t::kpicvDataType**

The asset type of the Kpicv. Allowed values are:

- Not used.
- Plain-asset.
- Package.

CCCmpuUniqueBuff_t **CCCmpuData_t::uniqueBuff**

The unique data buffer.

CCCmpuUniqueDataType_t **CCCmpuData_t::uniqueDataType**

The unique data type: Hbk0 or a random user-defined data.

The documentation for this struct was generated from the following file:

- *cc_cmpu.h*

1.6.7 CCCmpuUniqueBuff_t Union Reference

The device use of the unique buffer.

#include <cc_cmpu.h>

Data Fields

- uint8_t *hbk0* [*PROD_UNIQUE_BUFF_SIZE*]
- uint8_t *userData* [*PROD_UNIQUE_BUFF_SIZE*]

Detailed description

The device use of the unique buffer.

If the device uses Hbk0, then the hbk0 field is used. Otherwise, a random buffer for the userData field is used.

Field Documentation

uint8_t CCCmpuUniqueBuff_t::hbk0[*PROD_UNIQUE_BUFF_SIZE*]

The Hbk0 buffer, if used by the device.

uint8_t CCCmpuUniqueBuff_t::userData[*PROD_UNIQUE_BUFF_SIZE*]

Any random value, if Hbk0 is not used by the device.

The documentation for this union was generated from the following file:

- *cc_cmpu.h*

1.6.8 CCDmpuData_t Struct Reference

#include <cc_dmpu.h>

Data Fields

- *CCDmpuHBKType_t hbkType*
- *CCDmpuHbkBuff_t hbkBuff*
- *CCAssetType_t kcpDataType*
- *CCAssetBuff_t kcp*
- *CCAssetType_t kceDataType*
- *CCAssetBuff_t kce*
- uint32_t *oemMinVersion*
- uint32_t *oemDcuDefaultLock* [*PROD_DCU_LOCK_WORD_SIZE*]

Detailed description

The OEM production library input defines.

Field Documentation

CCDmpuHbkBuff_t **CCDmpuData_t::hbkBuff**

The Hbk buffer.

CCDmpuHBKType_t **CCDmpuData_t::hbkType**

The type of Hbk:

- Hbk1 with 128 bits.
- Hbk with 256 bits.

CCAssetBuff_t **CCDmpuData_t::kce**

The Kce buffer, if kceDataType is plain asset or package.

CCAssetType_t **CCDmpuData_t::kceDataType**

The Kce asset type. Allowed values are:

- Not used.
- Plain-asset.
- Package.

CCAssetBuff_t **CCDmpuData_t::kcp**

The Kcp buffer, if kcpDataType is plain asset or package.

CCAssetType_t **CCDmpuData_t::kcpDataType**

The Kcp asset type. Allowed values are:

- Not used.

- Plain-asset.
- Package.

uint32_t CCDmpuData_t::oemDcuDefaultLock[PROD_DCU_LOCK_WORD_SIZE]

The default DCU lock bits of the OEM.

uint32_t CCDmpuData_t::oemMinVersion

The minimal SW version of the OEM.

The documentation for this struct was generated from the following file:

- *cc_dmpu.h*

1.6.9 CCDmpuHbkBuff_t Union Reference

The device use of the Hbk buffer.

#include <cc_dmpu.h>

Data Fields

- uint8_t *hbk1* [*DMPU_HBK1_SIZE_IN_WORDS* **CC_PROD_32BIT_WORD_SIZE*]
- uint8_t *hbk* [*DMPU_HBK_SIZE_IN_WORDS* **CC_PROD_32BIT_WORD_SIZE*]

Detailed description

The device use of the Hbk buffer.

If the device uses Hbk0 and Hbk1, then the Hbk1 field is used. Otherwise, the Hbk field is used.

Field Documentation

uint8_t CCDmpuHbkBuff_t::hbk[DMPU_HBK_SIZE_IN_WORDS *CC_PROD_32BIT_WORD_SIZE]

Hbk buffer, that is, the full 256 bits.

uint8_t CCDmpuHbkBuff_t::hbk1[DMPU_HBK1_SIZE_IN_WORDS *CC_PROD_32BIT_WORD_SIZE]

Hbk1 buffer if used by the device.

The documentation for this union was generated from the following file:

- *cc_dmpu.h*

1.6.10 CCEcdhFipsKatContext_t Struct Reference

#include <cc_ecpki_types.h>

Data Fields

- *CCEcpkiUserPublKey_t pubKey*
- *CCEcpkiUserPrivKey_t privKey*
- union {

  CCEcpkiBuildTempData_t ecpkiTempData

    CCEcdhTempData_t ecdhTempBuff

  } tmpData

- uint8_t secretBuff [CC_ECPKI_FIPS_ORDER_LENGTH]

Detailed description

ECDH KAT data structures for FIPS certification.

Field Documentation

CCEcpkiUserPrivKey_t CCEcdhFipsKatContext_t::privKey

The data of the private key.

CCEcpkiUserPublKey_t CCEcdhFipsKatContext_t::pubKey

The data of the public key.

uint8_t CCEcdhFipsKatContext_t::secretBuff[CC_ECPKI_FIPS_ORDER_LENGTH]

The buffer for the secret key.

union { ... }   CCEcdhFipsKatContext_t::tmpData

Internal buffers.

The documentation for this struct was generated from the following file:

- *cc_ecpki_types.h*

1.6.11 CCEcdhTempData_t Struct Reference

#include <cc_ecpki_types.h>

Data Fields

- uint32_t ccEcdhIntBuff [CC_PKA_ECDH_BUFF_MAX_LENGTH_IN_WORDS]

Detailed description

The type of the ECDH temporary data.

Field Documentation

uint32_t CCEcdhTempData_t::ccEcdhIntBuff[CC_PKA_ECDH_BUFF_MAX_LENGTH_IN_WORDS]

Temporary buffers.

The documentation for this struct was generated from the following file:

- *cc_ecpki_types.h*

1.6.12 CCEcdsaFipsKatContext_t Struct Reference

#include <cc_ecpki_types.h>

Data Fields

- union {

  struct {

  CCEcpkiUserPrivKey_t PrivKey

  CCEcdsaSignUserContext_t signCtx

  } *userSignData*

  struct {

  CCEcpkiUserPublKey_t PublKey

  union {

  CCEcdsaVerifyUserContext_t verifyCtx

  CCEcpkiBuildTempData_t tempData

  } buildOrVerify

  } *userVerifyData*

  } *keyContextData*
- uint8_t *signBuff* [2 **CC_ECPKI_FIPS_ORDER_LENGTH*]

Detailed description

ECDSA KAT data structures for FIPS certification. The ECDSA KAT tests are defined for domain 256r1.

Field Documentation

union { ... } CCEcdsaFipsKatContext_t::keyContextData

The data of the key.

uint8_t CCEcdsaFipsKatContext_t::signBuff[2 *CC_ECPKI_FIPS_ORDER_LENGTH]

Internal buffer.

struct { ... } CCEcdsaFipsKatContext_t::userSignData

The data of the private key.

struct { ... } CCEcdsaFipsKatContext_t::userVerifyData

The data of the public key.

The documentation for this struct was generated from the following file:

- *cc_ecpki_types.h*

1.6.13 CCEcdsaSignUserContext_t Struct Reference

The context definition of the user for the signing operation.

#include <cc_ecpki_types.h>

Data Fields

- uint32_t *context_buff* [(sizeof(*EcdsaSignContext_t*)+3)/4]
- uint32_t *valid_tag*

Detailed description

The context definition of the user for the signing operation.

This context saves the state of the operation, and must be saved by the user until the end of the API flow.

Field Documentation

uint32_t CCEcdsaSignUserContext_t::context_buff[(sizeof(*EcdsaSignContext_t*)+3)/4]

The data of the signing process.

uint32_t CCEcdsaSignUserContext_t::valid_tag

The validation tag.

The documentation for this struct was generated from the following file:

- *cc_ecpki_types.h*

1.6.14 CCEcdsaVerifyUserContext_t Struct Reference

The context definition of the user for the verification operation.

#include <cc_ecpki_types.h>

Data Fields

- uint32_t *context_buff* [(sizeof(*EcdsaVerifyContext_t*)+3)/4]
- uint32_t *valid_tag*

Detailed description

The context definition of the user for the verification operation.

The context saves the state of the operation, and must be saved by the user until the end of the API flow.

Field Documentation

uint32_t CCEcdsaVerifyUserContext_t::context_buff[(sizeof(*EcdsaVerifyContext_t*)+3)/4]

The data of the verification process.

uint32_t CCEcdsaVerifyUserContext_t::valid_tag

The validation tag.

The documentation for this struct was generated from the following file:

- *cc_ecpki_types.h*

1.6.15 CCEciesTempData_t Struct Reference

#include <cc_ecpki_types.h>

Data Fields

- *CCEcpkiUserPrivKey_t PrivKey*
- *CCEcpkiUserPublKey_t PublKey*
- *CCEcpkiUserPublKey_t ConvPublKey*
- uint32_t *zz* [3 **CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS*+1]
- union {

  CCEcpkiBuildTempData_t buildTempbuff

  CCEcpkiKgTempData_t KgTempBuff

  CCEcdhTempData_t DhTempBuff

  } *tmp*

Detailed description

The temporary data definition of the ECIES.

Field Documentation

CCEcpkiUserPublKey_t CCEciesTempData_t::ConvPublKey

The public-key data used by convertion from Mbed TLS to CryptoCell.

CCEcpkiUserPrivKey_t CCEciesTempData_t::PrivKey

The data of the private key.

CCEcpkiUserPublKey_t CCEciesTempData_t::PublKey

The data of the public key.

union { ... } CCEciesTempData_t::tmp

Internal buffers.

uint32_t CCEciesTempData_t::zz[3 *CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS+1]

Internal buffer.

The documentation for this struct was generated from the following file:

- *cc_ecpki_types.h*

1.6.16 CCEcpkiBuildTempData_t Struct Reference

#include <cc_ecpki_types.h>

Data Fields

- uint32_t *ccBuildTmpIntBuff* [*CC_PKA_ECPKI_BUILD_TMP_BUFF_MAX_LENGTH_IN_WORDS*]

Detailed description

EC build temporary data.

Field Documentation

uint32_t CCEcpkiBuildTempData_t::ccBuildTmpIntBuff[*CC_PKA_ECPKI_BUILD_TMP_BUFF_MAX_LENGTH_IN_WORDS*]

Temporary buffers.

The documentation for this struct was generated from the following file:

- *cc_ecpki_types.h*

1.6.17 CCEcpkiDomain_t Struct Reference

The structure containing the EC domain parameters in little-endian form.

#include <cc_ecpki_types.h>

Data Fields

- uint32_t *ecP* [*CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS*]
- uint32_t *ecA* [*CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS*]
- uint32_t *ecB* [*CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS*]
- uint32_t *ecR* [*CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS*+1]
- uint32_t *ecGx* [*CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS*]
- uint32_t *ecGy* [*CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS*]
- uint32_t *ecH*
- uint32_t *llfBuff* [*CC_PKA_DOMAIN_LLF_BUFF_SIZE_IN_WORDS*]
- uint32_t *modSizeInBits*
- uint32_t *ordSizeInBits*
- uint32_t *barrTagSizeInWords*
- *CCEcpkiDomainID_t* *DomainID*
- int8_t *name* [20]

Detailed description

The structure containing the EC domain parameters in little-endian form.

EC equation: Y^2 = X^3 + A*X + B over prime field GFp.

Field Documentation

uint32_t CCEcpkiDomain_t::barrTagSizeInWords

The size of each inserted Barret tag in words. 0 if not inserted.

CCEcpkiDomainID_t **CCEcpkiDomain_t::DomainID**

The EC Domain identifier.

uint32_t CCEcpkiDomain_t::ecA[*CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS*]

EC equation parameter A.

uint32_t CCEcpkiDomain_t::ecB[CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS]

EC equation parameter B.

uint32_t CCEcpkiDomain_t::ecGx[CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS]

EC cofactor EC_Cofactor_K. The coordinates of the EC base point generator in projective form.

uint32_t CCEcpkiDomain_t::ecGy[CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS]

EC cofactor EC_Cofactor_K. The coordinates of the EC base point generator in projective form.

uint32_t CCEcpkiDomain_t::ecH

EC cofactor EC_Cofactor_K. The coordinates of the EC base point generator in projective form.

uint32_t CCEcpkiDomain_t::ecP[CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS]

EC modulus: P.

uint32_t CCEcpkiDomain_t::ecR[CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS+1]

Order of generator.

uint32_t CCEcpkiDomain_t::llfBuff[CC_PKA_DOMAIN_LLF_BUFF_SIZE_IN_WORDS]

Specific fields that are used by the low-level functions.

uint32_t CCEcpkiDomain_t::modSizeInBits

The size of fields in bits.

int8_t CCEcpkiDomain_t::name[20]

Internal buffer.

uint32_t CCEcpkiDomain_t::ordSizeInBits

The size of the order in bits.

The documentation for this struct was generated from the following file:

- *cc_ecpki_types.h*

1.6.18 CCEcpkiKgFipsContext_t Struct Reference

#include <cc_ecpki_types.h>

Data Fields

- union {

  CCEcdsaSignUserContext_t signCtx

  CCEcdsaVerifyUserContext_t verifyCtx

  } *operationCtx*
- uint32_t *signBuff* [2 **CC_ECPKI_ORDER_MAX_LENGTH_IN_WORDS*]

Detailed description

ECPKI data structures for FIPS certification.

Field Documentation

union { ... }   CCEcpkiKgFipsContext_t::operationCtx

Signing and verification data.

uint32_t CCEcpkiKgFipsContext_t::signBuff[2 *CC_ECPKI_ORDER_MAX_LENGTH_IN_WORDS]

Internal buffer.

The documentation for this struct was generated from the following file:

- *cc_ecpki_types.h*

1.6.19 CCEcpkiKgTempData_t Struct Reference

#include <cc_ecpki_types.h>

Data Fields

- uint32_t *ccKGIntBuff* [*CC_PKA_KG_BUFF_MAX_LENGTH_IN_WORDS*]

Detailed description

The temporary data type of the ECPKI KG.

Field Documentation

uint32_t CCEcpkiKgTempData_t::ccKGIntBuff[CC_PKA_KG_BUFF_MAX_LENGTH_IN_WORDS]

Internal buffer.

The documentation for this struct was generated from the following file:

- *cc_ecpki_types.h*

1.6.20 CCEcpkiPointAffine_t Struct Reference

#include <cc_ecpki_types.h>

Data Fields

- uint32_t *x* [*CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS*]
- uint32_t *y* [*CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS*]

Detailed description

The structure containing the EC point in affine coordinates and little endian form.

Field Documentation

uint32_t CCEcpkiPointAffine_t::x[CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS]

The X coordinate of the point.

uint32_t CCEcpkiPointAffine_t::y[CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS]

The Y coordinate of the point.

The documentation for this struct was generated from the following file:

- *cc_ecpki_types.h*

1.6.21 CCEcpkiPrivKey_t Struct Reference

#include <cc_ecpki_types.h>

Data Fields

- uint32_t *PrivKey* [*CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS*+1]
- *CCEcpkiDomain_t domain*
- *CCEcpkiScaProtection_t scaProtection*

Detailed description

The structure containing the data of the private key.

Field Documentation

CCEcpkiDomain_t CCEcpkiPrivKey_t::domain

The EC domain.

uint32_t CCEcpkiPrivKey_t::PrivKey[*CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS*+1]

The data of the private key.

CCEcpkiScaProtection_t CCEcpkiPrivKey_t::scaProtection

The SCA protection mode.

The documentation for this struct was generated from the following file:

- *cc_ecpki_types.h*

1.6.22 CCEcpkiPublKey_t Struct Reference

#include <cc_ecpki_types.h>

Data Fields

- uint32_t *x* [*CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS*]
- uint32_t *y* [*CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS*]
- *CCEcpkiDomain_t domain*
- uint32_t *pointType*

Detailed description

The structure containing the public key in affine coordinates.

Field Documentation

CCEcpkiDomain_t CCEcpkiPublKey_t::domain

The EC Domain.

uint32_t CCEcpkiPublKey_t::pointType

The point type.

uint32_t CCEcpkiPublKey_t::x[*CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS*]

The X coordinate of the public key.

uint32_t CCEcpkiPublKey_t::y[*CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS*]

The Y coordinate of the public key.

The documentation for this struct was generated from the following file:

- *cc_ecpki_types.h*

1.6.23 CCEcpkiUserPrivKey_t Struct Reference

The user structure prototype of the EC private key.

#include <cc_ecpki_types.h>

Data Fields

- uint32_t *valid_tag*
- uint32_t *PrivKeyDbBuff* [(sizeof(*CCEcpkiPrivKey_t*)+3)/4]

Detailed description

The user structure prototype of the EC private key.

This structure must be saved by the user. It is used as input to ECC functions, for example, CC_EcdsaSign().

Field Documentation

uint32_t CCEcpkiUserPrivKey_t::PrivKeyDbBuff[(sizeof(*CCEcpkiPrivKey_t*)+3)/4]

The data of the private key.

uint32_t CCEcpkiUserPrivKey_t::valid_tag

The validation tag.

The documentation for this struct was generated from the following file:

- *cc_ecpki_types.h*

1.6.24 CCEcpkiUserPublKey_t Struct Reference

The user structure prototype of the EC public key.

#include <cc_ecpki_types.h>

Data Fields

- uint32_t *valid_tag*
- uint32_t *PublKeyDbBuff* [(sizeof(*CCEcpkiPublKey_t*)+3)/4]

Detailed description

The user structure prototype of the EC public key.

This structure must be saved by the user. It is used as input to ECC functions, for example, CC_EcdsaVerify().

Field Documentation

uint32_t CCEcpkiUserPublKey_t::PublKeyDbBuff[(sizeof(*CCEcpkiPublKey_t*)+3)/4]

The data of the public key.

uint32_t CCEcpkiUserPublKey_t::valid_tag

The validation tag.

The documentation for this struct was generated from the following file:

- *cc_ecpki_types.h*

1.6.25 CCHashUserContext_t Struct Reference

The context prototype of the user.

#include <cc_hash_defs.h>

Data Fields

- uint32_t *buff* [*CC_HASH_USER_CTX_SIZE_IN_WORDS*]

Detailed description

The context prototype of the user.

The argument type that is passed by the user to the hash APIs. The context saves the state of the operation, and must be saved by the user until the end of the API flow.

Field Documentation

uint32_t CCHashUserContext_t::buff[*CC_HASH_USER_CTX_SIZE_IN_WORDS*]

The internal buffer

The documentation for this struct was generated from the following file:

- *cc_hash_defs.h*

1.6.26 CCRndContext_t Struct Reference

#include <cc_rnd_common.h>

Data Fields

- void * *rndState*
- void * *entropyCtx*
- *CCRndGenerateVectWorkFunc_t* *rndGenerateVectFunc*

Detailed description

The definition of the RND context that includes the CryptoCell RND state structure, and a function pointer for the RND-generation function.

Field Documentation

void* CCRndContext_t::entropyCtx

A pointer to the entropy context.

Note

This pointer should be allocated and assigned before calling *CC_LibInit()*.

CCRndGenerateVectWorkFunc_t CCRndContext_t::rndGenerateVectFunc

A pointer to the user-given function for generation of a random vector.

void* CCRndContext_t::rndState

A pointer to the internal state of the RND.

Note

This pointer should be allocated in a physical and contiguous memory, accessible to the CryptoCell DMA. This pointer should be allocated and assigned before calling *CC_LibInit()*.

The documentation for this struct was generated from the following file:

- *cc_rnd_common.h*

1.6.27 CCRndState_t Struct Reference

The structure for the RND state.

#include <cc_rnd_common.h>

Data Fields

- uint32_t *TrngProcesState*

Detailed description

The structure for the RND state.

This includes internal data that must be saved by the user between boots.

Field Documentation

uint32_t CCRndState_t::TrngProcesState

The TRNG process state used internally in the code

The documentation for this struct was generated from the following file:

- *cc_rnd_common.h*

1.6.28 CCRndWorkBuff_t Struct Reference

#include <cc_rnd_common.h>

Data Fields

- uint32_t *ccRndIntWorkBuff* [*CC_RND_WORK_BUFFER_SIZE_WORDS*]

Detailed description

The definition of the RAM buffer, for internal use in instantiation or reseeding operations.

Field Documentation

uint32_t CCRndWorkBuff_t::ccRndIntWorkBuff[*CC_RND_WORK_BUFFER_SIZE_WORDS*]

The internal buffer.

The documentation for this struct was generated from the following file:

- *cc_rnd_common.h*

1.6.29 CCSbCertInfo_t Struct Reference

#include <secureboot_defs.h>

Data Fields

- uint32_t *otpVersion*
- CCSbPubKeyIndexType_t *keyIndex*
- uint32_t *activeMinSwVersionVal*
- CCHashResult_t *pubKeyHash*
- uint32_t *initDataFlag*

Detailed description

Input or output structure to the Secure Boot verification API.

Field Documentation

uint32_t CCSbCertInfo_t::activeMinSwVersionVal

The value of the SW version for the certificate-chain.

uint32_t CCSbCertInfo_t::initDataFlag

The initialization indication. Internal flag.

CCSbPubKeyIndexType_t CCSbCertInfo_t::keyIndex

The key hash to retrieve:

- The 128-bit Hbk0.
- The 128-bit Hbk1.
- The 256-bit Hbk.

uint32_t CCSbCertInfo_t::otpVersion

The NV counter saved in OTP.

CCHashResult_t CCSbCertInfo_t::pubKeyHash

- In: The hash of the public key (N||Np), to compare to the public key stored in the certificate.

- Out: The hash of the public key (N||Np) stored in the certificate, to be used for verification of the public key of the next certificate in the chain.

The documentation for this struct was generated from the following file:

- *secureboot_defs.h*

1.6.30 EcdsaSignContext_t Struct Reference

#include <cc_ecpki_types.h>

Data Fields

- *CCEcpkiUserPrivKey_t ECDSA_SignerPrivKey*
- *mbedtls_md_context_t hash_ctx*
- *CCHashResultBuf_t hashResult*
- uint32_t *hashResultSizeWords*
- *CCEcpkiHashOpMode_t hashMode*
- *CCEcdsaSignIntBuff_t ecdsaSignIntBuff*

Detailed description

The context definition for the signing operation.

Field Documentation

CCEcpkiUserPrivKey_t EcdsaSignContext_t::ECDSA_SignerPrivKey

The data of the private key.

CCEcdsaSignIntBuff_t EcdsaSignContext_t::ecdsaSignIntBuff

Internal buffer.

mbedtls_md_context_t EcdsaSignContext_t::hash_ctx

The hash context.

CCEcpkiHashOpMode_t EcdsaSignContext_t::hashMode

The hash mode.

CCHashResultBuf_t EcdsaSignContext_t::hashResult

The hash result buffer.

uint32_t EcdsaSignContext_t::hashResultSizeWords

The size of the hash result in words.

The documentation for this struct was generated from the following file:

- *cc_ecpki_types.h*

1.6.31 EcdsaVerifyContext_t Struct Reference

#include <cc_ecpki_types.h>

Data Fields

- *CCEcpkiUserPublKey_t ECDSA_SignerPublKey*
- *mbedtls_md_context_t hash_ctx*
- *CCHashResultBuf_t hashResult*
- uint32_t *hashResultSizeWords*
- *CCEcpkiHashOpMode_t hashMode*
- *CCEcdsaVerifyIntBuff_t ccEcdsaVerIntBuff*

Detailed description

The context definition for verification operation.

Field Documentation

***CCEcdsaVerifyIntBuff_t* EcdsaVerifyContext_t::ccEcdsaVerIntBuff**

Internal buffer.

***CCEcpkiUserPublKey_t* EcdsaVerifyContext_t::ECDSA_SignerPublKey**

The data of the public key.

***mbedtls_md_context_t* EcdsaVerifyContext_t::hash_ctx**

The hash context.

***CCEcpkiHashOpMode_t* EcdsaVerifyContext_t::hashMode**

The hash mode.

***CCHashResultBuf_t* EcdsaVerifyContext_t::hashResult**

The hash result.

uint32_t EcdsaVerifyContext_t::hashResultSizeWords

The size of the hash result in words.

The documentation for this struct was generated from the following file:

- *cc_ecpki_types.h*

1.6.32 HmacHash_t Struct Reference

#include <cc_general_defs.h>

Data Fields

- uint16_t *hashResultSize*
- *CCHashOperationMode_t hashMode*

Detailed description

Hash parameters for HMAC operation.

Field Documentation

***CCHashOperationMode_t* HmacHash_t::hashMode**

The hash operation mode.

uint16_t HmacHash_t::hashResultSize

The size of the HMAC hash result.

The documentation for this struct was generated from the following file:

- *cc_general_defs.h*

1.6.33 mbedtls_aes_context Struct Reference

The AES context-type definition.

#include <aes_alt.h>

Data Fields

- uint32_t **buf** [MBEDTLS_AES_CONTEXT_SIZE_IN_WORDS]

Detailed description

The AES context-type definition.

Field Documentation

uint32_t buf [MBEDTLS_AES_CONTEXT_SIZE_IN_WORDS]

Unaligned data buffer.

The documentation for this struct was generated from the following file:

- *aes_alt.h*

1.6.34 mbedtls_ccm_context Struct Reference

The CCM context-type definition. The CCM context is passed to the APIs called.

#include <ccm_alt.h>

Data Fields

- uint32_t **buf** [MBEDTLS_CCM_CONTEXT_SIZE_IN_WORDS]

Detailed description

The CCM context-type definition. The CCM context is passed to the APIs called.

Field Documentation

uint32_t buf [MBEDTLS_CCM_CONTEXT_SIZE_IN_WORDS]

Unaligned data buffer.

The documentation for this struct was generated from the following file:

- *ccm_alt.h*

1.6.35 mbedtls_chacha_user_context Struct Reference

The context prototype of the user.

#include <mbedtls_cc_chacha.h>

Data Fields

- uint32_t *buff* [*CC_CHACHA_USER_CTX_SIZE_IN_WORDS*]

Detailed description

The context prototype of the user.

The argument type that is passed by the user to the ChaCha API.

The context saves the state of the operation. It must be saved by the user until the end of the API flow, for example, until *mbedtls_chacha_free* is called.

Field Documentation

uint32_t mbedtls_chacha_user_context::buff[*CC_CHACHA_USER_CTX_SIZE_IN_WORDS*]

The context buffer for internal use

The documentation for this struct was generated from the following file:

- *mbedtls_cc_chacha.h*

1.6.36 mbedtls_cipher_context_t Struct Reference

#include <cipher.h>

Data Fields

- const *mbedtls_cipher_info_t* * *cipher_info*
- int *key_bitlen*
- *mbedtls_operation_t operation*
- unsigned char *unprocessed_data* [*MBEDTLS_MAX_BLOCK_LENGTH*]
- size_t *unprocessed_len*
- unsigned char *iv* [*MBEDTLS_MAX_IV_LENGTH*]
- size_t *iv_size*
- void * *cipher_ctx*

Detailed description

Generic cipher context.

Field Documentation

void* mbedtls_cipher_context_t::cipher_ctx

The cipher-specific context.

const *mbedtls_cipher_info_t** mbedtls_cipher_context_t::cipher_info

Information about the associated cipher.

unsigned char mbedtls_cipher_context_t::iv[*MBEDTLS_MAX_IV_LENGTH*]

Current IV or NONCE_COUNTER for CTR-mode.

size_t mbedtls_cipher_context_t::iv_size

IV size in Bytes, for ciphers with variable-length IVs.

int mbedtls_cipher_context_t::key_bitlen

Key length to use.

mbedtls_operation_t mbedtls_cipher_context_t::operation

Operation that the key of the context has been initialized for.

unsigned char mbedtls_cipher_context_t::unprocessed_data[*MBEDTLS_MAX_BLOCK_LENGTH*]

Buffer for input that has not been processed yet.

size_t mbedtls_cipher_context_t::unprocessed_len

Number of Bytes that have not been processed yet.

The documentation for this struct was generated from the following file:

- *cipher.h*

1.6.37 mbedtls_cipher_info_t Struct Reference

#include <cipher.h>

Data Fields

- *mbedtls_cipher_type_t type*
- *mbedtls_cipher_mode_t mode*
- unsigned int *key_bitlen*
- const char * *name*
- unsigned int *iv_size*
- int *flags*
- unsigned int *block_size*
- const *mbedtls_cipher_base_t* * *base*

Detailed description

Cipher information. Allows calling cipher functions in a generic way.

Field Documentation

const *mbedtls_cipher_base_t** mbedtls_cipher_info_t::base

Struct for base cipher information and functions.

unsigned int mbedtls_cipher_info_t::block_size

The block size, in Bytes.

int mbedtls_cipher_info_t::flags

Flags to set. For example, if the cipher supports variable IV sizes or variable key sizes.

unsigned int mbedtls_cipher_info_t::iv_size

IV or nonce size, in Bytes. For ciphers that accept variable IV sizes, this is the recommended size.

unsigned int mbedtls_cipher_info_t::key_bitlen

The cipher key length, in bits. This is the default length for variable sized ciphers. Includes parity bits for ciphers like DES.

mbedtls_cipher_mode_t mbedtls_cipher_info_t::mode

The cipher mode. For example, MBEDTLS_MODE_CBC .

const char* mbedtls_cipher_info_t::name

Name of the cipher.

mbedtls_cipher_type_t mbedtls_cipher_info_t::type

Full cipher identifier. For example, MBEDTLS_CIPHER_AES_256_CBC.

The documentation for this struct was generated from the following file:

- *cipher.h*

1.6.38 mbedtls_cmac_context_t Struct Reference

The CMAC context-type definition.

#include <cmac_alt.h>

Data Fields

- uint32_t buf [MBEDTLS_CMAC_CONTEXT_SIZE_IN_WORDS]

Detailed description

The CMAC context-type definition.

Field Documentation

uint32_t mbedtls_cmac_context_t::buf[MBEDTLS_CMAC_CONTEXT_SIZE_IN_WORDS]

An internal buffer.

The documentation for this struct was generated from the following file:

- *cmac_alt.h*

1.6.39 mbedtls_ctr_drbg_context Struct Reference

The CTR_DRBG context structure.

#include <ctr_drbg.h>

Data Fields

- unsigned char *counter* [16]
- int *reseed_counter*
- int *prediction_resistance*
- size_t *entropy_len*
- int *reseed_interval*
- *mbedtls_aes_context* *aes_ctx*
- int(* *f_entropy*)(void *, unsigned char *, size_t)

- void * *p_entropy*

Detailed description

The CTR_DRBG context structure.

Field Documentation

mbedtls_aes_context mbedtls_ctr_drbg_context::aes_ctx

The AES context.

unsigned char mbedtls_ctr_drbg_context::counter[16]

The counter (V).

size_t mbedtls_ctr_drbg_context::entropy_len

The amount of entropy grabbed on each seed or reseed operation.

int(* mbedtls_ctr_drbg_context::f_entropy) (void *, unsigned char *, size_t)

The entropy callback function.

void* mbedtls_ctr_drbg_context::p_entropy

The context for the entropy function.

int mbedtls_ctr_drbg_context::prediction_resistance

This determines whether prediction resistance is enabled, that is whether to systematically reseed before each random generation.

int mbedtls_ctr_drbg_context::reseed_counter

The reseed counter.

int mbedtls_ctr_drbg_context::reseed_interval

The reseed interval.

The documentation for this struct was generated from the following file:

- *ctr_drbg.h*

1.6.40 mbedtls_dhm_context Struct Reference

The DHM context structure.

#include <dhm_alt.h>

Data Fields

- size_t *len*
- mbedtls_mpi *P*
- mbedtls_mpi *G*
- mbedtls_mpi *X*
- mbedtls_mpi *GX*
- mbedtls_mpi *GY*
- mbedtls_mpi *K*
- mbedtls_mpi *RP*
- mbedtls_mpi *Vi*

- mbedtls_mpi Vf
- mbedtls_mpi pX

Detailed description

The DHM context structure.

Field Documentation

mbedtls_mpi mbedtls_dhm_context::G

The generator.

mbedtls_mpi mbedtls_dhm_context::GX

Our public key = G^X mod P .

mbedtls_mpi mbedtls_dhm_context::GY

The public key of the peer = G^Y mod P .

mbedtls_mpi mbedtls_dhm_context::K

The shared secret = G^ (XY) mod P .

size_t mbedtls_dhm_context::len

The size of P in Bytes.

mbedtls_mpi mbedtls_dhm_context::P

The prime modulus.

mbedtls_mpi mbedtls_dhm_context::pX

The previous X .

mbedtls_mpi mbedtls_dhm_context::RP

The cached value = R^2 mod P .

mbedtls_mpi mbedtls_dhm_context::Vf

The unblinding value.

mbedtls_mpi mbedtls_dhm_context::Vi

The blinding value.

mbedtls_mpi mbedtls_dhm_context::X

Our secret value.

The documentation for this struct was generated from the following file:

- *dhm_alt.h*

1.6.41 mbedtls_ecdh_context Struct Reference

The ECDH context structure.

#include <ecdh.h>

Data Fields

- *mbedtls_ecp_group grp*

- mbedtls_mpi *d*
- *mbedtls_ecp_point Q*
- *mbedtls_ecp_point Qp*
- mbedtls_mpi *z*
- int *point_format*
- *mbedtls_ecp_point Vi*
- *mbedtls_ecp_point Vf*
- mbedtls_mpi *_d*

Detailed description

The ECDH context structure.

Field Documentation

mbedtls_mpi mbedtls_ecdh_context::_d

The previous d .

mbedtls_mpi mbedtls_ecdh_context::d

The private key.

mbedtls_ecp_group mbedtls_ecdh_context::grp

The elliptic curve used.

int mbedtls_ecdh_context::point_format

The format of point export in TLS messages.

mbedtls_ecp_point mbedtls_ecdh_context::Q

The public key.

mbedtls_ecp_point mbedtls_ecdh_context::Qp

The value of the public key of the peer.

mbedtls_ecp_point mbedtls_ecdh_context::Vf

The unblinding value.

mbedtls_ecp_point mbedtls_ecdh_context::Vi

The blinding value.

mbedtls_mpi mbedtls_ecdh_context::z

The shared secret.

The documentation for this struct was generated from the following file:

- *ecdh.h*

1.6.42 mbedtls_ecp_curve_info Struct Reference

#include <ecp.h>

Data Fields

- *mbedtls_ecp_group_id grp_id*

- uint16_t *tls_id*
- uint16_t *bit_size*
- const char * *name*

Detailed description

Curve information for use by other modules

Field Documentation

uint16_t mbedtls_ecp_curve_info::bit_size

Curve size in bits

mbedtls_ecp_group_id mbedtls_ecp_curve_info::grp_id

Internal identifier

const char* mbedtls_ecp_curve_info::name

Human-friendly name

uint16_t mbedtls_ecp_curve_info::tls_id

TLS NamedCurve identifier

The documentation for this struct was generated from the following file:

- *ecp.h*

1.6.43 mbedtls_ecp_group Struct Reference

ECP group structure.

#include <ecp.h>

Data Fields

- *mbedtls_ecp_group_id id*
- mbedtls_mpi *P*
- mbedtls_mpi *A*
- mbedtls_mpi *B*
- *mbedtls_ecp_point G*
- mbedtls_mpi *N*
- size_t *pbits*
- size_t *nbits*
- unsigned int *h*
- int(* *modp*)(mbedtls_mpi *)
- int(* *t_pre*)(*mbedtls_ecp_point* *, void *)
- int(* *t_post*)(*mbedtls_ecp_point* *, void *)
- void * *t_data*
- *mbedtls_ecp_point* * *T*
- size_t *T_size*

Detailed description

ECP group structure.

We consider two types of curves equations:

- Short Weierstrass y^2 = x^3 + A x + B mod P (SEC1 + RFC 4492)
- Montgomery, y^2 = x^3 + A x^2 + x mod P (Curve25519 + draft)

In both cases, a generator G for a prime-order subgroup is fixed. In the short weierstrass, this subgroup is actually the whole curve, and its cardinal is denoted by N.

In the case of Short Weierstrass curves, our code requires that N is an odd prime. (Use odd in *mbedtls_ecp_mul()* and prime in *mbedtls_ecdsa_sign()* for blinding.)

In the case of Montgomery curves, we don't store A but (A + 2) / 4 which is the quantity actually used in the formulas. Also, nbits is not the size of N but the required size for private keys.

If modp is NULL, reduction modulo P is done using a generic algorithm. Otherwise, it must point to a function that takes an mbedtls_mpi in the range 0..2^(2*pbits)-1 and transforms it in-place in an integer of little more than pbits, so that the integer may be efficiently brought in the 0..P-1 range by a few additions or substractions. It must return 0 on success and non-zero on failure.

Field Documentation

mbedtls_mpi mbedtls_ecp_group::A

A in the equation or (A + 2) / 4.

mbedtls_mpi mbedtls_ecp_group::B

B in the equation or unused.

mbedtls_ecp_point mbedtls_ecp_group::G

Generator of the (sub)group used.

unsigned int mbedtls_ecp_group::h

Internal: 1 if the constants are static.

mbedtls_ecp_group_id mbedtls_ecp_group::id

Internal group identifier.

int(* mbedtls_ecp_group::modp) (mbedtls_mpi *)

Function for fast reduction mod P.

mbedtls_mpi mbedtls_ecp_group::N

The order of G or unused.

size_t mbedtls_ecp_group::nbits

Number of bits in P or number of bits in private keys.

mbedtls_mpi mbedtls_ecp_group::P

Prime modulus of the base field.

size_t mbedtls_ecp_group::pbits

Number of bits in P.

mbedtls_ecp_point* mbedtls_ecp_group::T

Pre-computed points for ecp_mul_comb().

void* mbedtls_ecp_group::t_data

Unused.

int(* mbedtls_ecp_group::t_post) (mbedtls_ecp_point *, void *)

Unused.

int(* mbedtls_ecp_group::t_pre) (mbedtls_ecp_point *, void *)

Unused.

size_t mbedtls_ecp_group::T_size

Number for pre-computed points.

The documentation for this struct was generated from the following file:

- *ecp.h*

1.6.44 mbedtls_ecp_keypair Struct Reference

ECP key pair structure.

#include <ecp.h>

Data Fields

- *mbedtls_ecp_group grp*
- mbedtls_mpi *d*
- *mbedtls_ecp_point Q*

Detailed description

ECP key pair structure.

A generic key pair that could be used for ECDSA, fixed ECDH, etc.

Note

Members purposefully in the same order as struc mbedtls_ecdsa_context.

Field Documentation

mbedtls_mpi mbedtls_ecp_keypair::d

Our secret value.

mbedtls_ecp_group mbedtls_ecp_keypair::grp

Elliptic curve and base point.

mbedtls_ecp_point mbedtls_ecp_keypair::Q

Our public value.

The documentation for this struct was generated from the following file:

- *ecp.h*

1.6.45 mbedtls_ecp_point Struct Reference

ECP point structure (jacobian coordinates).

#include <ecp.h>

Data Fields

- mbedtls_mpi *X*
- mbedtls_mpi *Y*
- mbedtls_mpi *Z*

Detailed description

ECP point structure (jacobian coordinates)

Note

All functions expect and return points satisfying the following condition: Z == 0 or Z == 1. (Other values of Z are used by internal functions only.) The point is zero, or "at infinity", if Z == 0. Otherwise, X and Y are its standard (affine) coordinates.

Field Documentation

mbedtls_mpi mbedtls_ecp_point::X

the point's X coordinate

mbedtls_mpi mbedtls_ecp_point::Y

the point's Y coordinate

mbedtls_mpi mbedtls_ecp_point::Z

the point's Z coordinate

The documentation for this struct was generated from the following file:

- *ecp.h*

1.6.46 mbedtls_gcm_context Struct Reference

The GCM context structure.

#include <gcm_alt.h>

Data Fields

- uint32_t *buf*[MBEDTLS_GCM_CONTEXT_SIZE_IN_WORDS]

Detailed description

The GCM context structure.

Field Documentation

uint32_t buf[MBEDTLS_GCM_CONTEXT_SIZE_IN_WORDS]

Internal buffer.

The documentation for this struct was generated from the following file:

- *gcm_alt.h*

1.6.47 mbedtls_md_context_t Struct Reference

#include <md.h>

Data Fields

- const *mbedtls_md_info_t* * *md_info*
- void * *md_ctx*
- void * *hmac_ctx*

Detailed description

The generic message-digest context.

Field Documentation

void* mbedtls_md_context_t::hmac_ctx

The HMAC part of the context.

void* mbedtls_md_context_t::md_ctx

The digest-specific context.

const *mbedtls_md_info_t** mbedtls_md_context_t::md_info

Information about the associated message digest.

The documentation for this struct was generated from the following file:

- *md.h*

1.6.48 mbedtls_mng_apbcconfig Union Reference

#include <mbedtls_cc_mng.h>

Data Fields

- uint8_t *apbcConfigVal*
- struct {

  uint8_t isApbcSecAccessMode:1

  uint8_t isApbcSecModeLock:1

  uint8_t isApbcPrivAccessMode:1

  uint8_t isApbcPrivModeLock:1

  uint8_t isApbcInstAccessMode:1

  uint8_t isApbcInstModeLock:1

  uint8_t rfu:2

  } *apbcbits*

Detailed description

Input to the *mbedtls_mng_apbc_config_set()* function.

Field Documentation

struct { ... } mbedtls_mng_apbcconfig::apbcbits

APB-C bits.

uint8_t mbedtls_mng_apbcconfig::apbcConfigVal

APB-C configuration values.

The documentation for this union was generated from the following file:

- *mbedtls_cc_mng.h*

1.6.49 mbedtls_platform_context Struct Reference

The platform context structure.

#include <platform.h>

Data Fields

- char *dummy*

Detailed description

The platform context structure.

Note

This structure may be used to assist platform-specific setup or teardown operations.

Field Documentation

char mbedtls_platform_context::dummy

Placeholder member, as empty structs are not portable.

The documentation for this struct was generated from the following file:

- *platform.h*

1.6.50 mbedtls_rsa_context Struct Reference

The RSA context structure.

#include <rsa_alt.h>

Data Fields

- int *ver*
- size_t *len*
- mbedtls_mpi *N*
- mbedtls_mpi *E*
- mbedtls_mpi *D*

- mbedtls_mpi *P*
- mbedtls_mpi *Q*
- mbedtls_mpi *DP*
- mbedtls_mpi *DQ*
- mbedtls_mpi *QP*
- mbedtls_mpi *RN*
- mbedtls_mpi *RP*
- mbedtls_mpi *RQ*
- mbedtls_mpi *Vi*
- mbedtls_mpi *Vf*
- mbedtls_mpi *NP*
- mbedtls_mpi *BQP*
- mbedtls_mpi *BPP*
- int *padding*
- int *hash_id*

Detailed description

The RSA context structure.

Note

Direct manipulation of the members of this structure is deprecated. All manipulation should instead be done through the public interface functions.

Field Documentation

mbedtls_mpi mbedtls_rsa_context::BPP

Barrett mod P tag PP for P-factor.

mbedtls_mpi mbedtls_rsa_context::BQP

Barrett mod Q tag QP for Q-factor.

mbedtls_mpi mbedtls_rsa_context::D

The private exponent.

mbedtls_mpi mbedtls_rsa_context::DP

D % (P - 1)

mbedtls_mpi mbedtls_rsa_context::DQ

D % (Q - 1)

mbedtls_mpi mbedtls_rsa_context::E

The public exponent.

int mbedtls_rsa_context::hash_id

Hash identifier of mbedtls_md_type_t type, as specified in *md.h* for use in the MGF mask generating function used in the EME-OAEP and EMSA-PSS encodings.

size_t mbedtls_rsa_context::len

The size of N in Bytes.

mbedtls_mpi mbedtls_rsa_context::N

The public modulus.

mbedtls_mpi mbedtls_rsa_context::NP

Barrett mod N tag NP for N-modulus.

mbedtls_mpi mbedtls_rsa_context::P

The first prime factor.

int mbedtls_rsa_context::padding

Selects padding mode: *MBEDTLS_RSA_PKCS_V15* for 1.5 padding and *MBEDTLS_RSA_PKCS_V21* for OAEP or PSS.

mbedtls_mpi mbedtls_rsa_context::Q

The second prime factor.

mbedtls_mpi mbedtls_rsa_context::QP

1 / (Q % P)

mbedtls_mpi mbedtls_rsa_context::RN

cached R^2 mod N

mbedtls_mpi mbedtls_rsa_context::RP

cached R^2 mod P

mbedtls_mpi mbedtls_rsa_context::RQ

cached R^2 mod Q

int mbedtls_rsa_context::ver

Always 0.

mbedtls_mpi mbedtls_rsa_context::Vf

The cached un-blinding value.

mbedtls_mpi mbedtls_rsa_context::Vi

The cached blinding value.

The documentation for this struct was generated from the following file:

- *rsa.h*

1.6.51 mbedtls_sha1_context Struct Reference

The SHA-1 context structure.

#include <sha1_alt.h>

Data Fields

- uint32_t buff [CC_HASH_USER_CTX_SIZE_IN_WORDS]

Detailed description

The SHA-1 context structure.

Warning

SHA-1 is considered a weak message digest and its use constitutes a security risk. We recommend considering stronger message digests instead.

Field Documentation

uint32_t mbedtls_sha1_context::buff[CC_HASH_USER_CTX_SIZE_IN_WORDS]

Internal buffer.

The documentation for this struct was generated from the following file:

- *sha1_alt.h*

1.6.52 mbedtls_sha256_context Struct Reference

The SHA-256 context structure.

#include <sha256_alt.h>

Data Fields

- uint32_t buff [CC_HASH_USER_CTX_SIZE_IN_WORDS]

Detailed description

The SHA-256 context structure.

Field Documentation

uint32_t mbedtls_sha256_context::buff[CC_HASH_USER_CTX_SIZE_IN_WORDS]

Internal buffer.

The documentation for this struct was generated from the following file:

- *sha256_alt.h*

1.6.53 mbedtls_sha512_context Struct Reference

The SHA-512 context structure.

#include <sha512.h>

Data Fields

- uint64_t *total* [2]
- uint64_t *state* [8]
- unsigned char *buffer* [128]
- int *is384*

Detailed description

The SHA-512 context structure.

The structure is used both for SHA-384 and for SHA-512 checksum calculations. The choice between these two is made in the call to *mbedtls_sha512_starts_ret()*.

Field Documentation

unsigned char mbedtls_sha512_context::buffer[128]

The data block being processed.

int mbedtls_sha512_context::is384

Determines which function to use.

- 0: Use SHA-512.
- 1: Use SHA-384.

uint64_t mbedtls_sha512_context::state[8]

The intermediate digest state.

uint64_t mbedtls_sha512_context::total[2]

The number of Bytes processed.

The documentation for this struct was generated from the following file:

- *sha512.h*

1.6.54 mbedtls_srp_context Struct Reference

#include <mbedtls_cc_srp.h>

Data Fields

- *mbedtls_srp_entity_t* *srpType*
- *mbedtls_srp_version_t* *srpVer*
- *mbedtls_srp_group_param* *groupParam*
- *CCHashOperationMode_t* *hashMode*
- size_t *hashDigestSize*
- *CCRndContext_t* * *pRndCtx*
- *mbedtls_srp_modulus* *ephemPriv*
- size_t *ephemPrivSize*
- *mbedtls_srp_digest* *userNameDigest*
- *mbedtls_srp_digest* *credDigest*
- *mbedtls_srp_digest* *kMult*

Detailed description

The SRP context prototype

Field Documentation

mbedtls_srp_digest mbedtls_srp_context::credDigest

The cred digest.

mbedtls_srp_modulus mbedtls_srp_context::ephemPriv

The modulus.

size_t mbedtls_srp_context::ephemPrivSize

The modulus size.

mbedtls_srp_group_param mbedtls_srp_context::groupParam

The group parameter including the modulus information.

size_t mbedtls_srp_context::hashDigestSize

The hash digest size.

CCHashOperationMode_t mbedtls_srp_context::hashMode

The hash mode.

mbedtls_srp_digest mbedtls_srp_context::kMult

The SRP K multiplier.

CCRndContext_t* mbedtls_srp_context::pRndCtx

A pointer to the RND context.

mbedtls_srp_entity_t mbedtls_srp_context::srpType

The SRP entitiy type.

mbedtls_srp_version_t mbedtls_srp_context::srpVer

The SRP version.

mbedtls_srp_digest mbedtls_srp_context::userNameDigest

The user-name digest.

The documentation for this struct was generated from the following file:

- *mbedtls_cc_srp.h*

1.6.55 mbedtls_srp_group_param Struct Reference

Group parameters for the SRP.

#include <mbedtls_cc_srp.h>

Data Fields

- *mbedtls_srp_modulus* *modulus*
- uint8_t *gen*
- size_t *modSizeInBits*
- uint32_t *validNp*
- uint32_t *Np* [*CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS*]

Detailed description

Group parameters for the SRP.

Defines the modulus and the generator used.

Field Documentation

uint8_t mbedtls_srp_group_param::gen

The SRP generator.

size_t mbedtls_srp_group_param::modSizeInBits

The size of the SRP modulus in bits.

mbedtls_srp_modulus mbedtls_srp_group_param::modulus

The SRP modulus.

uint32_t mbedtls_srp_group_param::Np[CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS]

The SRP Np buffer.

uint32_t mbedtls_srp_group_param::validNp

The valid SRP Np.

The documentation for this struct was generated from the following file:

- *mbedtls_cc_srp.h*

1.6.56 mbedtls_util_keydata Struct Reference

#include <mbedtls_cc_util_defs.h>

Data Fields

- uint8_t * *pKey*
- size_t *keySize*

Detailed description

Key data.

Field Documentation

size_t mbedtls_util_keydata::keySize

The size of the key in Bytes.

uint8_t* mbedtls_util_keydata::pKey

A pointer to the key.

The documentation for this struct was generated from the following file:

- *mbedtls_cc_util_defs.h*

1.7 File Documentation

1.7.1 aes.h File Reference

This file contains the Mbed TLS AES APIs.

The Advanced Encryption Standard (AES) specifies a FIPS-approved cryptographic algorithm that can be used to protect electronic data.

#include "config.h"

#include <stddef.h>

#include <stdint.h>

Data structures

- struct *mbedtls_aes_context*

The AES context-type definition. Macros

- #define *MBEDTLS_AES_ENCRYPT* 1
- #define *MBEDTLS_AES_DECRYPT* 0
- #define *MBEDTLS_ERR_AES_INVALID_KEY_LENGTH* -0x0020
- #define *MBEDTLS_ERR_AES_INVALID_INPUT_LENGTH* -0x0022
- #define *MBEDTLS_ERR_AES_FEATURE_UNAVAILABLE* -0x0023
- #define *MBEDTLS_ERR_AES_HW_ACCEL_FAILED* -0x0025
- #define MBEDTLS_DEPRECATED

Functions

- void *mbedtls_aes_init* (*mbedtls_aes_context* *ctx)

  This function initializes the specified AES context.
- void *mbedtls_aes_free* (*mbedtls_aes_context* *ctx)

  This function releases and clears the specified AES context.
- int *mbedtls_aes_setkey_enc* (*mbedtls_aes_context* *ctx, const unsigned char *key, unsigned int keybits)

  This function sets the encryption key.
- int *mbedtls_aes_setkey_dec* (*mbedtls_aes_context* *ctx, const unsigned char *key, unsigned int keybits)

  This function sets the decryption key.
- int *mbedtls_aes_crypt_ecb* (*mbedtls_aes_context* *ctx, int mode, const unsigned char input[16], unsigned char output[16])

  This function performs an AES single-block encryption or decryption operation.
- int *mbedtls_internal_aes_encrypt* (*mbedtls_aes_context* *ctx, const unsigned char input[16], unsigned char output[16])

Internal AES block encryption function. This is only exposed to allow overriding it using MBEDTLS_AES_ENCRYPT_ALT .

- int *mbedtls_internal_aes_decrypt* (*mbedtls_aes_context* *ctx, const unsigned char input[16], unsigned char output[16])

  Internal AES block decryption function. This is only exposed to allow overriding it using see MBEDTLS_AES_DECRYPT_ALT .

- MBEDTLS_DEPRECATED void *mbedtls_aes_encrypt* (*mbedtls_aes_context* *ctx, const unsigned char input[16], unsigned char output[16])

  Deprecated internal AES block encryption function without return value.

- MBEDTLS_DEPRECATED void *mbedtls_aes_decrypt* (*mbedtls_aes_context* *ctx, const unsigned char input[16], unsigned char output[16])

  Deprecated internal AES block decryption function without return value.

- int *mbedtls_aes_self_test* (int verbose)

  Checkup routine.

Detailed description

The Advanced Encryption Standard (AES) specifies a FIPS-approved cryptographic algorithm that can be used to protect electronic data.

The AES algorithm is a symmetric block cipher that can encrypt and decrypt information. For more information, see FIPS Publication 197: Advanced Encryption Standard and ISO/IEC 18033-2:2006: Information technology – Security techniques – Encryption algorithms – Part 2: Asymmetric ciphers .

Macro definition documentation

#define MBEDTLS_AES_DECRYPT 0

AES decryption.

#define MBEDTLS_AES_ENCRYPT 1

AES encryption.

#define MBEDTLS_ERR_AES_FEATURE_UNAVAILABLE -0x0023

Feature not available. For example, an unsupported AES key size.

#define MBEDTLS_ERR_AES_HW_ACCEL_FAILED -0x0025

AES hardware accelerator failed.

#define MBEDTLS_ERR_AES_INVALID_INPUT_LENGTH -0x0022

Invalid data input length.

#define MBEDTLS_ERR_AES_INVALID_KEY_LENGTH -0x0020

Invalid key length.

Function documentation

int mbedtls_aes_crypt_ecb (*mbedtls_aes_context* * ctx, int mode, const unsigned char input[16], unsigned char output[16])

This function performs an AES single-block encryption or decryption operation.

It performs the operation defined in the mode parameter (encrypt or decrypt), on the input data buffer defined in the input parameter.

mbedtls_aes_init(), and either *mbedtls_aes_setkey_enc()* or *mbedtls_aes_setkey_dec()* must be called before the first call to this API with the same context.

Parameters:

Parameter	Description
`ctx`	The AES context to use for encryption or decryption.
`mode`	The AES operation: *MBEDTLS_AES_ENCRYPT* or *MBEDTLS_AES_DECRYPT*.
`input`	The 16-Byte buffer holding the input data.
`output`	The 16-Byte buffer holding the output data.

Returns:

0 on success.

MBEDTLS_DEPRECATED void mbedtls_aes_decrypt (*mbedtls_aes_context* * ctx, const unsigned char input[16], unsigned char output[16])

Deprecated internal AES block decryption function without return value.

***Deprecated*:**

Superseded by mbedtls_aes_decrypt_ext() in 2.5.0.

Parameters:

Parameter	Description
`ctx`	The AES context to use for decryption.
`input`	Ciphertext block.
`output`	Output (plaintext) block.

MBEDTLS_DEPRECATED void mbedtls_aes_encrypt (*mbedtls_aes_context* * ctx, const unsigned char input[16], unsigned char output[16])

Deprecated internal AES block encryption function without return value.

***Deprecated*:**

Superseded by mbedtls_aes_encrypt_ext() in 2.5.0.

Parameters:

Parameter	Description
`ctx`	The AES context to use for encryption.
`input`	Plaintext block.
`output`	Output (ciphertext) block.

void mbedtls_aes_free (*mbedtls_aes_context* * ctx)

This function releases and clears the specified AES context.

Parameters:

Parameter	Description
`ctx`	The AES context to clear.

void mbedtls_aes_init (mbedtls_aes_context * ctx)

This function initializes the specified AES context.

It must be the first API called before using the context.

Parameters:

Parameter	Description
`ctx`	The AES context to initialize.

int mbedtls_aes_self_test (int verbose)

Checkup routine.

Returns:

0 on success, or 1 on failure.

int mbedtls_aes_setkey_dec (mbedtls_aes_context * ctx, const unsigned char * key, unsigned int keybits)

This function sets the decryption key.

Parameters:

Parameter	Description
`ctx`	The AES context to which the key should be bound.
`key`	The decryption key.
`keybits`	The size of data passed. Valid options are: • 128 bits • 192 bits • 256 bits

Returns:

0 on success, or *MBEDTLS_ERR_AES_INVALID_KEY_LENGTH* on failure.

int mbedtls_aes_setkey_enc (mbedtls_aes_context * ctx, const unsigned char * key, unsigned int keybits)

This function sets the encryption key.

Parameters:

Parameter	Description
`ctx`	The AES context to which the key should be bound.
`key`	The encryption key.
`keybits`	The size of data passed in bits. Valid options are: • 128 bits • 192 bits • 256 bits

Returns:

0 on success or *MBEDTLS_ERR_AES_INVALID_KEY_LENGTH* on failure.

int mbedtls_internal_aes_decrypt (*mbedtls_aes_context* * ctx, const unsigned char input[16], unsigned char output[16])

Internal AES block decryption function. This is only exposed to allow overriding it using see MBEDTLS_AES_DECRYPT_ALT .

Parameters:

Parameter	Description
`ctx`	The AES context to use for decryption.
`input`	The ciphertext block.
`output`	The output (plaintext) block.

Returns:

0 on success.

int mbedtls_internal_aes_encrypt (*mbedtls_aes_context* * ctx, const unsigned char input[16], unsigned char output[16])

Internal AES block encryption function. This is only exposed to allow overriding it using MBEDTLS_AES_ENCRYPT_ALT .

Parameters:

Parameter	Description
`ctx`	The AES context to use for encryption.
`input`	The plaintext block.
`output`	The output (ciphertext) block.

Returns:

0 on success.

1.7.2 bootimagesverifier_def.h File Reference

This file contains definitions used for the Secure Boot and Secure Debug APIs.

#include "cc_pal_types.h"

Macros

- #define *CC_SB_MAX_NUM_OF_IMAGES* 16
- #define *CC_SB_MAX_CERT_SIZE_IN_BYTES* (0xB10)
- #define *CC_SB_MAX_CERT_SIZE_IN_WORDS* (*CC_SB_MAX_CERT_SIZE_IN_BYTES*/*CC_32BIT_WORD_SIZE*)
- #define *CC_SB_MIN_DBG_WORKSPACE_SIZE_IN_BYTES* (0x350)
- #define *CC_SB_MIN_WORKSPACE_SIZE_IN_BYTES* (*CC_SB_MAX_CERT_SIZE_IN_BYTES* + *CC_MAX*(*CC_SB_MIN_DBG_WORKSPACE_SIZE_IN_BYTES*, CC_DOUBLE_BUFFER_MAX_SIZE_IN_BYTES))

The minimal size of the Secure Boot workspace in Bytes.

Detailed description

This file contains definitions used for the Secure Boot and Secure Debug APIs.

1.7.3 cc_address_defs.h File Reference

This file contains general definitions for CryptoCell APIs.

Typedefs

- typedef uint32_t *CCSramAddr_t*
- typedef uint32_t *CCDmaAddr_t*

Detailed description

This file contains general definitions for CryptoCell APIs.

1.7.4 cc_aes_defs.h File Reference

This file contains the type definitions that are used by the CryptoCell AES APIs.

#include "cc_pal_types.h"

#include "cc_aes_defs_proj.h"

Data structures

- struct *CCAesUserContext_t*
- The context prototype of the user. struct *CCAesUserKeyData_t*
- struct *CCAesHwKeyData_t*

Macros

- #define *CC_AES_CRYPTO_BLOCK_SIZE_IN_WORDS* 4
- #define *CC_AES_BLOCK_SIZE_IN_BYTES* (*CC_AES_CRYPTO_BLOCK_SIZE_IN_WORDS* * sizeof(uint32_t))
- #define *CC_AES_IV_SIZE_IN_WORDS* *CC_AES_CRYPTO_BLOCK_SIZE_IN_WORDS*
- #define *CC_AES_IV_SIZE_IN_BYTES* (*CC_AES_IV_SIZE_IN_WORDS* * sizeof(uint32_t))

Typedefs

- typedef uint8_t *CCAesIv_t*[*CC_AES_IV_SIZE_IN_BYTES*]
- typedef uint8_t *CCAesKeyBuffer_t*[*CC_AES_KEY_MAX_SIZE_IN_BYTES*]
- typedef struct *CCAesUserContext_t* *CCAesUserContext_t*

  The context prototype of the user.
- typedef struct *CCAesUserKeyData_t* *CCAesUserKeyData_t*
- typedef struct *CCAesHwKeyData_t* *CCAesHwKeyData_t*

Enumerations

- enum *CCAesEncryptMode_t* { *CC_AES_ENCRYPT* = 0, *CC_AES_DECRYPT* = 1, *CC_AES_NUM_OF_ENCRYPT_MODES*, *CC_AES_ENCRYPT_MODE_LAST* = 0x7FFFFFFF }
- enum *CCAesOperationMode_t* { *CC_AES_MODE_ECB* = 0, *CC_AES_MODE_CBC* = 1, *CC_AES_MODE_CBC_MAC* = 2, *CC_AES_MODE_CTR* = 3, *CC_AES_MODE_XCBC_MAC* = 4, *CC_AES_MODE_CMAC* = 5, *CC_AES_MODE_XTS* = 6, *CC_AES_MODE_CBC_CTS* = 7, *CC_AES_MODE_OFB* = 8, *CC_AES_NUM_OF_OPERATION_MODES*, *CC_AES_OPERATION_MODE_LAST* = 0x7FFFFFFF }
- enum *CCAesPaddingType_t* { *CC_AES_PADDING_NONE* = 0, *CC_AES_PADDING_PKCS7* = 1, *CC_AES_NUM_OF_PADDING_TYPES*, *CC_AES_PADDING_TYPE_LAST* = 0x7FFFFFFF }
- enum *CCAesKeyType_t* { *CC_AES_USER_KEY* = 0, *CC_AES_PLATFORM_KEY* = 1, *CC_AES_CUSTOMER_KEY* = 2, *CC_AES_NUM_OF_KEY_TYPES*, *CC_AES_KEY_TYPE_LAST* = 0x7FFFFFFF }

Detailed description

This file contains the type definitions that are used by the CryptoCell AES APIs.

1.7.5 cc_aes_defs_proj.h File Reference

This file contains definitions that are used for CryptoCell AES APIs.

#include "cc_pal_types.h"

Macros

- #define *CC_AES_USER_CTX_SIZE_IN_WORDS* (4+8+8+4)
- #define *CC_AES_KEY_MAX_SIZE_IN_WORDS* 8
- #define *CC_AES_KEY_MAX_SIZE_IN_BYTES* (*CC_AES_KEY_MAX_SIZE_IN_WORDS* * sizeof(uint32_t))

Detailed description

This file contains definitions that are used for CryptoCell AES APIs.

1.7.6 cc_cmpu.h File Reference

This file contains all of the ICV production library APIs, their enums and definitions.

#include "cc_pal_types_plat.h"

#include "cc_prod.h"

Data structures

- union *CCCmpuUniqueBuff_t*
- The device use of the unique buffer. struct *CCCmpuData_t*

Macros

- #define *CMPU_WORKSPACE_MINIMUM_SIZE* 4096
- #define *PROD_UNIQUE_BUFF_SIZE* 16

Enumerations

- enum *CCCmpuUniqueDataType_t* { *CMPU_UNIQUE_IS_HBK0* = 1, *CMPU_UNIQUE_IS_USER_DATA* = 2, *CMPU_UNIQUE_RESERVED* = 0x7FFFFFFF }

Functions

- CIMPORT_C CCError_t *CCProd_Cmpu* (unsigned long ccHwRegBaseAddr, *CCCmpuData_t* *pCmpuData, unsigned long workspaceBaseAddr, uint32_t workspaceSize)

  This function burns all ICV assets into the OTP of the device.

Detailed description

This file contains all of the ICV production library APIs, their enums and definitions.

1.7.7 cc_dmpu.h File Reference

This file contains all of the OEM production library APIs, their enums and definitions.

#include "cc_pal_types_plat.h"

#include "cc_prod.h"

Data structures

- union *CCDmpuHbkBuff_t*
- The device use of the Hbk buffer. struct *CCDmpuData_t*

Macros

- #define *DMPU_WORKSPACE_MINIMUM_SIZE* 1536
- #define *DMPU_HBK1_SIZE_IN_WORDS* 4
- #define *DMPU_HBK_SIZE_IN_WORDS* 8

Enumerations

- enum *CCDmpuHBKType_t* { *DMPU_HBK_TYPE_HBK1* = 1, *DMPU_HBK_TYPE_HBK* = 2, *DMPU_HBK_TYPE_RESERVED* = 0x7FFFFFFF }

Functions

- CIMPORT_C CCError_t *CCProd_Dmpu* (unsigned long ccHwRegBaseAddr, *CCDmpuData_t* *pDmpuData, unsigned long workspaceBaseAddr, uint32_t workspaceSize)

  This function burns all OEM assets into the OTP of the device.

Detailed description

This file contains all of the OEM production library APIs, their enums and definitions.

1.7.8 cc_ecpki_domains_defs.h File Reference

This file contains CryptoCell ECPKI domains supported by the project.

#include "cc_ecpki_domain_secp192r1.h"

#include "cc_ecpki_domain_secp224r1.h"

#include "cc_ecpki_domain_secp256r1.h"

#include "cc_ecpki_domain_secp521r1.h"

#include "cc_ecpki_domain_secp192k1.h"

#include "cc_ecpki_domain_secp224k1.h"

#include "cc_ecpki_domain_secp256k1.h"

#include "cc_ecpki_domain_secp384r1.h"

Typedefs

- typedef const *CCEcpkiDomain_t* *(* *getDomainFuncP*) (void)

Detailed description

This file contains CryptoCell ECPKI domains supported by the project.

1.7.9 cc_ecpki_types.h File Reference

This file contains all the type definitions that are used for the CryptoCell ECPKI APIs.

#include "cc_bitops.h"

#include "cc_pal_types_plat.h"

#include "cc_hash_defs.h"

#include "cc_pka_defs_hw.h"

#include "cc_pal_compiler.h"

#include "mbedtls/md.h"

Data structures

- struct *CCEcpkiDomain_t*
- The structure containing the EC domain parameters in little-endian form. struct *CCEcpkiPointAffine_t*
- struct *CCEcpkiPublKey_t*
- struct *CCEcpkiUserPublKey_t*
- The user structure prototype of the EC public key. struct *CCEcpkiPrivKey_t*
- struct *CCEcpkiUserPrivKey_t*
- The user structure prototype of the EC private key. struct *CCEcdhTempData_t*
- struct *CCEcpkiBuildTempData_t*
- struct *EcdsaSignContext_t*
- struct *CCEcdsaSignUserContext_t*
- The context definition of the user for the signing operation. struct *EcdsaVerifyContext_t*
- struct *CCEcdsaVerifyUserContext_t*
- The context definition of the user for the verification operation. struct *CCEcpkiKgTempData_t*
- struct *CCEciesTempData_t*
- struct *CCEcpkiKgFipsContext_t*
- struct *CCEcdsaFipsKatContext_t*
- struct *CCEcdhFipsKatContext_t*

Macros

- #define *CC_PKA_DOMAIN_LLF_BUFF_SIZE_IN_WORDS* (10 + 3**CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS*)
- #define *CC_ECPKI_FIPS_ORDER_LENGTH* (256/*CC_BITS_IN_BYTE*)

Typedefs

- typedef struct *CCEcpkiUserPublKey_t* *CCEcpkiUserPublKey_t*

  The user structure prototype of the EC public key.
- typedef struct *CCEcpkiUserPrivKey_t* *CCEcpkiUserPrivKey_t*

  The user structure prototype of the EC private key.

- typedef struct *CCEcdhTempData_t* *CCEcdhTempData_t*
- typedef struct *CCEcpkiBuildTempData_t* *CCEcpkiBuildTempData_t*
- typedef uint32_t *CCEcdsaSignIntBuff_t*[*CC_PKA_ECDSA_SIGN_BUFF_MAX_LENGTH_IN_WORDS*]
- typedef struct *CCEcdsaSignUserContext_t* *CCEcdsaSignUserContext_t*

  The context definition of the user for the signing operation.
- typedef uint32_t *CCEcdsaVerifyIntBuff_t*[*CC_PKA_ECDSA_VERIFY_BUFF_MAX_LENGTH_IN_WORDS*]
- typedef struct *CCEcdsaVerifyUserContext_t* *CCEcdsaVerifyUserContext_t*

  The context definition of the user for the verification operation.
- typedef struct *CCEcpkiKgTempData_t* *CCEcpkiKgTempData_t*
- typedef struct *CCEciesTempData_t* *CCEciesTempData_t*
- typedef struct *CCEcpkiKgFipsContext_t* *CCEcpkiKgFipsContext_t*
- typedef struct *CCEcdsaFipsKatContext_t* *CCEcdsaFipsKatContext_t*
- typedef struct *CCEcdhFipsKatContext_t* *CCEcdhFipsKatContext_t*

Enumerations

- enum *CCEcpkiDomainID_t* { *CC_ECPKI_DomainID_secp192k1*, *CC_ECPKI_DomainID_secp192r1*, *CC_ECPKI_DomainID_secp224k1*, *CC_ECPKI_DomainID_secp224r1*, *CC_ECPKI_DomainID_secp256k1*, *CC_ECPKI_DomainID_secp256r1*, *CC_ECPKI_DomainID_secp384r1*, *CC_ECPKI_DomainID_secp521r1*, *CC_ECPKI_DomainID_Builded*, *CC_ECPKI_DomainID_OffMode*, *CC_ECPKI_DomainIDLast* = 0x7FFFFFFF }EC domain idetifiers.
- enum *CCEcpkiHashOpMode_t* { *CC_ECPKI_HASH_SHA1_mode* = 0, *CC_ECPKI_HASH_SHA224_mode* = 1, *CC_ECPKI_HASH_SHA256_mode* = 2, *CC_ECPKI_HASH_SHA384_mode* = 3, *CC_ECPKI_HASH_SHA512_mode* = 4, *CC_ECPKI_AFTER_HASH_SHA1_mode* = 5, *CC_ECPKI_AFTER_HASH_SHA224_mode* = 6, *CC_ECPKI_AFTER_HASH_SHA256_mode* = 7, *CC_ECPKI_AFTER_HASH_SHA384_mode* = 8, *CC_ECPKI_AFTER_HASH_SHA512_mode* = 9, *CC_ECPKI_HASH_NumOfModes*, *CC_ECPKI_HASH_OpModeLast* = 0x7FFFFFFF }Hash operation mode.
- enum *CCEcpkiPointCompression_t* { *CC_EC_PointCompressed* = 2, *CC_EC_PointUncompressed* = 4, *CC_EC_PointContWrong* = 5, *CC_EC_PointHybrid* = 6, *CC_EC_PointCompresOffMode* = 8, *CC_ECPKI_PointCompressionLast* = 0x7FFFFFFF }
- enum *ECPublKeyCheckMode_t* { *CheckPointersAndSizesOnly* = 0, *ECpublKeyPartlyCheck* = 1, *ECpublKeyFullCheck* = 2, PublKeyChecingOffMode, *EC_PublKeyCheckModeLast* = 0x7FFFFFFF }
- enum *CCEcpkiScaProtection_t* { SCAP_Inactive, *SCAP_Active*, *SCAP_OFF_MODE*, *SCAP_LAST* = 0x7FFFFFFF }

Detailed description

This file contains all the type definitions that are used for the CryptoCell ECPKI APIs.

1.7.10 cc_error.h File Reference

This file defines the error return code types and the numbering spaces for each module of the layers listed.

#include "cc_pal_types.h"

Macros

- #define *CC_ERROR_BASE* 0x00F00000UL
- #define *CC_ERROR_LAYER_RANGE* 0x00010000UL
- #define *CC_ERROR_MODULE_RANGE* 0x00000100UL
- #define *CC_LAYER_ERROR_IDX* 0x00UL
- #define *LLF_LAYER_ERROR_IDX* 0x01UL
- #define *GENERIC_ERROR_IDX* 0x05UL
- #define *AES_ERROR_IDX* 0x00UL
- #define *DES_ERROR_IDX* 0x01UL
- #define *HASH_ERROR_IDX* 0x02UL
- #define *HMAC_ERROR_IDX* 0x03UL
- #define *RSA_ERROR_IDX* 0x04UL
- #define *DH_ERROR_IDX* 0x05UL
- #define *ECPKI_ERROR_IDX* 0x08UL
- #define *RND_ERROR_IDX* 0x0CUL
- #define *COMMON_ERROR_IDX* 0x0DUL
- #define *KDF_ERROR_IDX* 0x11UL
- #define *HKDF_ERROR_IDX* 0x12UL
- #define *AESCCM_ERROR_IDX* 0x15UL
- #define *FIPS_ERROR_IDX* 0x17UL
- #define *PKA_MODULE_ERROR_IDX* 0x21UL
- #define *CHACHA_ERROR_IDX* 0x22UL
- #define *EC_MONT_EDW_ERROR_IDX* 0x23UL
- #define *CHACHA_POLY_ERROR_IDX* 0x24UL
- #define *POLY_ERROR_IDX* 0x25UL
- #define *SRP_ERROR_IDX* 0x26UL
- #define *AESGCM_ERROR_IDX* 0x27UL
- #define *AES_KEYWRAP_ERROR_IDX* 0x28UL
- #define *MNG_ERROR_IDX* 0x29UL
- #define *PROD_ERROR_IDX* 0x2AUL
- #define *FFCDH_ERROR_IDX* 0x2BUL
- #define *FFC_DOMAIN_ERROR_IDX* 0x2CUL
- #define *EXT_DMA_ERROR_IDX* 0x2DUL
- #define *CC_AES_MODULE_ERROR_BASE*
- #define *CC_DES_MODULE_ERROR_BASE*
- #define *CC_HASH_MODULE_ERROR_BASE*

- #define CC_HMAC_MODULE_ERROR_BASE
- #define CC_RSA_MODULE_ERROR_BASE
- #define CC_DH_MODULE_ERROR_BASE
- #define CC_ECPKI_MODULE_ERROR_BASE
- #define LLF_ECPKI_MODULE_ERROR_BASE
- #define CC_RND_MODULE_ERROR_BASE
- #define LLF_RND_MODULE_ERROR_BASE
- #define CC_COMMON_MODULE_ERROR_BASE
- #define CC_KDF_MODULE_ERROR_BASE
- #define CC_HKDF_MODULE_ERROR_BASE
- #define CC_AESCCM_MODULE_ERROR_BASE
- #define CC_FIPS_MODULE_ERROR_BASE
- #define PKA_MODULE_ERROR_BASE
- #define CC_CHACHA_MODULE_ERROR_BASE
- #define CC_EC_MONT_EDW_MODULE_ERROR_BASE
- #define CC_CHACHA_POLY_MODULE_ERROR_BASE
- #define CC_POLY_MODULE_ERROR_BASE
- #define CC_SRP_MODULE_ERROR_BASE
- #define CC_AESGCM_MODULE_ERROR_BASE
- #define CC_AES_KEYWRAP_MODULE_ERROR_BASE
- #define CC_MNG_MODULE_ERROR_BASE
- #define CC_PROD_MODULE_ERROR_BASE
- #define CC_FFCDH_MODULE_ERROR_BASE
- #define CC_FFC_DOMAIN_MODULE_ERROR_BASE
- #define CC_EXT_DMA_MODULE_ERROR_BASE
- #define GENERIC_ERROR_BASE (CC_ERROR_BASE + (CC_ERROR_LAYER_RANGE * GENERIC_ERROR_IDX))
- #define CC_FATAL_ERROR (GENERIC_ERROR_BASE + 0x00UL)
- #define CC_OUT_OF_RESOURCE_ERROR (GENERIC_ERROR_BASE + 0x01UL)
- #define CC_ILLEGAL_RESOURCE_VAL_ERROR (GENERIC_ERROR_BASE + 0x02UL)
- #define CC_CRYPTO_RETURN_ERROR(retCode, retcodeInfo, funcHandler) ((retCode) == 0 ? CC_OK : funcHandler(retCode, retcodeInfo))

Detailed description

This file defines the error return code types and the numbering spaces for each module of the layers listed.

1.7.11 cc_general_defs.h File Reference

This file contains general definitions of the CryptoCell runtime SW APIs.

#include "cc_hash_defs.h"

Data structures

- struct *HmacHash_t*

Macros

- #define *CC_HASH_NAME_MAX_SIZE* 10
- #define *CC_AES_KDR_MAX_SIZE_BYTES* 32
- #define *CC_AES_KDR_MAX_SIZE_WORDS* (*CC_AES_KDR_MAX_SIZE_BYTES*/sizeof(uint32_t))
- #define *CC_LCS_CHIP_MANUFACTURE_LCS* 0x0
- #define *CC_LCS_SECURE_LCS* 0x5

Variables

- const *HmacHash_t* *HmacHashInfo_t* [*CC_HASH_NumOfModes*]
- const uint8_t *HmacSupportedHashModes_t* [*CC_HASH_NumOfModes*]
- const char *HashAlgMode2mbedtlsString* [*CC_HASH_NumOfModes*][*CC_HASH_NAME_MAX_SIZE*]

Detailed description

This file contains general definitions of the CryptoCell runtime SW APIs.

1.7.12 cc_hash_defs.h File Reference

This file contains definitions of the CryptoCell hash APIs.

#include "cc_pal_types.h"

#include "cc_error.h"

#include "cc_hash_defs_proj.h"

Data structures

- struct *CCHashUserContext_t*

The context prototype of the user. Macros

- #define *CC_HASH_RESULT_SIZE_IN_WORDS* 16
- #define *CC_HASH_MD5_DIGEST_SIZE_IN_BYTES* 16
- #define *CC_HASH_MD5_DIGEST_SIZE_IN_WORDS* 4
- #define *CC_HASH_SHA1_DIGEST_SIZE_IN_BYTES* 20
- #define *CC_HASH_SHA1_DIGEST_SIZE_IN_WORDS* 5
- #define *CC_HASH_SHA224_DIGEST_SIZE_IN_WORDS* 7
- #define *CC_HASH_SHA256_DIGEST_SIZE_IN_WORDS* 8
- #define *CC_HASH_SHA384_DIGEST_SIZE_IN_WORDS* 12
- #define *CC_HASH_SHA512_DIGEST_SIZE_IN_WORDS* 16
- #define *CC_HASH_SHA224_DIGEST_SIZE_IN_BYTES* 28
- #define *CC_HASH_SHA256_DIGEST_SIZE_IN_BYTES* 32
- #define *CC_HASH_SHA384_DIGEST_SIZE_IN_BYTES* 48
- #define *CC_HASH_SHA512_DIGEST_SIZE_IN_BYTES* 64
- #define *CC_HASH_BLOCK_SIZE_IN_WORDS* 16
- #define *CC_HASH_BLOCK_SIZE_IN_BYTES* 64
- #define *CC_HASH_SHA512_BLOCK_SIZE_IN_WORDS* 32
- #define *CC_HASH_SHA512_BLOCK_SIZE_IN_BYTES* 128
- #define *CC_HASH_UPDATE_DATA_MAX_SIZE_IN_BYTES* (1 << 29)

Typedefs

- typedef uint32_t *CCHashResultBuf_t*[*CC_HASH_RESULT_SIZE_IN_WORDS*]
- typedef struct *CCHashUserContext_t* *CCHashUserContext_t*

  The context prototype of the user.

Enumerations

- enum *CCHashOperationMode_t* { *CC_HASH_SHA1_mode* = 0, *CC_HASH_SHA224_mode* = 1, *CC_HASH_SHA256_mode* = 2, *CC_HASH_SHA384_mode* = 3, *CC_HASH_SHA512_mode* = 4, *CC_HASH_MD5_mode* = 5, *CC_HASH_NumOfModes*, *CC_HASH_OperationModeLast* = 0x7FFFFFFF }

Detailed description

This file contains definitions of the CryptoCell hash APIs.

1.7.13 cc_hash_defs_proj.h File Reference

This file contains the project-specific definitions of hash APIs.

Macros

- #define *CC_HASH_USER_CTX_SIZE_IN_WORDS* 60

Detailed description

This file contains the project-specific definitions of hash APIs.

1.7.14 cc_lib.h File Reference

This file contains all of the CryptoCell library basic APIs, their enums and definitions.

#include "cc_pal_types.h"

#include "cc_rnd_common.h"

Macros

- #define *DX_VERSION_PRODUCT_BIT_SHIFT* 0x18UL
- #define *DX_VERSION_PRODUCT_BIT_SIZE* 0x8UL

Enumerations

- enum *CClibRetCode_t* { *CC_LIB_RET_OK* = 0, *CC_LIB_RET_EINVAL_CTX_PTR*, *CC_LIB_RET_EINVAL_WORK_BUF_PTR*, *CC_LIB_RET_HAL*, *CC_LIB_RET_PAL*, *CC_LIB_RET_RND_INST_ERR*, *CC_LIB_RET_EINVAL_PIDR*, *CC_LIB_RET_EINVAL_CIDR*, *CC_LIB_AO_WRITE_FAILED_ERR*, *CC_LIB_RESERVE32B* = 0x7FFFFFFFL }

Functions

- *CClibRetCode_t* *CC_LibInit* (*CCRndContext_t* *rndContext_ptr, *CCRndWorkBuff_t* *rndWorkBuff_ptr)

  This function performs global initialization of the CryptoCell runtime library.

- *CClibRetCode_t* *CC_LibFini* (*CCRndContext_t* *rndContext_ptr)

  This function finalizes library operations.

Detailed description

This file contains all of the CryptoCell library basic APIs, their enums and definitions.

1.7.15 cc_pal_abort.h File Reference

This file includes all PAL APIs.

#include "cc_pal_abort_plat.h"

Functions

- void *CC_PalAbort* (const char *exp)

  This function performs the "Abort" operation.

Detailed description

This file includes all PAL APIs.

1.7.16 cc_pal_barrier.h File Reference

This file contains the definitions and APIs for memory-barrier implementation.

Functions

- void *CC_PalWmb* (void)
- void *CC_PalRmb* (void)

Detailed description

This file contains the definitions and APIs for memory-barrier implementation.

This is a placeholder for platform-specific memory barrier implementation. The secure core driver should include a memory barrier, before and after the last word of the descriptor, to allow correct order between the words and different descriptors.

1.7.17 cc_pal_compiler.h File Reference

This file contains CryptoCell PAL platform-dependent compiler-related definitions.

Macros

- #define *CC_PAL_COMPILER_SECTION*(sectionName) __attribute__((section(sectionName)))
- #define *CC_PAL_COMPILER_KEEP_SYMBOL* __attribute__((used))
- #define *CC_PAL_COMPILER_ALIGN*(alignement) __attribute__((aligned(alignement)))
- #define *CC_PAL_COMPILER_FUNC_NEVER_RETURNS* __attribute__((noreturn))
- #define *CC_PAL_COMPILER_FUNC_DONT_INLINE* __attribute__((noinline))
- #define *CC_PAL_COMPILER_TYPE_MAY_ALIAS* __attribute__((__may_alias__))
- #define *CC_PAL_COMPILER_SIZEOF_STRUCT_MEMBER*(type_name, member_name) sizeof(((type_name *)0)->member_name)
- #define *CC_ASSERT_CONCAT_*(a, b) a##b
- #define *CC_ASSERT_CONCAT*(a, b) *CC_ASSERT_CONCAT_*(a, b)
- #define *CC_PAL_COMPILER_ASSERT*(cond, message) enum { *CC_ASSERT_CONCAT*(assert_line_, __LINE__) = 1/(!!(cond)) }

Detailed description

This file contains CryptoCell PAL platform-dependent compiler-related definitions.

1.7.18 cc_pal_error.h File Reference

This file contains the error definitions of the platform-dependent PAL APIs.

Macros

- #define *CC_PAL_BASE_ERROR* 0x0F000000
- #define *CC_PAL_MEM_BUF1_GREATER* *CC_PAL_BASE_ERROR* + 0x01UL
- #define *CC_PAL_MEM_BUF2_GREATER* *CC_PAL_BASE_ERROR* + 0x02UL
- #define *CC_PAL_SEM_CREATE_FAILED* *CC_PAL_BASE_ERROR* + 0x03UL
- #define *CC_PAL_SEM_DELETE_FAILED* *CC_PAL_BASE_ERROR* + 0x04UL
- #define *CC_PAL_SEM_WAIT_TIMEOUT* *CC_PAL_BASE_ERROR* + 0x05UL
- #define *CC_PAL_SEM_WAIT_FAILED* *CC_PAL_BASE_ERROR* + 0x06UL
- #define *CC_PAL_SEM_RELEASE_FAILED* *CC_PAL_BASE_ERROR* + 0x07UL
- #define *CC_PAL_ILLEGAL_ADDRESS* *CC_PAL_BASE_ERROR* + 0x08UL

Detailed description

This file contains the error definitions of the platform-dependent PAL APIs.

1.7.19 cc_pal_init.h File Reference

This file contains the PAL layer entry point.

#include "cc_pal_types.h"

Functions

- int *CC_PalInit* (void)

  This function performs all initializations that may be required by your PAL implementation, specifically by the DMA-able buffer scheme.

- void *CC_PalTerminate* (void)

  This function terminates the PAL implementation and frees the resources that were allocated by *CC_PalInit*.

Detailed description

This file contains the PAL layer entry point.

It includes the definitions and APIs for PAL initialization and termination.

1.7.20 cc_pal_log.h File Reference

This file contains the PAL layer log definitions. The log is disabled by default.

#include "cc_pal_types.h"

#include "cc_pal_log_plat.h"

Macros

- #define *CC_PAL_LOG_LEVEL_NULL* (-1)
- #define *CC_PAL_LOG_LEVEL_ERR* 0
- #define *CC_PAL_LOG_LEVEL_WARN* 1
- #define *CC_PAL_LOG_LEVEL_INFO* 2
- #define *CC_PAL_LOG_LEVEL_DEBUG* 3
- #define *CC_PAL_LOG_LEVEL_TRACE* 4
- #define *CC_PAL_LOG_LEVEL_DATA* 5
- #define *CC_PAL_LOG_CUR_COMPONENT* 0xFFFFFFFF
- #define *CC_PAL_LOG_CUR_COMPONENT_NAME* "CC"
- #define *CC_PAL_MAX_LOG_LEVEL* *CC_PAL_LOG_LEVEL_ERR* /**CC_PAL_LOG_LEVEL_DEBUG**/
- #define *__CC_PAL_LOG_LEVEL_EVAL*(level) level
- #define *_CC_PAL_MAX_LOG_LEVEL* *__CC_PAL_LOG_LEVEL_EVAL*(*CC_PAL_MAX_LOG_LEVEL*)
- #define inline __inline
- #define *CC_PalLogInit*() do {} while (0)
- #define *CC_PalLogLevelSet*(setLevel) do {} while (0)
- #define *CC_PalLogMaskSet*(setMask) do {} while (0)
- #define *_CC_PAL_LOG*(level, format, ...)
- #define *CC_PAL_LOG_ERR*(format, ...) *_CC_PAL_LOG*(ERR, format, ##__VA_ARGS__)
- #define *CC_PAL_LOG_WARN*(...) do {} while (0)
- #define *CC_PAL_LOG_INFO*(...) do {} while (0)
- #define *CC_PAL_LOG_DEBUG*(...) do {} while (0)
- #define *CC_PAL_LOG_DUMP_BUF*(msg, buf, size) do {} while (0)
- #define *CC_PAL_LOG_TRACE*(...) do {} while (0)
- #define *CC_PAL_LOG_DATA*(...) do {} while (0)

Detailed description

This file contains the PAL layer log definitions. The log is disabled by default.

1.7.21 cc_pal_mem.h File Reference

This file contains functions for memory operations.

#include "cc_pal_types.h"

#include "cc_pal_mem_plat.h"

#include "cc_pal_malloc_plat.h"

#include <stdlib.h>

#include <string.h>

Macros

- #define *CC_PalMemCmp*(aTarget, aSource, aSize) CC_PalMemCmpPlat(aTarget, aSource, aSize)

  This function compares between two given buffers, according to the given size.

- #define *CC_PalMemCopy*(aDestination, aSource, aSize) CC_PalMemCopyPlat(aDestination, aSource, aSize)

  This function copies aSize Bytes from the source buffer to the destination buffer.

- #define *CC_PalMemMove*(aDestination, aSource, aSize) CC_PalMemMovePlat(aDestination, aSource, aSize)

  This function moves aSize Bytes from the source buffer to the destination buffer. This function supports overlapped buffers.

- #define *CC_PalMemSet*(aTarget, aChar, aSize) CC_PalMemSetPlat(aTarget, aChar, aSize)

  This function sets aSize Bytes of aChar in the given buffer.

- #define *CC_PalMemSetZero*(aTarget, aSize) CC_PalMemSetZeroPlat(aTarget, aSize)

  This function sets aSize Bytes in the given buffer with zeroes.

- #define *CC_PalMemMalloc*(aSize) CC_PalMemMallocPlat(aSize)

  This function allocates a memory buffer according to aSize .

- #define *CC_PalMemRealloc*(aBuffer, aNewSize) CC_PalMemReallocPlat(aBuffer, aNewSize)

  This function reallocates a memory buffer according to aNewSize . The contents of the old buffer is moved to the new location.

- #define *CC_PalMemFree*(aBuffer) CC_PalMemFreePlat(aBuffer)

  This function frees a previously-allocated buffer.

Detailed description

This file contains functions for memory operations.

The functions are generally implemented as wrappers to different operating-system calls.

Note

None of the described functions validate the input parameters, so that the behavior of the APIs in case of an illegal parameter is dependent on the behavior of the operating system.

1.7.22 cc_pal_memmap.h File Reference

This file contains functions for memory mapping.

#include "cc_pal_types.h"

#include "cc_address_defs.h"

Functions

- uint32_t *CC_PalMemMap* (*CCDmaAddr_t* physicalAddress, uint32_t mapSize, uint32_t **ppVirtBuffAddr)

  This function returns the base virtual address that maps the base physical address.
- uint32_t *CC_PalMemUnMap* (uint32_t *pVirtBuffAddr, uint32_t mapSize)

  This function unmaps a specified address range that was previously mapped by *CC_PalMemMap*.

Detailed description

This file contains functions for memory mapping.

Note

None of the described functions validate the input parameters, so that the behavior of the APIs in case of an illegal parameter is dependent on the behavior of the operating system.

1.7.23 cc_pal_mutex.h File Reference

This file contains functions for resource management (mutex operations).

#include "cc_pal_mutex_plat.h"

#include "cc_pal_types_plat.h"

Functions

- CCError_t *CC_PalMutexCreate* (CC_PalMutex *pMutexId)

  This function creates a mutex.
- CCError_t *CC_PalMutexDestroy* (CC_PalMutex *pMutexId)

  This function destroys a mutex.
- CCError_t *CC_PalMutexLock* (CC_PalMutex *pMutexId, uint32_t aTimeOut)

  This function waits for a mutex with aTimeOut . aTimeOut is specified in milliseconds. A value of aTimeOut=CC_INFINITE means that the function will not return.
- CCError_t *CC_PalMutexUnlock* (CC_PalMutex *pMutexId)

  This function releases the mutex.

Detailed description

This file contains functions for resource management (mutex operations).

These functions are generally implemented as wrappers to different operating-system calls.

Note

None of the described functions validate the input parameters, so that the behavior of the APIs in case of an illegal parameter is dependent on the behavior of the operating system.

1.7.24 cc_pal_pm.h File Reference

This file contains the definitions and APIs for power-management implementation.

Functions

- void *CC_PalPowerSaveModeInit* (void)

  This function initiates an atomic counter.
- int32_t *CC_PalPowerSaveModeStatus* (void)

  This function returns the number of active registered CryptoCell operations.
- CCError_t *CC_PalPowerSaveModeSelect* (*CCBool* isPowerSaveMode)

  This function updates the atomic counter on each call to CryptoCell.

Detailed description

This file contains the definitions and APIs for power-management implementation.

This is a placeholder for platform-specific power management implementation. The module should be updated whether CryptoCell is active or not, to notify the external PMU when it might be powered down.

1.7.25 cc_pal_sb_plat.h File Reference

This file contains platform-dependent definitions used in the Boot Services code.

#include "cc_pal_types.h"

Typedefs

- typedef uint32_t *CCDmaAddr_t*
- typedef uint32_t *CCAddr_t*

Detailed description

This file contains platform-dependent definitions used in the Boot Services code.

1.7.26 cc_pal_trng.h File Reference

This file contains APIs for retrieving TRNG user parameters.

#include "cc_pal_types.h"

Data structures

- struct *CC_PalTrngParams_t*

Typedefs

- typedef struct *CC_PalTrngParams_t* *CC_PalTrngParams_t*

Functions

- CCError_t *CC_PalTrngParamGet* (*CC_PalTrngParams_t* *pTrngParams, size_t *pParamsSize)

  This function returns the TRNG user parameters.

Detailed description

This file contains APIs for retrieving TRNG user parameters.

1.7.27 cc_pal_types.h File Reference

This file contains definitions and types of CryptoCell PAL platform-dependent APIs.

#include "cc_pal_types_plat.h"

Macros

- #define *CC_SUCCESS* 0UL
- #define *CC_FAIL* 1UL
- #define *CC_OK* 0
- #define *CC_UNUSED_PARAM*(prm) ((void)prm)
- #define *CC_MAX_UINT32_VAL* (0xFFFFFFFF)
- #define *CC_MIN*(a, b) min(a , b)
- #define *CC_MAX*(a, b) max(a , b)
- #define *CALC_FULL_BYTES*(numBits) ((numBits)/*CC_BITS_IN_BYTE* + (((numBits) & (*CC_BITS_IN_BYTE*-1)) > 0))
- #define *CALC_FULL_32BIT_WORDS*(numBits) ((numBits)/*CC_BITS_IN_32BIT_WORD* + (((numBits) & (*CC_BITS_IN_32BIT_WORD*-1)) > 0))
- #define *CALC_32BIT_WORDS_FROM_BYTES*(sizeBytes) ((sizeBytes)/*CC_32BIT_WORD_SIZE* + (((sizeBytes) & (*CC_32BIT_WORD_SIZE*-1)) > 0))
- #define *CALC_32BIT_WORDS_FROM_64BIT_DWORD*(sizeWords) (sizeWords * *CC_32BIT_WORD_IN_64BIT_DWORD*)
- #define *ROUNDUP_BITS_TO_32BIT_WORD*(numBits) (*CALC_FULL_32BIT_WORDS*(numBits) * *CC_BITS_IN_32BIT_WORD*)
- #define *ROUNDUP_BITS_TO_BYTES*(numBits) (*CALC_FULL_BYTES*(numBits) * *CC_BITS_IN_BYTE*)
- #define *ROUNDUP_BYTES_TO_32BIT_WORD*(sizeBytes) (*CALC_32BIT_WORDS_FROM_BYTES*(sizeBytes) * *CC_32BIT_WORD_SIZE*)
- #define *CC_1K_SIZE_IN_BYTES* 1024
- #define *CC_BITS_IN_BYTE* 8
- #define *CC_BITS_IN_32BIT_WORD* 32
- #define *CC_32BIT_WORD_SIZE* 4
- #define *CC_32BIT_WORD_IN_64BIT_DWORD* 2

Enumerations

- enum *CCBool* { *CC_FALSE* = 0, *CC_TRUE* = 1 }

Detailed description

This file contains definitions and types of CryptoCell PAL platform-dependent APIs.

1.7.28 cc_pka_defs_hw.h File Reference

This file contains all of the enums and definitions that are used in PKA APIs.

#include "cc_pal_types.h"

#include "cc_pka_hw_plat_defs.h"

Macros

- #define CC_RSA_MAXIMUM_MOD_BUFFER_SIZE_IN_WORDS ((CC_RSA_MAX_VALID_KEY_SIZE_VALUE_IN_BITS + CC_PKA_WORD_SIZE_IN_BITS) / CC_BITS_IN_32BIT_WORD)
- #define CC_ECPKI_MODUL_MAX_LENGTH_IN_BITS 521
- #define CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS 5
- #define CC_PKA_ECPKI_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS
- #define CC_PKA_MAXIMUM_MOD_BUFFER_SIZE_IN_WORDS CC_RSA_MAXIMUM_MOD_BUFFER_SIZE_IN_WORDS
- #define CC_PKA_PUB_KEY_BUFF_SIZE_IN_WORDS (2*CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS)
- #define CC_PKA_PRIV_KEY_BUFF_SIZE_IN_WORDS (2*CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS)
- #define CC_PKA_KGDATA_BUFF_SIZE_IN_WORDS (3*CC_PKA_MAXIMUM_MOD_BUFFER_SIZE_IN_WORDS + 3*CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS)
- #define CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS 18
- #define CC_ECPKI_ORDER_MAX_LENGTH_IN_WORDS (CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS + 1)
- #define CC_PKA_DOMAIN_BUFF_SIZE_IN_WORDS (2*CC_PKA_BARRETT_MOD_TAG_BUFF_SIZE_IN_WORDS)
- #define COUNT_NAF_WORDS_PER_KEY_WORD 8
- #define CC_PKA_ECDSA_NAF_BUFF_MAX_LENGTH_IN_WORDS (COUNT_NAF_WORDS_PER_KEY_WORD*CC_ECPKI_ORDER_MAX_LENGTH_IN_WORDS + 1)
- #define CC_PKA_ECPKI_SCALAR_MUL_BUFF_MAX_LENGTH_IN_WORDS (CC_PKA_ECDSA_NAF_BUFF_MAX_LENGTH_IN_WORDS+CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS+2)
- #define CC_PKA_ECPKI_BUILD_TMP_BUFF_MAX_LENGTH_IN_WORDS (3*CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS+CC_PKA_ECPKI_SCALAR_MUL_BUFF_MAX_LENGTH_IN_WORDS)

- #define CC_PKA_ECDSA_SIGN_BUFF_MAX_LENGTH_IN_WORDS (6*CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS+CC_PKA_ECPKI_SCALAR_MUL_BUFF_MAX_LENGTH_IN_WORDS)
- #define CC_PKA_ECDH_BUFF_MAX_LENGTH_IN_WORDS (2*CC_ECPKI_ORDER_MAX_LENGTH_IN_WORDS + CC_PKA_ECPKI_SCALAR_MUL_BUFF_MAX_LENGTH_IN_WORDS)
- #define CC_PKA_KG_BUFF_MAX_LENGTH_IN_WORDS (2*CC_ECPKI_ORDER_MAX_LENGTH_IN_WORDS + CC_PKA_ECPKI_SCALAR_MUL_BUFF_MAX_LENGTH_IN_WORDS)
- #define CC_PKA_ECDSA_VERIFY_BUFF_MAX_LENGTH_IN_WORDS (3*CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS)
- #define CC_EC_MONT_EDW_MODULUS_MAX_SIZE_IN_BYTES 32U
- #define CC_EC_MONT_EDW_MODULUS_MAX_SIZE_IN_WORDS 8U
- #define CC_EC_MONT_TEMP_BUFF_SIZE_IN_32BIT_WORDS (8 * CC_EC_MONT_EDW_MODULUS_MAX_SIZE_IN_WORDS)
- #define CC_EC_EDW_TEMP_BUFF_SIZE_IN_32BIT_WORDS (8*CC_EC_MONT_EDW_MODULUS_MAX_SIZE_IN_WORDS + (sizeof(CCHashUserContext_t)+CC_32BIT_WORD_SIZE-1)/CC_32BIT_WORD_SIZE)

Detailed description

This file contains all of the enums and definitions that are used in PKA APIs.

1.7.29 cc_pka_hw_plat_defs.h File Reference

This file contains the platform-dependent definitions of the CryptoCell PKA APIs.

#include "cc_pal_types.h"

Macros

- #define *CC_PKA_WORD_SIZE_IN_BITS* 64
- #define *CC_SRP_MAX_MODULUS_SIZE_IN_BITS* 3072
- #define *CC_RSA_MAX_VALID_KEY_SIZE_VALUE_IN_BITS* 4096
- #define *CC_RSA_MAX_KEY_GENERATION_HW_SIZE_BITS* 3072
- #define *CC_RSA_MAX_VALID_KEY_SIZE_VALUE_IN_WORDS* *CC_RSA_MAX_VALID_KEY_SIZE_VALUE_IN_BITS* / *CC_BITS_IN_32BIT_WORD*
- #define *SB_CERT_RSA_KEY_SIZE_IN_BITS* 3072UL
- #define *SB_CERT_RSA_KEY_SIZE_IN_BYTES* (*SB_CERT_RSA_KEY_SIZE_IN_BITS*/*CC_BITS_IN_BYTE*)
- #define *SB_CERT_RSA_KEY_SIZE_IN_WORDS* (*SB_CERT_RSA_KEY_SIZE_IN_BITS*/*CC_BITS_IN_32BIT_WORD*)
- #define *PKA_EXTRA_BITS* 8
- #define *PKA_MAX_COUNT_OF_PHYS_MEM_REGS* 32

Detailed description

This file contains the platform-dependent definitions of the CryptoCell PKA APIs.

1.7.30 cc_prod.h File Reference

This file contains all of the enums and definitions that are used for the ICV and OEM production libraries.

Data structures

- union *CCAssetBuff_t*

The asset buffer. Macros

- #define *CC_PROD_32BIT_WORD_SIZE* sizeof(uint32_t)
- #define *PROD_ASSET_SIZE* 16
- #define *PROD_ASSET_PKG_SIZE* 64
- #define *PROD_ASSET_PKG_WORD_SIZE* (*PROD_ASSET_PKG_SIZE*/*CC_PROD_32BIT_WORD_SIZE*)
- #define *PROD_DCU_LOCK_WORD_SIZE* 4

Typedefs

- typedef uint8_t *CCPlainAsset_t*[*PROD_ASSET_SIZE*]
- typedef uint32_t *CCAssetPkg_t*[*PROD_ASSET_PKG_WORD_SIZE*]

Enumerations

- enum *CCAssetType_t* { *ASSET_NO_KEY* = 0, *ASSET_PLAIN_KEY* = 1, *ASSET_PKG_KEY* = 2, *ASSET_TYPE_RESERVED* = 0x7FFFFFFF }

Detailed description

This file contains all of the enums and definitions that are used for the ICV and OEM production libraries.

1.7.31 cc_prod_error.h File Reference

This file contains the error definitions of the CryptoCell production-library APIs.

#include "cc_error.h"

Macros

- #define *CC_PROD_INIT_ERR* (*CC_PROD_MODULE_ERROR_BASE* + 0x01UL)
- #define *CC_PROD_INVALID_PARAM_ERR* (*CC_PROD_MODULE_ERROR_BASE* + 0x02UL)
- #define *CC_PROD_ILLEGAL_ZERO_COUNT_ERR* (*CC_PROD_MODULE_ERROR_BASE* + 0x03UL)
- #define *CC_PROD_ILLEGAL_LCS_ERR* (*CC_PROD_MODULE_ERROR_BASE* + 0x04UL)
- #define *CC_PROD_ASSET_PKG_PARAM_ERR* (*CC_PROD_MODULE_ERROR_BASE* + 0x05UL)
- #define *CC_PROD_ASSET_PKG_VERIFY_ERR* (*CC_PROD_MODULE_ERROR_BASE* + 0x06UL)
- #define *CC_PROD_HAL_FATAL_ERR* (*CC_PROD_MODULE_ERROR_BASE* + 0x07UL)

Detailed description

This file contains the error definitions of the CryptoCell production-library APIs.

1.7.32 cc_rnd_common.h File Reference

This file contains the CryptoCell random-number generation APIs.

#include "cc_error.h"

#include "cc_aes_defs.h"

Data structures

- struct *CCRndWorkBuff_t*
- struct *CCRndState_t*
- The structure for the RND state. struct *CCRndContext_t*

Macros

- #define *CC_RND_SEED_MAX_SIZE_WORDS* 12
- #define *CC_RND_MAX_GEN_VECTOR_SIZE_BITS* 0x7FFFF
- #define *CC_RND_MAX_GEN_VECTOR_SIZE_BYTES* 0xFFFF
- #define *CC_RND_REQUESTED_SIZE_COUNTER* 0x3FFFF
- #define *CC_RND_WORK_BUFFER_SIZE_WORDS* 1528
- #define *CC_RND_TRNG_SRC_INNER_OFFSET_WORDS* 2
- #define *CC_RND_TRNG_SRC_INNER_OFFSET_BYTES* (*CC_RND_TRNG_SRC_INNER_OFFSET_WORDS**sizeof(uint32_t))

Typedefs

- typedef int(* *CCRndGenerateVectWorkFunc_t*) (void *rndState_ptr, unsigned char *out_ptr, size_t outSizeBytes)

Enumerations

- enum *CCRndMode_t* { *CC_RND_FE* = 1, CC_RND_ModeLast = 0x7FFFFFFF }

Functions

- CCError_t *CC_RndSetGenerateVectorFunc* (*CCRndContext_t* *rndContext_ptr, *CCRndGenerateVectWorkFunc_t* rndGenerateVectFunc)

  This function sets the RND vector-generation function into the RND context.

Detailed description

This file contains the CryptoCell random-number generation APIs.

The random-number generation module implements NIST Special Publication 800-90A: Recommendation for Random Number Generation Using Deterministic Random Bit Generators.

1.7.33 cc_sram_map.h File Reference

This file contains internal SRAM mapping definitions.

Macros

- #define *CC_SRAM_PKA_BASE_ADDRESS* 0x0
- #define *CC_PKA_SRAM_SIZE_IN_KBYTES* 6
- #define *CC_SRAM_RND_HW_DMA_ADDRESS* 0x0
- #define *CC_SRAM_RND_MAX_SIZE* 0x800
- #define *CC_SRAM_MAX_SIZE* 0x1000

Detailed description

This file contains internal SRAM mapping definitions.

1.7.34 cc_util_error.h File Reference

This file contains the error definitions of the CryptoCell utility APIs.

Macros

- #define CC_UTIL_OK 0x00UL
- #define CC_UTIL_MODULE_ERROR_BASE 0x80000000
- #define CC_UTIL_INVALID_KEY_TYPE (CC_UTIL_MODULE_ERROR_BASE + 0x00UL)
- #define CC_UTIL_DATA_IN_POINTER_INVALID_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x01UL)
- #define CC_UTIL_DATA_IN_SIZE_INVALID_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x02UL)
- #define CC_UTIL_DATA_OUT_POINTER_INVALID_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x03UL)
- #define CC_UTIL_DATA_OUT_SIZE_INVALID_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x04UL)
- #define CC_UTIL_FATAL_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x05UL)
- #define CC_UTIL_ILLEGAL_PARAMS_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x06UL)
- #define CC_UTIL_BAD_ADDR_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x07UL)
- #define CC_UTIL_EK_DOMAIN_INVALID_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x08UL)
- #define CC_UTIL_KDR_INVALID_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x09UL)
- #define CC_UTIL_LCS_INVALID_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x0AUL)
- #define CC_UTIL_SESSION_KEY_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x0BUL)
- #define CC_UTIL_INVALID_USER_KEY_SIZE (CC_UTIL_MODULE_ERROR_BASE + 0x0DUL)
- #define CC_UTIL_ILLEGAL_LCS_FOR_OPERATION_ERR (CC_UTIL_MODULE_ERROR_BASE + 0x0EUL)
- #define CC_UTIL_INVALID_PRF_TYPE (CC_UTIL_MODULE_ERROR_BASE + 0x0FUL)
- #define CC_UTIL_INVALID_HASH_MODE (CC_UTIL_MODULE_ERROR_BASE + 0x10UL)
- #define CC_UTIL_UNSUPPORTED_HASH_MODE (CC_UTIL_MODULE_ERROR_BASE + 0x11UL)
- #define CC_UTIL_KEY_UNUSABLE_ERROR (CC_UTIL_MODULE_ERROR_BASE + 0x12UL)

Detailed description

This file contains the error definitions of the CryptoCell utility APIs.

1.7.35 ccm.h File Reference

This file contains the Mbed TLS CCM APIs.

CCM combines Counter mode encryption with CBC-MAC authentication for 128-bit block ciphers.

#include "cipher.h"

Data structures

- struct *mbedtls_ccm_context*

The CCM context-type definition. The CCM context is passed to the APIs called. Macros

- #define *MBEDTLS_ERR_CCM_BAD_INPUT* -0x000D
- #define *MBEDTLS_ERR_CCM_AUTH_FAILED* -0x000F
- #define *MBEDTLS_ERR_CCM_HW_ACCEL_FAILED* -0x0011

Functions

- void *mbedtls_ccm_init* (*mbedtls_ccm_context* *ctx)

  This function initializes the specified CCM context, to make references valid, and prepare the context for *mbedtls_ccm_setkey()* or *mbedtls_ccm_free()*.

- int *mbedtls_ccm_setkey* (*mbedtls_ccm_context* *ctx, *mbedtls_cipher_id_t* cipher, const unsigned char *key, unsigned int keybits)

  This function initializes the CCM context set in the ctx parameter and sets the encryption key.

- void *mbedtls_ccm_free* (*mbedtls_ccm_context* *ctx)

  This function releases and clears the specified CCM context and underlying cipher sub-context.

- int *mbedtls_ccm_encrypt_and_tag* (*mbedtls_ccm_context* *ctx, size_t length, const unsigned char *iv, size_t iv_len, const unsigned char *add, size_t add_len, const unsigned char *input, unsigned char *output, unsigned char *tag, size_t tag_len)

  This function encrypts a buffer using CCM.

- int *mbedtls_ccm_auth_decrypt* (*mbedtls_ccm_context* *ctx, size_t length, const unsigned char *iv, size_t iv_len, const unsigned char *add, size_t add_len, const unsigned char *input, unsigned char *output, const unsigned char *tag, size_t tag_len)

  This function performs a CCM authenticated decryption of a buffer.

Detailed description

CCM combines Counter mode encryption with CBC-MAC authentication for 128-bit block ciphers.

Input to CCM includes the following elements:

- Payload - data that is both authenticated and encrypted.
- Associated data (Adata) - data that is authenticated but not encrypted, For example, a header.
- Nonce - A unique value that is assigned to the payload and the associated data.

Macro definition documentation

#define MBEDTLS_ERR_CCM_AUTH_FAILED -0x000F

Authenticated decryption failed.

#define MBEDTLS_ERR_CCM_BAD_INPUT -0x000D

Bad input parameters to the function.

#define MBEDTLS_ERR_CCM_HW_ACCEL_FAILED -0x0011

CCM hardware accelerator failed.

Function documentation

int mbedtls_ccm_auth_decrypt (*mbedtls_ccm_context* * ctx, size_t length, const unsigned char * iv, size_t iv_len, const unsigned char * add, size_t add_len, const unsigned char * input, unsigned char * output, const unsigned char * tag, size_t tag_len)

This function performs a CCM authenticated decryption of a buffer.

Parameters:

Parameter	Description
`ctx`	The CCM context to use for decryption.
`length`	The length of the input data in Bytes.
`iv`	Initialization vector.
`iv_len`	The length of the IV in Bytes: 7, 8, 9, 10, 11, 12, or 13.
`add`	The additional data field.
`add_len`	The length of additional data in Bytes.
`input`	The buffer holding the input data.
`output`	The buffer holding the output data.
`tag`	The buffer holding the tag.
`tag_len`	The length of the tag in Bytes.

Returns:

0 if successful and authenticated, or *MBEDTLS_ERR_CCM_AUTH_FAILED* if the tag does not match.

int mbedtls_ccm_encrypt_and_tag (*mbedtls_ccm_context* * ctx, size_t length, const unsigned char * iv, size_t iv_len, const unsigned char * add, size_t add_len, const unsigned char * input, unsigned char * output, unsigned char * tag, size_t tag_len)

This function encrypts a buffer using CCM.

Parameters:

Parameter	Description
`ctx`	The CCM context to use for encryption.
`length`	The length of the input data in Bytes.
`iv`	Initialization vector (nonce).

Parameter	Description
`iv_len`	The length of the IV in Bytes: 7, 8, 9, 10, 11, 12, or 13.
`add`	The additional data field.
`add_len`	The length of additional data in Bytes. Must be less than 2^16 - 2^8.
`input`	The buffer holding the input data.
`output`	The buffer holding the output data. Must be at least length Bytes wide.
`tag`	The buffer holding the tag.
`tag_len`	The length of the tag to generate in Bytes: 4, 6, 8, 10, 14 or 16. **Note** The tag is written to a separate buffer. To concatenate the tag with the output , as done in RFC-3610: Counter with CBC-MAC (CCM) , use tag = output + length , and make sure that the output buffer is at least length + tag_len wide.

Returns:

0   on success.

void mbedtls_ccm_free (mbedtls_ccm_context *   ctx)

This function releases and clears the specified CCM context and underlying cipher sub-context.

Parameters:

Parameter	Description
`ctx`	The CCM context to clear.

void mbedtls_ccm_init (mbedtls_ccm_context *   ctx)

This function initializes the specified CCM context, to make references valid, and prepare the context for *mbedtls_ccm_setkey()* or *mbedtls_ccm_free()*.

Parameters:

Parameter	Description
`ctx`	The CCM context to initialize.

int mbedtls_ccm_setkey (mbedtls_ccm_context *   ctx, mbedtls_cipher_id_t cipher, const unsigned char *   key, unsigned int   keybits)

This function initializes the CCM context set in the ctx   parameter and sets the encryption key.

Parameters:

Parameter	Description
`ctx`	The CCM context to initialize.
`cipher`	The 128-bit block cipher to use.
`key`	The encryption key.
`keybits`	The key size in bits. This must be acceptable by the cipher.

Returns:

0   on success, or a cipher-specific error code.

1.7.36 cipher.h File Reference

This file contains the Mbed TLS generic cipher wrapper.

#include "config.h"

#include <stddef.h>

Data structures

- struct *mbedtls_cipher_info_t*
- struct *mbedtls_cipher_context_t*

Macros

- #define *MBEDTLS_ERR_CIPHER_FEATURE_UNAVAILABLE* -0x6080
- #define *MBEDTLS_ERR_CIPHER_BAD_INPUT_DATA* -0x6100
- #define *MBEDTLS_ERR_CIPHER_ALLOC_FAILED* -0x6180
- #define *MBEDTLS_ERR_CIPHER_INVALID_PADDING* -0x6200
- #define *MBEDTLS_ERR_CIPHER_FULL_BLOCK_EXPECTED* -0x6280
- #define *MBEDTLS_ERR_CIPHER_AUTH_FAILED* -0x6300
- #define *MBEDTLS_ERR_CIPHER_INVALID_CONTEXT* -0x6380
- #define *MBEDTLS_ERR_CIPHER_HW_ACCEL_FAILED* -0x6400
- #define *MBEDTLS_CIPHER_VARIABLE_IV_LEN* 0x01
- #define *MBEDTLS_CIPHER_VARIABLE_KEY_LEN* 0x02
- #define *MBEDTLS_MAX_IV_LENGTH* 16
- #define *MBEDTLS_MAX_BLOCK_LENGTH* 16

Typedefs

- typedef struct *mbedtls_cipher_base_t* *mbedtls_cipher_base_t*
- typedef struct *mbedtls_cmac_context_t* *mbedtls_cmac_context_t*

Enumerations

- enum *mbedtls_cipher_id_t* { MBEDTLS_CIPHER_ID_NONE = 0, MBEDTLS_CIPHER_ID_NULL, MBEDTLS_CIPHER_ID_AES, MBEDTLS_CIPHER_ID_DES, MBEDTLS_CIPHER_ID_3DES, MBEDTLS_CIPHER_ID_CAMELLIA, MBEDTLS_CIPHER_ID_BLOWFISH, MBEDTLS_CIPHER_ID_ARC4 }An enumeration of supported ciphers.

- enum *mbedtls_cipher_type_t* { MBEDTLS_CIPHER_NONE = 0, MBEDTLS_CIPHER_NULL, MBEDTLS_CIPHER_AES_128_ECB, MBEDTLS_CIPHER_AES_192_ECB, MBEDTLS_CIPHER_AES_256_ECB, MBEDTLS_CIPHER_AES_128_CBC, MBEDTLS_CIPHER_AES_192_CBC, MBEDTLS_CIPHER_AES_256_CBC, MBEDTLS_CIPHER_AES_128_CFB128, MBEDTLS_CIPHER_AES_192_CFB128, MBEDTLS_CIPHER_AES_256_CFB128, MBEDTLS_CIPHER_AES_128_CTR, MBEDTLS_CIPHER_AES_192_CTR, MBEDTLS_CIPHER_AES_256_CTR, MBEDTLS_CIPHER_AES_128_GCM, MBEDTLS_CIPHER_AES_192_GCM, MBEDTLS_CIPHER_AES_256_GCM, MBEDTLS_CIPHER_CAMELLIA_128_ECB, MBEDTLS_CIPHER_CAMELLIA_192_ECB, MBEDTLS_CIPHER_CAMELLIA_256_ECB, MBEDTLS_CIPHER_CAMELLIA_128_CBC, MBEDTLS_CIPHER_CAMELLIA_192_CBC, MBEDTLS_CIPHER_CAMELLIA_256_CBC, MBEDTLS_CIPHER_CAMELLIA_128_CFB128, MBEDTLS_CIPHER_CAMELLIA_192_CFB128, MBEDTLS_CIPHER_CAMELLIA_256_CFB128, MBEDTLS_CIPHER_CAMELLIA_128_CTR, MBEDTLS_CIPHER_CAMELLIA_192_CTR, MBEDTLS_CIPHER_CAMELLIA_256_CTR, MBEDTLS_CIPHER_CAMELLIA_128_GCM, MBEDTLS_CIPHER_CAMELLIA_192_GCM, MBEDTLS_CIPHER_CAMELLIA_256_GCM, MBEDTLS_CIPHER_DES_ECB, MBEDTLS_CIPHER_DES_CBC, MBEDTLS_CIPHER_DES_EDE_ECB, MBEDTLS_CIPHER_DES_EDE_CBC, MBEDTLS_CIPHER_DES_EDE3_ECB, MBEDTLS_CIPHER_DES_EDE3_CBC, MBEDTLS_CIPHER_BLOWFISH_ECB, MBEDTLS_CIPHER_BLOWFISH_CBC, MBEDTLS_CIPHER_BLOWFISH_CFB64, MBEDTLS_CIPHER_BLOWFISH_CTR, MBEDTLS_CIPHER_ARC4_128, MBEDTLS_CIPHER_AES_128_CCM, MBEDTLS_CIPHER_AES_192_CCM, MBEDTLS_CIPHER_AES_256_CCM, MBEDTLS_CIPHER_CAMELLIA_128_CCM, MBEDTLS_CIPHER_CAMELLIA_192_CCM, MBEDTLS_CIPHER_CAMELLIA_256_CCM }An enumeration of supported (cipher, mode) pairs.
- enum *mbedtls_cipher_mode_t* { MBEDTLS_MODE_NONE = 0, MBEDTLS_MODE_ECB, MBEDTLS_MODE_CBC, MBEDTLS_MODE_CFB, MBEDTLS_MODE_OFB, MBEDTLS_MODE_CTR, MBEDTLS_MODE_GCM, MBEDTLS_MODE_STREAM, MBEDTLS_MODE_CCM }
- enum *mbedtls_cipher_padding_t* { *MBEDTLS_PADDING_PKCS7* = 0, *MBEDTLS_PADDING_ONE_AND_ZEROS*, *MBEDTLS_PADDING_ZEROS_AND_LEN*, *MBEDTLS_PADDING_ZEROS*, *MBEDTLS_PADDING_NONE* }
- enum *mbedtls_operation_t* { MBEDTLS_OPERATION_NONE = -1, MBEDTLS_DECRYPT = 0, MBEDTLS_ENCRYPT }
- enum { *MBEDTLS_KEY_LENGTH_NONE* = 0, *MBEDTLS_KEY_LENGTH_DES* = 64, *MBEDTLS_KEY_LENGTH_DES_EDE* = 128, *MBEDTLS_KEY_LENGTH_DES_EDE3* = 192 }

Functions

- const int * *mbedtls_cipher_list* (void)

  This function retrieves the list of ciphers supported by the generic cipher module.
- const *mbedtls_cipher_info_t* * *mbedtls_cipher_info_from_string* (const char *cipher_name)

  This function retrieves the cipher-information structure associated with the given cipher name.
- const *mbedtls_cipher_info_t* * *mbedtls_cipher_info_from_type* (const *mbedtls_cipher_type_t* cipher_type)

  This function retrieves the cipher-information structure associated with the given cipher type.

- const *mbedtls_cipher_info_t* * *mbedtls_cipher_info_from_values* (const *mbedtls_cipher_id_t* cipher_id, int key_bitlen, const *mbedtls_cipher_mode_t* mode)

  This function retrieves the cipher-information structure associated with the given cipher ID, key size and mode.

- void *mbedtls_cipher_init* (*mbedtls_cipher_context_t* *ctx)

  This function initializes a cipher_context as NONE.

- void *mbedtls_cipher_free* (*mbedtls_cipher_context_t* *ctx)

  This function frees and clears the cipher-specific context of ctx . Freeing ctx itself remains the responsibility of the caller.

- int *mbedtls_cipher_setup* (*mbedtls_cipher_context_t* *ctx, const *mbedtls_cipher_info_t* *cipher_info)

  This function initializes and fills the cipher-context structure with the appropriate values. It also clears the structure.

- int *mbedtls_cipher_setkey* (*mbedtls_cipher_context_t* *ctx, const unsigned char *key, int key_bitlen, const *mbedtls_operation_t* operation)

  This function sets the key to use with the given context.

- int *mbedtls_cipher_set_iv* (*mbedtls_cipher_context_t* *ctx, const unsigned char *iv, size_t iv_len)

  This function sets the initialization vector (IV) or nonce.

- int *mbedtls_cipher_reset* (*mbedtls_cipher_context_t* *ctx)

  This function resets the cipher state.

- int *mbedtls_cipher_update* (*mbedtls_cipher_context_t* *ctx, const unsigned char *input, size_t ilen, unsigned char *output, size_t *olen)

  The generic cipher update function. It encrypts or decrypts using the given cipher context. Writes as many block-sized blocks of data as possible to output. Any data that cannot be written immediately is either added to the next block, or flushed when *mbedtls_cipher_finish()* is called. Exception: For MBEDTLS_MODE_ECB, expects a single block in size. For example, 16 Bytes for AES.

- int *mbedtls_cipher_finish* (*mbedtls_cipher_context_t* *ctx, unsigned char *output, size_t *olen)

  The generic cipher finalization function. If data still needs to be flushed from an incomplete block, the data contained in it is padded to the size of the last block, and written to the output buffer.

- int *mbedtls_cipher_crypt* (*mbedtls_cipher_context_t* *ctx, const unsigned char *iv, size_t iv_len, const unsigned char *input, size_t ilen, unsigned char *output, size_t *olen)

  The generic all-in-one encryption/decryption function, for all ciphers except AEAD constructs.

Detailed description

The generic cipher wrapper.

Macro definition documentation

#define MBEDTLS_CIPHER_VARIABLE_IV_LEN 0x01

Cipher accepts IVs of variable length.

#define MBEDTLS_CIPHER_VARIABLE_KEY_LEN 0x02

Cipher accepts keys of variable length.

#define MBEDTLS_ERR_CIPHER_ALLOC_FAILED -0x6180

Failed to allocate memory.

#define MBEDTLS_ERR_CIPHER_AUTH_FAILED -0x6300

Authentication failed (for AEAD modes).

#define MBEDTLS_ERR_CIPHER_BAD_INPUT_DATA -0x6100

Bad input parameters.

#define MBEDTLS_ERR_CIPHER_FEATURE_UNAVAILABLE -0x6080

The selected feature is not available.

#define MBEDTLS_ERR_CIPHER_FULL_BLOCK_EXPECTED -0x6280

Decryption of block requires a full block.

#define MBEDTLS_ERR_CIPHER_HW_ACCEL_FAILED -0x6400

Cipher hardware accelerator failed.

#define MBEDTLS_ERR_CIPHER_INVALID_CONTEXT -0x6380

The context is invalid. For example, because it was freed.

#define MBEDTLS_ERR_CIPHER_INVALID_PADDING -0x6200

Input data contains invalid padding and is rejected.

#define MBEDTLS_MAX_BLOCK_LENGTH 16

Maximum block size of any cipher, in Bytes.

#define MBEDTLS_MAX_IV_LENGTH 16

Maximum length of any IV, in Bytes.

Typedef documentation

typedef struct *mbedtls_cipher_base_t* *mbedtls_cipher_base_t*

Base cipher information (opaque struct).

typedef struct *mbedtls_cmac_context_t* *mbedtls_cmac_context_t*

CMAC context (opaque struct).

Enumeration type documentation

anonymous enum

Enumerator:

Enum	Description
MBEDTLS_KEY_LENGTH_NONE	Undefined key length.
MBEDTLS_KEY_LENGTH_DES	Key length, in bits (including parity), for DES keys.

Enum	Description
MBEDTLS_KEY_LENGTH_DES_EDE	Key length in bits, including parity, for DES in two-key EDE.
MBEDTLS_KEY_LENGTH_DES_EDE3	Key length in bits, including parity, for DES in three-key EDE.

enum *mbedtls_cipher_id_t*

An enumeration of supported ciphers.

Warning

ARC4 and DES are considered weak ciphers and their use constitutes a security risk. We recommend considering stronger ciphers instead.

enum *mbedtls_cipher_mode_t*

Supported cipher modes.

enum *mbedtls_cipher_padding_t*

Supported cipher padding types.

Enumerator:

Enum	Description
MBEDTLS_PADDING_PKCS7	PKCS7 padding (default).
MBEDTLS_PADDING_ONE_AND_ZEROS	ISO/IEC 7816-4 padding.
MBEDTLS_PADDING_ZEROS_AND_LEN	ANSI X.923 padding.
MBEDTLS_PADDING_ZEROS	zero padding (not reversible).
MBEDTLS_PADDING_NONE	never pad (full blocks only).

enum *mbedtls_cipher_type_t*

An enumeration of supported (cipher, mode) pairs.

Warning

ARC4 and DES are considered weak ciphers and their use constitutes a security risk. We recommend considering stronger ciphers instead.

enum *mbedtls_operation_t*

Type of operation.

Function documentation

int mbedtls_cipher_crypt (*mbedtls_cipher_context_t* * ctx, const unsigned char * iv, size_t iv_len, const unsigned char * input, size_t ilen, unsigned char * output, size_t * olen)

The generic all-in-one encryption/decryption function, for all ciphers except AEAD constructs.

Parameters:

Parameter	Description
`ctx`	The generic cipher context.
`iv`	The IV to use, or NONCE_COUNTER for CTR-mode ciphers.
`iv_len`	The IV length for ciphers with variable-size IV. This parameter is discarded by ciphers with fixed-size IV.
`input`	The buffer holding the input data.
`ilen`	The length of the input data.
`output`	The buffer for the output data. Must be able to hold at least ilen + block_size. Must not be the same buffer as input.
`olen`	The length of the output data, to be updated with the actual number of Bytes written. **Note** Some ciphers do not use IVs nor nonce. For these ciphers, use iv = NULL and iv_len = 0.

Returns:

0 on success, or *MBEDTLS_ERR_CIPHER_BAD_INPUT_DATA*, or *MBEDTLS_ERR_CIPHER_FULL_BLOCK_EXPECTED* if decryption expected a full block but was not provided one, or *MBEDTLS_ERR_CIPHER_INVALID_PADDING* on invalid padding while decrypting, or a cipher-specific error code on failure for any other reason.

int mbedtls_cipher_finish (*mbedtls_cipher_context_t* * ctx, unsigned char * output, size_t * olen)

The generic cipher finalization function. If data still needs to be flushed from an incomplete block, the data contained in it is padded to the size of the last block, and written to the output buffer.

Parameters:

Parameter	Description
`ctx`	The generic cipher context.
`output`	The buffer to write data to. Needs block_size available.
`olen`	The length of the data written to the output buffer.

Returns:

0 on success, *MBEDTLS_ERR_CIPHER_BAD_INPUT_DATA* if parameter verification fails, *MBEDTLS_ERR_CIPHER_FULL_BLOCK_EXPECTED* if decryption expected a full block but was not provided one, *MBEDTLS_ERR_CIPHER_INVALID_PADDING* on

invalid padding while decrypting, or a cipher-specific error code on failure for any other reason.

const *mbedtls_cipher_info_t** mbedtls_cipher_info_from_string (const char * cipher_name)

This function retrieves the cipher-information structure associated with the given cipher name.

Parameters:

Parameter	Description
`cipher_name`	Name of the cipher to search for.

Returns:

The cipher information structure associated with the given cipher_name , or NULL if not found.

const *mbedtls_cipher_info_t** mbedtls_cipher_info_from_type (const *mbedtls_cipher_type_t* cipher_type)

This function retrieves the cipher-information structure associated with the given cipher type.

Parameters:

Parameter	Description
`cipher_type`	Type of the cipher to search for.

Returns:

The cipher information structure associated with the given cipher_type , or NULL if not found.

const *mbedtls_cipher_info_t** mbedtls_cipher_info_from_values (const *mbedtls_cipher_id_t* cipher_id, int key_bitlen, const *mbedtls_cipher_mode_t* mode)

This function retrieves the cipher-information structure associated with the given cipher ID, key size and mode.

Parameters:

Parameter	Description
`cipher_id`	The ID of the cipher to search for. For example, MBEDTLS_CIPHER_ID_AES .
`key_bitlen`	The length of the key in bits.
`mode`	The cipher mode. For example, MBEDTLS_MODE_CBC .

Returns:

The cipher information structure associated with the given cipher_id , or NULL if not found.

const int* mbedtls_cipher_list (void)

This function retrieves the list of ciphers supported by the generic cipher module.

Returns:

A statically-allocated array of ciphers. The last entry is zero.

int mbedtls_cipher_reset (mbedtls_cipher_context_t * ctx)

This function resets the cipher state.

Parameters:

Parameter	Description
`ctx`	The generic cipher context.

Returns:

0 on success, *MBEDTLS_ERR_CIPHER_BAD_INPUT_DATA* if parameter verification fails.

int mbedtls_cipher_set_iv (mbedtls_cipher_context_t * ctx, const unsigned char * iv, size_t iv_len)

This function sets the initialization vector (IV) or nonce.

Parameters:

Parameter	Description
`ctx`	The generic cipher context.
`iv`	The IV to use, or NONCE_COUNTER for CTR-mode ciphers.
`iv_len`	The IV length for ciphers with variable-size IV. This parameter is discarded by ciphers with fixed-size IV.

Returns:

0 on success, or *MBEDTLS_ERR_CIPHER_BAD_INPUT_DATA*

Note

Some ciphers do not use IVs nor nonce. For these ciphers, this function has no effect.

int mbedtls_cipher_setkey (mbedtls_cipher_context_t * ctx, const unsigned char * key, int key_bitlen, const mbedtls_operation_t operation)

This function sets the key to use with the given context.

Parameters:

Parameter	Description
`ctx`	The generic cipher context. May not be NULL. Must have been initialized using *mbedtls_cipher_info_from_type()* or *mbedtls_cipher_info_from_string()*.
`key`	The key to use.
`key_bitlen`	The key length to use, in bits.
`operation`	The operation that the key will be used for: MBEDTLS_ENCRYPT or MBEDTLS_DECRYPT .

Returns:

0 on success, *MBEDTLS_ERR_CIPHER_BAD_INPUT_DATA* if parameter verification fails, or a cipher-specific error code.

int mbedtls_cipher_setup (*mbedtls_cipher_context_t* * ctx, const *mbedtls_cipher_info_t* * cipher_info)

This function initializes and fills the cipher-context structure with the appropriate values. It also clears the structure.

Parameters:

Parameter	Description
ctx	The context to initialize. May not be NULL.
cipher_info	The cipher to use.

Returns:

0 on success, *MBEDTLS_ERR_CIPHER_BAD_INPUT_DATA* on parameter failure, *MBEDTLS_ERR_CIPHER_ALLOC_FAILED* if allocation of the cipher-specific context failed.

int mbedtls_cipher_update (*mbedtls_cipher_context_t* * ctx, const unsigned char * input, size_t ilen, unsigned char * output, size_t * olen)

The generic cipher update function. It encrypts or decrypts using the given cipher context. Writes as many block-sized blocks of data as possible to output. Any data that cannot be written immediately is either added to the next block, or flushed when *mbedtls_cipher_finish()* is called. Exception: For MBEDTLS_MODE_ECB, expects a single block in size. For example, 16 Bytes for AES.

Parameters:

Parameter	Description
ctx	The generic cipher context.
input	The buffer holding the input data.
ilen	The length of the input data.
output	The buffer for the output data. Must be able to hold at least ilen + block_size. Must not be the same buffer as input.
olen	The length of the output data, to be updated with the actual number of Bytes written.

Returns:

0 on success, *MBEDTLS_ERR_CIPHER_BAD_INPUT_DATA* if parameter verification fails, *MBEDTLS_ERR_CIPHER_FEATURE_UNAVAILABLE* on an unsupported mode for a cipher, or a cipher-specific error code.

Note

If the underlying cipher is GCM, all calls to this function, except the last one before *mbedtls_cipher_finish()*. Must have ilen as a multiple of the block_size.

1.7.37 cmac.h File Reference

This file contains the Mbed TLS CMAC APIs.

The Cipher-based Message Authentication Code (CMAC) Mode for Authentication.

#include "mbedtls/cipher.h"

Data structures

- struct *mbedtls_cmac_context_t*

Macros

- #define *MBEDTLS_ERR_CMAC_HW_ACCEL_FAILED* -0x007A
- #define *MBEDTLS_AES_BLOCK_SIZE* 16
- #define *MBEDTLS_DES3_BLOCK_SIZE* 8
- #define *MBEDTLS_CIPHER_BLKSIZE_MAX* 8 /* The longest block used by CMAC is that of 3DES. */

Functions

- int *mbedtls_cipher_cmac_starts* (*mbedtls_cipher_context_t* *ctx, const unsigned char *key, size_t keybits)

  This function sets the CMAC key, and prepares to authenticate the input data. Must be called with an initialized cipher context.
- int *mbedtls_cipher_cmac_update* (*mbedtls_cipher_context_t* *ctx, const unsigned char *input, size_t ilen)

  This function feeds an input buffer into an ongoing CMAC computation.
- int *mbedtls_cipher_cmac_finish* (*mbedtls_cipher_context_t* *ctx, unsigned char *output)

  This function finishes the CMAC operation, and writes the result to the output buffer.
- int *mbedtls_cipher_cmac_reset* (*mbedtls_cipher_context_t* *ctx)

  This function prepares the authentication of another message with the same key as the previous CMAC operation.
- int *mbedtls_cipher_cmac* (const *mbedtls_cipher_info_t* *cipher_info, const unsigned char *key, size_t keylen, const unsigned char *input, size_t ilen, unsigned char *output)

  This function calculates the full generic CMAC on the input buffer with the provided key.

Detailed description

The Cipher-based Message Authentication Code (CMAC) Mode for Authentication.

Macro definition documentation

#define MBEDTLS_AES_BLOCK_SIZE 16

The size of the AES block.

#define MBEDTLS_CIPHER_BLKSIZE_MAX 8 /* The longest block used by CMAC is that of 3DES. */

The maximal size of block used by CMAC.

#define MBEDTLS_DES3_BLOCK_SIZE 8

The size of the 3DES block.

#define MBEDTLS_ERR_CMAC_HW_ACCEL_FAILED -0x007A

CMAC hardware accelerator failed.

Function documentation

int mbedtls_cipher_cmac (const mbedtls_cipher_info_t * cipher_info, const unsigned char * key, size_t keylen, const unsigned char * input, size_t ilen, unsigned char * output)

This function calculates the full generic CMAC on the input buffer with the provided key.

The function allocates the context, performs the calculation, and frees the context.

The CMAC result is calculated as output = generic CMAC(cmac key, input buffer).

Parameters:

Parameter	Description
`cipher_info`	The cipher information.
`key`	The CMAC key.
`keylen`	The length of the CMAC key in bits.
`input`	The buffer holding the input data.
`ilen`	The length of the input data.
`output`	The buffer for the generic CMAC result.

Returns:

0 on success, or *MBEDTLS_ERR_MD_BAD_INPUT_DATA* if parameter verification fails.

int mbedtls_cipher_cmac_finish (mbedtls_cipher_context_t * ctx, unsigned char * output)

This function finishes the CMAC operation, and writes the result to the output buffer.

It is called after *mbedtls_cipher_cmac_update()*. It can be followed by *mbedtls_cipher_cmac_reset()* and *mbedtls_cipher_cmac_update()*, or *mbedtls_cipher_free()*.

Parameters:

Parameter	Description
`ctx`	The cipher context used for the CMAC operation.
`output`	The output buffer for the CMAC checksum result.

Returns:

0 on success, or *MBEDTLS_ERR_MD_BAD_INPUT_DATA* if parameter verification fails.

int mbedtls_cipher_cmac_reset (mbedtls_cipher_context_t * ctx)

This function prepares the authentication of another message with the same key as the previous CMAC operation.

It is called after *mbedtls_cipher_cmac_finish()* and before *mbedtls_cipher_cmac_update()*.

Parameters:

Parameter	Description
`ctx`	The cipher context used for the CMAC operation.

Returns:

0 on success, or *MBEDTLS_ERR_MD_BAD_INPUT_DATA* if parameter verification fails.

int mbedtls_cipher_cmac_starts (*mbedtls_cipher_context_t* * ctx, const unsigned char * key, size_t keybits)

This function sets the CMAC key, and prepares to authenticate the input data. Must be called with an initialized cipher context.

Parameters:

Parameter	Description
`ctx`	The cipher context used for the CMAC operation, initialized as one of the following types: • MBEDTLS_CIPHER_AES_128_ECB • MBEDTLS_CIPHER_AES_192_ECB • MBEDTLS_CIPHER_AES_256_ECB • MBEDTLS_CIPHER_DES_EDE3_ECB
`key`	The CMAC key.
`keybits`	The length of the CMAC key in bits. Must be supported by the cipher.

Returns:

0 on success, or a cipher-specific error code.

int mbedtls_cipher_cmac_update (*mbedtls_cipher_context_t* * ctx, const unsigned char * input, size_t ilen)

This function feeds an input buffer into an ongoing CMAC computation.

It is called between *mbedtls_cipher_cmac_starts()* or *mbedtls_cipher_cmac_reset()*, and *mbedtls_cipher_cmac_finish()*. Can be called repeatedly.

Parameters:

Parameter	Description
`ctx`	The cipher context used for the CMAC operation.
`input`	The buffer holding the input data.
`ilen`	The length of the input data.

Returns:

0 on success, or *MBEDTLS_ERR_MD_BAD_INPUT_DATA* if parameter verification fails.

1.7.38 ctr_drbg.h File Reference

This file contains the Mbed TLS CTR_DRBG APIs.

CTR_DRBG is based on AES-256, as defined in NIST SP 800-90A: Recommendation for Random Number Generation Using Deterministic Random Bit Generators .

#include "aes.h"

Data structures

- struct *mbedtls_ctr_drbg_context*

The CTR_DRBG context structure. Macros

- #define *MBEDTLS_ERR_CTR_DRBG_ENTROPY_SOURCE_FAILED* -0x0034
- #define *MBEDTLS_ERR_CTR_DRBG_REQUEST_TOO_BIG* -0x0036
- #define *MBEDTLS_ERR_CTR_DRBG_INPUT_TOO_BIG* -0x0038
- #define *MBEDTLS_ERR_CTR_DRBG_FILE_IO_ERROR* -0x003A
- #define *MBEDTLS_CTR_DRBG_BLOCKSIZE* 16
- #define *MBEDTLS_CTR_DRBG_KEYSIZE* 32
- #define *MBEDTLS_CTR_DRBG_KEYBITS* (*MBEDTLS_CTR_DRBG_KEYSIZE* * 8)
- #define *MBEDTLS_CTR_DRBG_SEEDLEN* (*MBEDTLS_CTR_DRBG_KEYSIZE* + *MBEDTLS_CTR_DRBG_BLOCKSIZE*)
- #define *MBEDTLS_CTR_DRBG_PR_OFF* 0
- #define *MBEDTLS_CTR_DRBG_PR_ON* 1

Module settings

The configuration options you can set for this module are in this section. Either change them in config.h or define them using the compiler command line.

- #define *MBEDTLS_CTR_DRBG_ENTROPY_LEN* 32
- #define *MBEDTLS_CTR_DRBG_RESEED_INTERVAL* 10000
- #define *MBEDTLS_CTR_DRBG_MAX_INPUT* 256
- #define *MBEDTLS_CTR_DRBG_MAX_REQUEST* 1024
- #define *MBEDTLS_CTR_DRBG_MAX_SEED_INPUT* 384

Functions

- void *mbedtls_ctr_drbg_init* (*mbedtls_ctr_drbg_context* *ctx)

  This function initializes the CTR_DRBG context, and prepares it for *mbedtls_ctr_drbg_seed()* or *mbedtls_ctr_drbg_free()*.

- int *mbedtls_ctr_drbg_seed* (*mbedtls_ctr_drbg_context* *ctx, int(*f_entropy)(void *, unsigned char *, size_t), void *p_entropy, const unsigned char *custom, size_t len)

  This function seeds and sets up the CTR_DRBG entropy source for future reseeds.

- void *mbedtls_ctr_drbg_free* (*mbedtls_ctr_drbg_context* *ctx)

  This function clears CTR_CRBG context data.

- void *mbedtls_ctr_drbg_set_prediction_resistance* (*mbedtls_ctr_drbg_context* *ctx, int resistance)

  This function turns prediction resistance on or off. The default value is off.

- void *mbedtls_ctr_drbg_set_entropy_len* (*mbedtls_ctr_drbg_context* *ctx, size_t len)

  This function sets the amount of entropy grabbed on each seed or reseed. The default value is *MBEDTLS_CTR_DRBG_ENTROPY_LEN*.

- void *mbedtls_ctr_drbg_set_reseed_interval* (*mbedtls_ctr_drbg_context* *ctx, int interval)

  This function sets the reseed interval. The default value is *MBEDTLS_CTR_DRBG_RESEED_INTERVAL*.

- int *mbedtls_ctr_drbg_reseed* (*mbedtls_ctr_drbg_context* *ctx, const unsigned char *additional, size_t len)

  This function reseeds the CTR_DRBG context, that is extracts data from the entropy source.

- void *mbedtls_ctr_drbg_update* (*mbedtls_ctr_drbg_context* *ctx, const unsigned char *additional, size_t add_len)

  This function updates the state of the CTR_DRBG context.

- int *mbedtls_ctr_drbg_random_with_add* (void *p_rng, unsigned char *output, size_t output_len, const unsigned char *additional, size_t add_len)

  This function updates a CTR_DRBG instance with additional data and uses it to generate random data.

- int *mbedtls_ctr_drbg_random* (void *p_rng, unsigned char *output, size_t output_len)

  This function uses CTR_DRBG to generate random data.

- int *mbedtls_ctr_drbg_self_test* (int verbose)

  The CTR_DRBG checkup routine.

- int mbedtls_ctr_drbg_seed_entropy_len (*mbedtls_ctr_drbg_context* *, int(*)(void *, unsigned char *, size_t), void *, const unsigned char *, size_t, size_t)

Detailed description

CTR_DRBG is based on AES-256, as defined in NIST SP 800-90A: Recommendation for Random Number Generation Using Deterministic Random Bit Generators .

Macro definition documentation

#define MBEDTLS_CTR_DRBG_BLOCKSIZE 16

The block size used by the cipher.

#define MBEDTLS_CTR_DRBG_ENTROPY_LEN 32

Amount of entropy used per seed by default:

- 48 with SHA-512.
- 32 with SHA-256.

#define MBEDTLS_CTR_DRBG_KEYBITS (*MBEDTLS_CTR_DRBG_KEYSIZE* * 8)

The key size for the DRBG operation, in bits.

#define MBEDTLS_CTR_DRBG_KEYSIZE 32

The key size used by the cipher.

#define MBEDTLS_CTR_DRBG_MAX_INPUT 256

The maximum number of additional input Bytes.

#define MBEDTLS_CTR_DRBG_MAX_REQUEST 1024

The maximum number of requested Bytes per call.

#define MBEDTLS_CTR_DRBG_MAX_SEED_INPUT 384

The maximum size of seed or reseed buffer.

#define MBEDTLS_CTR_DRBG_PR_OFF 0

Prediction resistance is disabled.

#define MBEDTLS_CTR_DRBG_PR_ON 1

Prediction resistance is enabled.

#define MBEDTLS_CTR_DRBG_RESEED_INTERVAL 10000

The interval before reseed is performed by default.

#define MBEDTLS_CTR_DRBG_SEEDLEN (*MBEDTLS_CTR_DRBG_KEYSIZE* + *MBEDTLS_CTR_DRBG_BLOCKSIZE*)

The seed length, calculated as (counter + AES key).

#define MBEDTLS_ERR_CTR_DRBG_ENTROPY_SOURCE_FAILED -0x0034

The entropy source failed.

#define MBEDTLS_ERR_CTR_DRBG_FILE_IO_ERROR -0x003A

Read or write error in file.

#define MBEDTLS_ERR_CTR_DRBG_INPUT_TOO_BIG -0x0038

The input (entropy + additional data) is too large.

#define MBEDTLS_ERR_CTR_DRBG_REQUEST_TOO_BIG -0x0036

The requested random buffer length is too big.

Function documentation

void mbedtls_ctr_drbg_free (*mbedtls_ctr_drbg_context* * ctx)

This function clears CTR_CRBG context data.

Parameters:

Parameter	Description
`ctx`	`The CTR_DRBG context to clear.`

void mbedtls_ctr_drbg_init (*mbedtls_ctr_drbg_context* * ctx)

This function initializes the CTR_DRBG context, and prepares it for *mbedtls_ctr_drbg_seed()* or *mbedtls_ctr_drbg_free()*.

Parameters:

Parameter	Description
`ctx`	The CTR_DRBG context to initialize.

int mbedtls_ctr_drbg_random (void * p_rng, unsigned char * output, size_t output_len)

This function uses CTR_DRBG to generate random data.

Note

The function automatically reseeds if the reseed counter is exceeded.

Parameters:

Parameter	Description
`p_rng`	The CTR_DRBG context. This must be a pointer to a *mbedtls_ctr_drbg_context* structure.

Parameter	Description
`output`	The buffer to fill.
`output_len`	The length of the buffer.

Returns:

0 on success, or *MBEDTLS_ERR_CTR_DRBG_ENTROPY_SOURCE_FAILED* or *MBEDTLS_ERR_CTR_DRBG_REQUEST_TOO_BIG* on failure.

int mbedtls_ctr_drbg_random_with_add (void * p_rng, unsigned char * output, size_t output_len, const unsigned char * additional, size_t add_len)

This function updates a CTR_DRBG instance with additional data and uses it to generate random data.

Note

The function automatically reseeds if the reseed counter is exceeded.

Parameters:

Parameter	Description
`p_rng`	The CTR_DRBG context. This must be a pointer to a *mbedtls_ctr_drbg_context* structure.
`output`	The buffer to fill.
`output_len`	The length of the buffer.
`additional`	Additional data to update. Can be NULL.
`add_len`	The length of the additional data.

Returns:

0 on success, or *MBEDTLS_ERR_CTR_DRBG_ENTROPY_SOURCE_FAILED* or *MBEDTLS_ERR_CTR_DRBG_REQUEST_TOO_BIG* on failure.

int mbedtls_ctr_drbg_reseed (*mbedtls_ctr_drbg_context* * ctx, const unsigned char * additional, size_t len)

This function reseeds the CTR_DRBG context, that is extracts data from the entropy source.

Parameters:

Parameter	Description
`ctx`	The CTR_DRBG context.
`additional`	Additional data to add to the state. Can be NULL.
`len`	The length of the additional data.

Returns:

0 on success, or *MBEDTLS_ERR_CTR_DRBG_ENTROPY_SOURCE_FAILED* on failure.

int mbedtls_ctr_drbg_seed (*mbedtls_ctr_drbg_context* * ctx, int(*)(void *, unsigned char *, size_t) f_entropy, void * p_entropy, const unsigned char * custom, size_t len)

This function seeds and sets up the CTR_DRBG entropy source for future reseeds.

Note

Personalization data can be provided in addition to the more generic entropy source, to make this instantiation as unique as possible.

Parameters:

Parameter	Description
`ctx`	The CTR_DRBG context to seed.
`f_entropy`	The entropy callback, taking as arguments the p_entropy context, the buffer to fill, and the length of the buffer.
`p_entropy`	The entropy context.
`custom`	Personalization data, that is device-specific identifiers. Can be NULL.
`len`	The length of the personalization data.

Returns:

0 on success, or *MBEDTLS_ERR_CTR_DRBG_ENTROPY_SOURCE_FAILED* on failure.

int mbedtls_ctr_drbg_self_test (int verbose)

The CTR_DRBG checkup routine.

Returns:

0 on success, or 1 on failure.

void mbedtls_ctr_drbg_set_entropy_len (*mbedtls_ctr_drbg_context* * ctx, size_t len)

This function sets the amount of entropy grabbed on each seed or reseed. The default value is *MBEDTLS_CTR_DRBG_ENTROPY_LEN*.

Parameters:

Parameter	Description
`ctx`	The CTR_DRBG context.
`len`	The amount of entropy to grab.

void mbedtls_ctr_drbg_set_prediction_resistance (*mbedtls_ctr_drbg_context* * ctx, int resistance)

This function turns prediction resistance on or off. The default value is off.

Note

If enabled, entropy is gathered at the beginning of every call to *mbedtls_ctr_drbg_random_with_add()*. Only use this if your entropy source has sufficient throughput.

Parameters:

Parameter	Description
`ctx`	The CTR_DRBG context.
`resistance`	*MBEDTLS_CTR_DRBG_PR_ON* or *MBEDTLS_CTR_DRBG_PR_OFF*.

void mbedtls_ctr_drbg_set_reseed_interval (*mbedtls_ctr_drbg_context* * ctx, int interval)

This function sets the reseed interval. The default value is *MBEDTLS_CTR_DRBG_RESEED_INTERVAL*.

Parameters:

Parameter	Description
`ctx`	The CTR_DRBG context.
`interval`	The reseed interval.

void mbedtls_ctr_drbg_update (*mbedtls_ctr_drbg_context* * ctx, const unsigned char * additional, size_t add_len)

This function updates the state of the CTR_DRBG context.

Parameters:

Parameter	Description
`ctx`	The CTR_DRBG context.
`additional`	The data to update the state with.
`add_len`	Length of additional data. **Note** If add_len is greater than *MBEDTLS_CTR_DRBG_MAX_SEED_INPUT*, only the first *MBEDTLS_CTR_DRBG_MAX_SEED_INPUT* Bytes are used. The remaining Bytes are silently discarded.

1.7.39 dhm.h File Reference

This file contains the Mbed TLS Diffie-Hellman-Merkle key exchange APIs.

#include "config.h"

#include "bignum.h"

Data structures

- struct *mbedtls_dhm_context*

The DHM context structure. Macros

- #define *MBEDTLS_ERR_DHM_BAD_INPUT_DATA* -0x3080
- #define *MBEDTLS_ERR_DHM_READ_PARAMS_FAILED* -0x3100
- #define *MBEDTLS_ERR_DHM_MAKE_PARAMS_FAILED* -0x3180
- #define *MBEDTLS_ERR_DHM_READ_PUBLIC_FAILED* -0x3200
- #define *MBEDTLS_ERR_DHM_MAKE_PUBLIC_FAILED* -0x3280

- #define *MBEDTLS_ERR_DHM_CALC_SECRET_FAILED* -0x3300
- #define *MBEDTLS_ERR_DHM_INVALID_FORMAT* -0x3380
- #define *MBEDTLS_ERR_DHM_ALLOC_FAILED* -0x3400
- #define *MBEDTLS_ERR_DHM_FILE_IO_ERROR* -0x3480
- #define *MBEDTLS_ERR_DHM_HW_ACCEL_FAILED* -0x3500
- #define *MBEDTLS_ERR_DHM_SET_GROUP_FAILED* -0x3580
- #define *MBEDTLS_DEPRECATED_STRING_CONSTANT*(VAL) VAL
- #define *MBEDTLS_DHM_RFC5114_MODP_P*
- #define *MBEDTLS_DHM_RFC5114_MODP_2048_G*
- #define *MBEDTLS_DHM_RFC3526_MODP_2048_P*
- #define *MBEDTLS_DHM_RFC3526_MODP_2048_G* *MBEDTLS_DEPRECATED_STRING_CONSTANT*("02")
- #define *MBEDTLS_DHM_RFC3526_MODP_3072_P*
- #define *MBEDTLS_DHM_RFC3526_MODP_3072_G* *MBEDTLS_DEPRECATED_STRING_CONSTANT*("02")
- #define *MBEDTLS_DHM_RFC3526_MODP_4096_P*
- #define *MBEDTLS_DHM_RFC3526_MODP_4096_G* *MBEDTLS_DEPRECATED_STRING_CONSTANT*("02")
- #define MBEDTLS_DHM_RFC3526_MODP_2048_P_BIN
- #define MBEDTLS_DHM_RFC3526_MODP_2048_G_BIN { 0x02 }
- #define MBEDTLS_DHM_RFC3526_MODP_3072_P_BIN
- #define MBEDTLS_DHM_RFC3526_MODP_3072_G_BIN { 0x02 }
- #define MBEDTLS_DHM_RFC3526_MODP_4096_P_BIN
- #define MBEDTLS_DHM_RFC3526_MODP_4096_G_BIN { 0x02 }
- #define MBEDTLS_DHM_RFC7919_FFDHE2048_P_BIN
- #define MBEDTLS_DHM_RFC7919_FFDHE2048_G_BIN { 0x02 }
- #define MBEDTLS_DHM_RFC7919_FFDHE3072_P_BIN
- #define MBEDTLS_DHM_RFC7919_FFDHE3072_G_BIN { 0x02 }
- #define MBEDTLS_DHM_RFC7919_FFDHE4096_P_BIN
- #define MBEDTLS_DHM_RFC7919_FFDHE4096_G_BIN { 0x02 }
- #define MBEDTLS_DHM_RFC7919_FFDHE6144_P_BIN
- #define MBEDTLS_DHM_RFC7919_FFDHE6144_G_BIN { 0x02 }
- #define MBEDTLS_DHM_RFC7919_FFDHE8192_P_BIN
- #define MBEDTLS_DHM_RFC7919_FFDHE8192_G_BIN { 0x02 }

Functions

- void *mbedtls_dhm_init* (*mbedtls_dhm_context* *ctx)

  This function initializes the DHM context.

- int *mbedtls_dhm_read_params* (*mbedtls_dhm_context* *ctx, unsigned char **p, const unsigned char *end)

  This function parses the ServerKeyExchange parameters.

- int *mbedtls_dhm_make_params* (*mbedtls_dhm_context* *ctx, int x_size, unsigned char *output, size_t *olen, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

This function sets up and writes the ServerKeyExchange parameters.

- int *mbedtls_dhm_set_group* (*mbedtls_dhm_context* *ctx, const mbedtls_mpi *P, const mbedtls_mpi *G)

  Set prime modulus and generator.

- int *mbedtls_dhm_read_public* (*mbedtls_dhm_context* *ctx, const unsigned char *input, size_t ilen)

  This function imports the public value G^Y of the peer.

- int *mbedtls_dhm_make_public* (*mbedtls_dhm_context* *ctx, int x_size, unsigned char *output, size_t olen, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  This function creates its own private value X and exports G^X .

- int *mbedtls_dhm_calc_secret* (*mbedtls_dhm_context* *ctx, unsigned char *output, size_t output_size, size_t *olen, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  This function derives and exports the shared secret (G^Y)^X mod P .

- void *mbedtls_dhm_free* (*mbedtls_dhm_context* *ctx)

  This function frees and clears the components of a DHM key.

- int *mbedtls_dhm_self_test* (int verbose)

  The DMH checkup routine.

Detailed description

Diffie-Hellman-Merkle key exchange.

RFC-3526: More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE) defines a number of standardized Diffie-Hellman groups for IKE.

RFC-5114: Additional Diffie-Hellman Groups for Use with IETF Standards defines a number of standardized Diffie-Hellman groups that can be used.

Warning

- The security of the DHM key exchange relies on the proper choice of prime modulus - optimally, it should be a safe prime. The usage of non-safe primes both decreases the difficulty of the underlying discrete logarithm problem and can lead to small subgroup attacks leaking private exponent bits when invalid public keys are used and not detected. This is especially relevant if the same DHM parameters are reused for multiple key exchanges as in static DHM, while the criticality of small-subgroup attacks is lower for ephemeral DHM.
- For performance reasons, the code does neither perform primality nor safe primality tests, nor the expensive checks for invalid subgroups. Moreover, even if these were performed, non-standardized primes cannot be trusted because of the possibility of backdoors that cannot be effectively checked for.
- Diffie-Hellman-Merkle is therefore a security risk when not using standardized primes generated using a trustworthy ("nothing up my sleeve") method, such as the RFC 3526 or RFC 7919 primes. In the TLS protocol, DH parameters need to be negotiated, so using the default primes systematically is not always an option. If possible, use Elliptic Curve Diffie-Hellman (ECDH), which has better performance, and for which the TLS protocol mandates the use of standard parameters.

Macro definition documentation

#define MBEDTLS_DEPRECATED_STRING_CONSTANT(VAL)   VAL

RFC 3526, RFC 5114 and RFC 7919 standardize a number of Diffie-Hellman groups, some of which are included here for use within the SSL/TLS module and the user's convenience when configuring the Diffie-Hellman parameters by hand through mbedtls_ssl_conf_dh_param .

The following lists the source of the above groups in the standards:

- RFC 5114 section 2.2: 2048-bit MODP Group with 224-bit Prime Order Subgroup
- RFC 3526 section 3: 2048-bit MODP Group
- RFC 3526 section 4: 3072-bit MODP Group
- RFC 3526 section 5: 4096-bit MODP Group
- RFC 7919 section A.1: ffdhe2048
- RFC 7919 section A.2: ffdhe3072
- RFC 7919 section A.3: ffdhe4096
- RFC 7919 section A.4: ffdhe6144
- RFC 7919 section A.5: ffdhe8192

The constants with suffix "_p" denote the chosen prime moduli, while the constants with suffix "_g" denote the chosen generator of the associated prime field.

The constants further suffixed with "_bin" are provided in binary format, while all other constants represent null-terminated strings holding the hexadecimal presentation of the respective numbers.

The primes from RFC 3526 and RFC 7919 have been generating by the following trust-worthy procedure:

- Fix N in { 2048, 3072, 4096, 6144, 8192 } and consider the N-bit number the first and last 64 bits are all 1, and the remaining N - 128 bits of which are 0x7ff...ff.
- Add the smallest multiple of the first N - 129 bits of the binary expansion of pi (for RFC 5236) or e (for RFC 7919) to this intermediate bit-string such that the resulting integer is a safe-prime.
- The result is the respective RFC 3526 / 7919 prime, and the corresponding generator is always chosen to be 2 (which is a square for these prime, hence the corresponding subgroup has order (p-1)/2 and avoids leaking a bit in the private exponent).

#define MBEDTLS_DHM_RFC3526_MODP_2048_G   MBEDTLS_DEPRECATED_STRING_CONSTANT("02")

The hexadecimal presentation of the chosen generator of the 2048-bit MODP Group, as defined in RFC-3526: More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE) .

#define MBEDTLS_DHM_RFC3526_MODP_2048_P

```
Value:MBEDTLS_DEPRECATED_STRING_CONSTANT(                             \
        "FFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD1"      \
        "29024E088A67CC74020BBEA63B139B22514A08798E3404DD"      \
        "EF9519B3CD3A431B302B0A6DF25F14374FE1356D6D51C245"      \
        "E485B576625E7EC6F44C42E9A637ED6B0BFF5CB6F406B7ED"      \
        "EE386BFB5A899FA5AE9F24117C4B1FE649286651ECE45B3D"      \
        "C2007CB8A163BF0598DA48361C55D39A69163FA8FD24CF5F"      \
        "83655D23DCA3AD961C62F356208552BB9ED529077096966D"      \
        "670C354E4ABC9804F1746C08CA18217C32905E462E36CE3B"      \
        "E39E772C180E86039B2783A2EC07A28FB5C55DF06F4C52C9"      \
        "DE2BCBF6955817183995497CEA956AE515D2261898FA0510"      \
        "15728E5A8AACAA68FFFFFFFFFFFFFFFF" )
```

The hexadecimal presentation of the prime underlying the 2048-bit MODP Group, as defined in RFC-3526: More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE) .

Deprecated:

> The hex-encoded primes from RFC 3625 are deprecated and superseded by the corresponding macros providing them as binary constants. Their hex-encoded constants are likely to be removed in a future version of the library.

#define MBEDTLS_DHM_RFC3526_MODP_3072_G *MBEDTLS_DEPRECATED_STRING_CONSTANT*("02")

The hexadecimal presentation of the chosen generator of the 3072-bit MODP Group, as defined in RFC-3526: More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE) .

#define MBEDTLS_DHM_RFC3526_MODP_3072_P

```
Value:MBEDTLS_DEPRECATED_STRING_CONSTANT(                         \
        "FFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD1"      \
        "29024E088A67CC74020BBEA63B139B22514A08798E3404DD"      \
        "EF9519B3CD3A431B302B0A6DF25F14374FE1356D6D51C245"      \
        "E485B576625E7EC6F44C42E9A637ED6B0BFF5CB6F406B7ED"      \
        "EE386BFB5A899FA5AE9F24117C4B1FE649286651ECE45B3D"      \
        "C2007CB8A163BF0598DA48361C55D39A69163FA8FD24CF5F"      \
        "83655D23DCA3AD961C62F356208552BB9ED529077096966D"      \
        "670C354E4ABC9804F1746C08CA18217C32905E462E36CE3B"      \
        "E39E772C180E86039B2783A2EC07A28FB5C55DF06F4C52C9"      \
        "DE2BCBF6955817183995497CEA956AE515D2261898FA0510"      \
        "15728E5A8AAAC42DAD33170D04507A33A85521ABDF1CBA64"      \
        "ECFB850458DBEF0A8AEA71575D060C7DB3970F85A6E1E4C7"      \
        "ABF5AE8CDB0933D71E8C94E04A25619DCEE3D2261AD2EE6B"      \
        "F12FFA06D98A0864D87602733EC86A64521F2B18177B200C"      \
        "BBE117577A615D6C770988C0BAD946E208E24FA074E5AB31"      \
        "43DB5BFCE0FD108E4B82D120A93AD2CAFFFFFFFFFFFFFFFF" )
```

The hexadecimal presentation of the prime underlying the 3072-bit MODP Group, as defined in RFC-3072: More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE) .

#define MBEDTLS_DHM_RFC3526_MODP_4096_G *MBEDTLS_DEPRECATED_STRING_CONSTANT*("02")

The hexadecimal presentation of the chosen generator of the 4096-bit MODP Group, as defined in RFC-3526: More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE) .

#define MBEDTLS_DHM_RFC3526_MODP_4096_P

```
Value:MBEDTLS_DEPRECATED_STRING_CONSTANT(                      \
        "FFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD1"   \
        "29024E088A67CC74020BBEA63B139B22514A08798E3404DD"   \
        "EF9519B3CD3A431B302B0A6DF25F14374FE1356D6D51C245"   \
        "E485B576625E7EC6F44C42E9A637ED6B0BFF5CB6F406B7ED"   \
        "EE386BFB5A899FA5AE9F24117C4B1FE649286651ECE45B3D"   \
        "C2007CB8A163BF0598DA48361C55D39A69163FA8FD24CF5F"   \
        "83655D23DCA3AD961C62F356208552BB9ED529077096966D"   \
        "670C354E4ABC9804F1746C08CA18217C32905E462E36CE3B"   \
        "E39E772C180E86039B2783A2EC07A28FB5C55DF06F4C52C9"   \
        "DE2BCBF6955817183995497CEA956AE515D2261898FA0510"   \
        "15728E5A8AAAC42DAD33170D04507A33A85521ABDF1CBA64"   \
        "ECFB850458DBEF0A8AEA71575D060C7DB3970F85A6E1E4C7"   \
        "ABF5AE8CDB0933D71E8C94E04A25619DCEE3D2261AD2EE6B"   \
        "F12FFA06D98A0864D87602733EC86A64521F2B18177B200C"   \
        "BBE117577A615D6C770988C0BAD946E208E24FA074E5AB31"   \
        "43DB5BFCE0FD108E4B82D120A92108011A723C12A787E6D7"   \
        "88719A10BDBA5B2699C327186AF4E23C1A946834B6150BDA"   \
        "2583E9CA2AD44CE8DBBBC2DB04DE8EF92E8EFC141FBECAA6"   \
        "287C59474E6BC05D99B2964FA090C3A2233BA186515BE7ED"   \
        "1F612970CEE2D7AFB81BDD762170481CD0069127D5B05AA9"   \
        "93B4EA988D8FDDC186FFB7DC90A6C08F4DF435C934063199"   \
        "FFFFFFFFFFFFFFFF" )
```

The hexadecimal presentation of the prime underlying the 4096-bit MODP Group, as defined in RFC-3526: More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE) .

#define MBEDTLS_DHM_RFC5114_MODP_2048_G

```
Value:MBEDTLS_DEPRECATED_STRING_CONSTANT(                         \
              "AC4032EF4F2D9AE39DF30B5C8FFDAC506CDEBE7B89998CAF"      \
              "74866A08CFE4FFE3A6824A4E10B9A6F0DD921F01A70C4AFA"      \
              "AB739D7700C29F52C57DB17C620A8652BE5E9001A8D66AD7"      \
              "C17669101999024AF4D027275AC1348BB8A762D0521BC98A"      \
              "E247150422EA1ED409939D54DA7460CDB5F6C6B250717CBE"      \
              "F180EB34118E98D119529A45D6F834566E3025E316A330EF"      \
              "BB77A86F0C1AB15B051AE3D428C8F8ACB70A8137150B8EEB"      \
              "10E183EDD19963DDD9E263E4770589EF6AA21E7F5F2FF381"      \
              "B539CCE3409D13CD566AFBB48D6C019181E1BCFE94B30269"      \
              "EDFE72FE9B6AA4BD7B5A0F1C71CFFF4C19C418E1F6EC0179"      \
              "81BC087F2A7065B384B890D3191F2BFA" )
```

The hexadecimal presentation of the chosen generator of the 2048-bit MODP Group with 224-bit Prime Order Subgroup, as defined in RFC-5114: Additional Diffie-Hellman Groups for Use with IETF Standards .

#define MBEDTLS_DHM_RFC5114_MODP_P

```
Value:MBEDTLS_DEPRECATED_STRING_CONSTANT(                         \
              "AD107E1E9123A9D0D660FAA79559C51FA20D64E5683B9FD1"      \
              "B54B1597B61D0A75E6FA141DF95A56DBAF9A3C407BA1DF15"      \
              "EB3D688A309C180E1DE6B85A1274A0A66D3F8152AD6AC212"      \
              "9037C9EDEFDA4DF8D91E8FEF55B7394B7AD5B7D0B6C12207"      \
              "C9F98D11ED34DBF6C6BA0B2C8BBC27BE6A00E0A0B9C49708"      \
              "B3BF8A317091883681286130BC8985DB1602E714415D9330"      \
              "278273C7DE31EFDC7310F7121FD5A07415987D9ADC0A486D"      \
              "CDF93ACC44328387315D75E198C641A480CD86A1B9E587E8"      \
              "BE60E69CC928B2B9C52172E413042E9B23F10B0E16E79763"      \
              "C9B53DCF4BA80A29E3FB73C16B8E75B97EF363E2FFA31F71"      \
              "CF9DE5384E71B81C0AC4DFFE0C10E64F" )
```

Warning

The origin of the primes in RFC 5114 is not documented and their use therefore constitutes a security risk!

Deprecated:

> The hex-encoded primes from RFC 5114 are deprecated and are likely to be removed in a future version of the library without replacement.

The hexadecimal presentation of the prime underlying the 2048-bit MODP Group with 224-bit Prime Order Subgroup, as defined in RFC-5114: Additional Diffie-Hellman Groups for Use with IETF Standards .

#define MBEDTLS_ERR_DHM_ALLOC_FAILED -0x3400

Allocation of memory failed.

#define MBEDTLS_ERR_DHM_BAD_INPUT_DATA -0x3080

Bad input parameters.

#define MBEDTLS_ERR_DHM_CALC_SECRET_FAILED -0x3300

Calculation of the DHM secret failed.

#define MBEDTLS_ERR_DHM_FILE_IO_ERROR -0x3480

Read or write of file failed.

#define MBEDTLS_ERR_DHM_HW_ACCEL_FAILED -0x3500

DHM hardware accelerator failed.

#define MBEDTLS_ERR_DHM_INVALID_FORMAT -0x3380

The ASN.1 data is not formatted correctly.

#define MBEDTLS_ERR_DHM_MAKE_PARAMS_FAILED -0x3180

Making of the DHM parameters failed.

#define MBEDTLS_ERR_DHM_MAKE_PUBLIC_FAILED -0x3280

Making of the public value failed.

#define MBEDTLS_ERR_DHM_READ_PARAMS_FAILED -0x3100

Reading of the DHM parameters failed.

#define MBEDTLS_ERR_DHM_READ_PUBLIC_FAILED -0x3200

Reading of the public values failed.

#define MBEDTLS_ERR_DHM_SET_GROUP_FAILED -0x3580

Setting the modulus and generator failed.

Function documentation

int mbedtls_dhm_calc_secret (*mbedtls_dhm_context* * ctx, unsigned char * output, size_t output_size, size_t * olen, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

This function derives and exports the shared secret (G^Y)^X mod P .

Parameters:

Parameter	Description
`ctx`	The DHM context.

Parameter	Description
output	The destination buffer.
output_size	The size of the destination buffer. Must be at least the size of ctx->len.
olen	On exit, holds the actual number of Bytes written.
f_rng	The RNG function, for blinding purposes.
p_rng	The RNG parameter.

Returns:

0 on success, or an MBEDTLS_ERR_DHM_XXX error code on failure.

Note

If non-NULL, f_rng is used to blind the input as a countermeasure against timing attacks. Blinding is used only if our secret value X is re-used and omitted otherwise. Therefore, we recommend always passing a non-NULL f_rng argument.

void mbedtls_dhm_free (*mbedtls_dhm_context* * ctx)

This function frees and clears the components of a DHM key.

Parameters:

Parameter	Description
ctx	The DHM context to free and clear.

void mbedtls_dhm_init (*mbedtls_dhm_context* * ctx)

This function initializes the DHM context.

Parameters:

Parameter	Description
ctx	The DHM context to initialize.

int mbedtls_dhm_make_params (*mbedtls_dhm_context* * ctx, int x_size, unsigned char * output, size_t * olen, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

This function sets up and writes the ServerKeyExchange parameters.

Note

The destination buffer must be large enough to hold the reduced binary presentation of the modulus, the generator and the public key, each wrapped with a 2-byte length field. It is the responsibility of the caller to ensure that enough space is available. Refer to mbedtls_mpi_size to computing the byte-size of an MPI.

This function assumes that ctx->P and ctx->G have already been properly set. For that, use *mbedtls_dhm_set_group()* below in conjunction with mbedtls_mpi_read_binary() and mbedtls_mpi_read_string().

Parameters:

Parameter	Description
ctx	The DHM context.

Parameter	Description
`x_size`	The private value size in Bytes.
`olen`	The number of characters written.
`output`	The destination buffer.
`f_rng`	The RNG function.
`p_rng`	The RNG parameter.

Returns:

0 on success, or an MBEDTLS_ERR_DHM_XXX error code on failure.

int mbedtls_dhm_make_public (mbedtls_dhm_context * ctx, int x_size, unsigned char * output, size_t olen, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

This function creates its own private value X and exports G^X .

Note

The destination buffer will always be fully written so as to contain a big-endian presentation of G^X mod P. If it is larger than ctx->len, it will accordingly be padded with zero-bytes in the beginning.

Parameters:

Parameter	Description
`ctx`	The DHM context.
`x_size`	The private value size in Bytes.
`output`	The destination buffer.
`olen`	The length of the destination buffer. Must be at least equal to ctx->len (the size of P).
`f_rng`	The RNG function.
`p_rng`	The RNG parameter.

Returns:

0 on success, or an MBEDTLS_ERR_DHM_XXX error code on failure.

int mbedtls_dhm_read_params (mbedtls_dhm_context * ctx, unsigned char ** p, const unsigned char * end)

This function parses the ServerKeyExchange parameters.

Parameters:

Parameter	Description
`ctx`	The DHM context.
`p`	On input, *p must be the start of the input buffer. On output, *p is updated to point to the end of the data that has been read. On success, this is the first byte past the end of the ServerKeyExchange parameters. On error, this is the point at which an error has been detected, which is usually not useful except to debug failures.
`end`	The end of the input buffer.

Returns:

0 on success, or an MBEDTLS_ERR_DHM_XXX error code on failure.

int mbedtls_dhm_read_public (*mbedtls_dhm_context* * ctx, const unsigned char * input, size_t ilen)

This function imports the public value G^Y of the peer.

Parameters:

Parameter	Description
`ctx`	The DHM context.
`input`	The input buffer.
`ilen`	The size of the input buffer.

Returns:

0 on success, or an MBEDTLS_ERR_DHM_XXX error code on failure.

int mbedtls_dhm_self_test (int verbose)

The DMH checkup routine.

Returns:

0 on success, or 1 on failure.

int mbedtls_dhm_set_group (*mbedtls_dhm_context* * ctx, const mbedtls_mpi * P, const mbedtls_mpi * G)

Set prime modulus and generator.

Note

This function can be used to set P, G in preparation for mbedtls_dhm_make_params .

Parameters:

Parameter	Description
`ctx`	The DHM context.
`P`	The MPI holding DHM prime modulus.
`G`	The MPI holding DHM generator.

Returns:

0 if successful, or an MBEDTLS_ERR_DHM_XXX error code on failure.

1.7.40 ecdh.h File Reference

This file contains the Mbed TLS Elliptic Curve Diffie-Hellman (ECDH) protocol APIs.

#include "ecp.h"

Data structures

- struct *mbedtls_ecdh_context*

The ECDH context structure. Enumerations

- enum *mbedtls_ecdh_side* { MBEDTLS_ECDH_OURS, MBEDTLS_ECDH_THEIRS }

Functions

- int *mbedtls_ecdh_gen_public* (*mbedtls_ecp_group* *grp, mbedtls_mpi *d, *mbedtls_ecp_point* *Q, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  This function generates an ECDH keypair on an elliptic curve.

- int *mbedtls_ecdh_compute_shared* (*mbedtls_ecp_group* *grp, mbedtls_mpi *z, const *mbedtls_ecp_point* *Q, const mbedtls_mpi *d, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  This function computes the shared secret.

- void *mbedtls_ecdh_init* (*mbedtls_ecdh_context* *ctx)

  This function initializes an ECDH context.

- void *mbedtls_ecdh_free* (*mbedtls_ecdh_context* *ctx)

  This function frees a context.

- int *mbedtls_ecdh_make_params* (*mbedtls_ecdh_context* *ctx, size_t *olen, unsigned char *buf, size_t blen, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  This function generates a public key and a TLS ServerKeyExchange payload.

- int *mbedtls_ecdh_read_params* (*mbedtls_ecdh_context* *ctx, const unsigned char **buf, const unsigned char *end)

  This function parses and processes a TLS ServerKeyExhange payload.

- int *mbedtls_ecdh_get_params* (*mbedtls_ecdh_context* *ctx, const *mbedtls_ecp_keypair* *key, *mbedtls_ecdh_side* side)

  This function sets up an ECDH context from an EC key.

- int *mbedtls_ecdh_make_public* (*mbedtls_ecdh_context* *ctx, size_t *olen, unsigned char *buf, size_t blen, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  This function generates a public key and a TLS ClientKeyExchange payload.

- int *mbedtls_ecdh_read_public* (*mbedtls_ecdh_context* *ctx, const unsigned char *buf, size_t blen)

  This function parses and processes a TLS ClientKeyExchange payload.

- int *mbedtls_ecdh_calc_secret* (*mbedtls_ecdh_context* *ctx, size_t *olen, unsigned char *buf, size_t blen, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  This function derives and exports the shared secret.

Detailed description

The Elliptic Curve Diffie-Hellman (ECDH) protocol APIs.

ECDH is an anonymous key agreement protocol allowing two parties to establish a shared secret over an insecure channel. Each party must have an elliptic-curve public–private key pair.

For more information, see NIST SP 800-56A Rev. 2: Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography .

Enumeration type documentation

enum mbedtls_ecdh_side

Defines the source of the imported EC key:

- Our key.
- The key of the peer.

Function documentation

int mbedtls_ecdh_calc_secret (mbedtls_ecdh_context * ctx, size_t * olen, unsigned char * buf, size_t blen, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

This function derives and exports the shared secret.

This is the last function used by both TLS client and servers.

Note

If f_rng is not NULL, it is used to implement countermeasures against potential elaborate timing attacks. For more information, see *mbedtls_ecp_mul()*.

Parameters:

Parameter	Description
`ctx`	The ECDH context.
`olen`	The number of Bytes written.
`buf`	The destination buffer.
`blen`	The length of the destination buffer.
`f_rng`	The RNG function.
`p_rng`	The RNG parameter.

Returns:

0 on success, or an MBEDTLS_ERR_ECP_XXX error code on failure.

See also:

ecp.h

int mbedtls_ecdh_compute_shared (mbedtls_ecp_group * grp, mbedtls_mpi * z, const mbedtls_ecp_point * Q, const mbedtls_mpi * d, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

This function computes the shared secret.

This function performs the second of two core computations implemented during the ECDH key exchange. The first core computation is performed by *mbedtls_ecdh_gen_public()*.

Parameters:

Parameter	Description
`grp`	The ECP group.
`z`	The destination MPI (shared secret).
`Q`	The public key from another party.

Parameter	Description
`d`	Our secret exponent (private key).
`f_rng`	The RNG function.
`p_rng`	The RNG parameter.

Returns:

0 on success, or an MBEDTLS_ERR_ECP_XXX or MBEDTLS_MPI_XXX error code on failure.

See also:

ecp.h

Note

If f_rng is not NULL, it is used to implement countermeasures against potential elaborate timing attacks. For more information, see *mbedtls_ecp_mul()*.

void mbedtls_ecdh_free (*mbedtls_ecdh_context* * ctx)

This function frees a context.

Parameters:

Parameter	Description
`ctx`	The context to free.

int mbedtls_ecdh_gen_public (*mbedtls_ecp_group* * grp, mbedtls_mpi * d, *mbedtls_ecp_point* * Q, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

This function generates an ECDH keypair on an elliptic curve.

This function performs the first of two core computations implemented during the ECDH key exchange. The second core computation is performed by *mbedtls_ecdh_compute_shared()*.

Parameters:

Parameter	Description
`grp`	The ECP group.
`d`	The destination MPI (private key).
`Q`	The destination point (public key).
`f_rng`	The RNG function.
`p_rng`	The RNG parameter.

Returns:

0 on success, or an MBEDTLS_ERR_ECP_XXX or MBEDTLS_MPI_XXX error code on failure.

See also:

ecp.h

int mbedtls_ecdh_get_params (mbedtls_ecdh_context * ctx, const mbedtls_ecp_keypair * key, mbedtls_ecdh_side side)

This function sets up an ECDH context from an EC key.

It is used by clients and servers in place of the ServerKeyEchange for static ECDH, and imports ECDH parameters from the EC key information of a certificate.

Parameters:

Parameter	Description
`ctx`	The ECDH context to set up.
`key`	The EC key to use.
`side`	Defines the source of the key: • 1: Our key. • 0: The key of the peer.

Returns:

0 on success, or an MBEDTLS_ERR_ECP_XXX error code on failure.

See also:

ecp.h

void mbedtls_ecdh_init (mbedtls_ecdh_context * ctx)

This function initializes an ECDH context.

Parameters:

Parameter	Description
`ctx`	The ECDH context to initialize.

int mbedtls_ecdh_make_params (mbedtls_ecdh_context * ctx, size_t * olen, unsigned char * buf, size_t blen, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

This function generates a public key and a TLS ServerKeyExchange payload.

This is the first function used by a TLS server for ECDHE cipher suites.

Note

This function assumes that the ECP group (grp) of the ctx context has already been properly set, for example, using *mbedtls_ecp_group_load()*.

Parameters:

Parameter	Description
`ctx`	The ECDH context.
`olen`	The number of characters written.
`buf`	The destination buffer.
`blen`	The length of the destination buffer.
`f_rng`	The RNG function.
`p_rng`	The RNG parameter.

Returns:

0 on success, or an MBEDTLS_ERR_ECP_XXX error code on failure.

See also:

ecp.h

int mbedtls_ecdh_make_public (*mbedtls_ecdh_context* * ctx, size_t * olen, unsigned char * buf, size_t blen, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

This function generates a public key and a TLS ClientKeyExchange payload.

This is the second function used by a TLS client for ECDH(E) ciphersuites.

Parameters:

Parameter	Description
`ctx`	The ECDH context.
`olen`	The number of Bytes written.
`buf`	The destination buffer.
`blen`	The size of the destination buffer.
`f_rng`	The RNG function.
`p_rng`	The RNG parameter.

Returns:

0 on success, or an MBEDTLS_ERR_ECP_XXX error code on failure.

See also:

ecp.h

int mbedtls_ecdh_read_params (*mbedtls_ecdh_context* * ctx, const unsigned char ** buf, const unsigned char * end)

This function parses and processes a TLS ServerKeyExhange payload.

This is the first function used by a TLS client for ECDHE ciphersuites.

Parameters:

Parameter	Description
`ctx`	The ECDH context.
`buf`	The pointer to the start of the input buffer.
`end`	The address for one Byte past the end of the buffer.

Returns:

0 on success, or an MBEDTLS_ERR_ECP_XXX error code on failure.

See also:

ecp.h

int mbedtls_ecdh_read_public (mbedtls_ecdh_context * ctx, const unsigned char * buf, size_t blen)

This function parses and processes a TLS ClientKeyExchange payload.

This is the second function used by a TLS server for ECDH(E) ciphersuites.

Parameters:

Parameter	Description
`ctx`	The ECDH context.
`buf`	The start of the input buffer.
`blen`	The length of the input buffer.

Returns:

0 on success, or an MBEDTLS_ERR_ECP_XXX error code on failure.

See also:

ecp.h

1.7.41 ecdsa.h File Reference

This file contains the Mbed TLS Elliptic Curve Digital Signature Algorithm (ECDSA) APIs.

#include "ecp.h"

#include "md.h"

Macros

- #define *MBEDTLS_ECDSA_MAX_LEN* (3 + 2 * (3 + MBEDTLS_ECP_MAX_BYTES))

Typedefs

- typedef *mbedtls_ecp_keypair* *mbedtls_ecdsa_context*

  The ECDSA context structure.

Functions

- int *mbedtls_ecdsa_sign* (*mbedtls_ecp_group* *grp, mbedtls_mpi *r, mbedtls_mpi *s, const mbedtls_mpi *d, const unsigned char *buf, size_t blen, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  This function computes the ECDSA signature of a previously-hashed message.

- int *mbedtls_ecdsa_verify* (*mbedtls_ecp_group* *grp, const unsigned char *buf, size_t blen, const *mbedtls_ecp_point* *Q, const mbedtls_mpi *r, const mbedtls_mpi *s)

  This function verifies the ECDSA signature of a previously-hashed message.

- int *mbedtls_ecdsa_write_signature* (*mbedtls_ecdsa_context* *ctx, *mbedtls_md_type_t* md_alg, const unsigned char *hash, size_t hlen, unsigned char *sig, size_t *slen, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  This function computes the ECDSA signature and writes it to a buffer, serialized as defined in RFC-4492: Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) .

- int *mbedtls_ecdsa_read_signature* (*mbedtls_ecdsa_context* *ctx, const unsigned char *hash, size_t hlen, const unsigned char *sig, size_t slen)

  This function reads and verifies an ECDSA signature.
- int *mbedtls_ecdsa_genkey* (*mbedtls_ecdsa_context* *ctx, *mbedtls_ecp_group_id* gid, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  This function generates an ECDSA keypair on the given curve.
- int *mbedtls_ecdsa_from_keypair* (*mbedtls_ecdsa_context* *ctx, const *mbedtls_ecp_keypair* *key)

  This function sets an ECDSA context from an EC key pair.
- void *mbedtls_ecdsa_init* (*mbedtls_ecdsa_context* *ctx)

  This function initializes an ECDSA context.
- void *mbedtls_ecdsa_free* (*mbedtls_ecdsa_context* *ctx)

  This function frees an ECDSA context.

Detailed description

The Elliptic Curve Digital Signature Algorithm (ECDSA).

ECDSA is defined in Standards for Efficient Cryptography Group (SECG): SEC1 Elliptic Curve Cryptography . The use of ECDSA for TLS is defined in RFC-4492: Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) .

Macro definition documentation

#define MBEDTLS_ECDSA_MAX_LEN (3 + 2 * (3 + MBEDTLS_ECP_MAX_BYTES))

The maximal size of an ECDSA signature in Bytes.

Function documentation

void mbedtls_ecdsa_free (*mbedtls_ecdsa_context* * ctx)

This function frees an ECDSA context.

Parameters:

Parameter	Description
`ctx`	The ECDSA context to free.

int mbedtls_ecdsa_from_keypair (*mbedtls_ecdsa_context* * ctx, const *mbedtls_ecp_keypair* * key)

This function sets an ECDSA context from an EC key pair.

Parameters:

Parameter	Description
`ctx`	The ECDSA context to set.
`key`	The EC key to use.

Returns:

0 on success, or an MBEDTLS_ERR_ECP_XXX code on failure.

See also:

ecp.h

int mbedtls_ecdsa_genkey (*mbedtls_ecdsa_context* * ctx, *mbedtls_ecp_group_id* gid, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

This function generates an ECDSA keypair on the given curve.

Parameters:

Parameter	Description
`ctx`	The ECDSA context to store the keypair in.
`gid`	The elliptic curve to use. One of the various MBEDTLS_ECP_DP_XXX macros depending on configuration.
`f_rng`	The RNG function.
`p_rng`	The RNG parameter.

Returns:

0 on success, or an MBEDTLS_ERR_ECP_XXX code on failure.

See also:

ecp.h

void mbedtls_ecdsa_init (*mbedtls_ecdsa_context* * ctx)

This function initializes an ECDSA context.

Parameters:

Parameter	Description
`ctx`	The ECDSA context to initialize.

int mbedtls_ecdsa_read_signature (*mbedtls_ecdsa_context* * ctx, const unsigned char * hash, size_t hlen, const unsigned char * sig, size_t slen)

This function reads and verifies an ECDSA signature.

Note

If the bitlength of the message hash is larger than the bitlength of the group order, then the hash is truncated as defined in Standards for Efficient Cryptography Group (SECG): SEC1 Elliptic Curve Cryptography , section 4.1.4, step 3.

Parameters:

Parameter	Description
`ctx`	The ECDSA context.
`hash`	The message hash.
`hlen`	The size of the hash.
`sig`	The signature to read and verify.
`slen`	The size of sig .

Returns:

0 on success, *MBEDTLS_ERR_ECP_BAD_INPUT_DATA* if signature is invalid, *MBEDTLS_ERR_ECP_SIG_LEN_MISMATCH* if the signature is valid but its actual length is less than siglen , or an MBEDTLS_ERR_ECP_XXX or MBEDTLS_ERR_MPI_XXX error code on failure for any other reason.

See also:

ecp.h

int mbedtls_ecdsa_sign (*mbedtls_ecp_group* * grp, mbedtls_mpi * r, mbedtls_mpi * s, const mbedtls_mpi * d, const unsigned char * buf, size_t blen, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

This function computes the ECDSA signature of a previously-hashed message.

Note

If the bitlength of the message hash is larger than the bitlength of the group order, then the hash is truncated as defined in Standards for Efficient Cryptography Group (SECG): SEC1 Elliptic Curve Cryptography , section 4.1.3, step 5.

Note

The deterministic version is usually preferred.

Parameters:

Parameter	Description
grp	The ECP group.
r	The first output integer.
s	The second output integer.
d	The private signing key.
buf	The message hash.
blen	The length of buf .
f_rng	The RNG function.
p_rng	The RNG parameter.

Returns:

0 on success, or an MBEDTLS_ERR_ECP_XXX or MBEDTLS_MPI_XXX error code on failure.

See also:

ecp.h

int mbedtls_ecdsa_verify (*mbedtls_ecp_group* * grp, const unsigned char * buf, size_t blen, const *mbedtls_ecp_point* * Q, const mbedtls_mpi * r, const mbedtls_mpi * s)

This function verifies the ECDSA signature of a previously-hashed message.

Note

If the bitlength of the message hash is larger than the bitlength of the group order, then the hash is truncated as defined in Standards for Efficient Cryptography Group (SECG): SEC1 Elliptic Curve Cryptography , section 4.1.4, step 3.

Parameters:

Parameter	Description
`grp`	The ECP group.
`buf`	The message hash.
`blen`	The length of buf .
`Q`	The public key to use for verification.
`r`	The first integer of the signature.
`s`	The second integer of the signature.

Returns:

0 on success, *MBEDTLS_ERR_ECP_BAD_INPUT_DATA* if signature is invalid, or an MBEDTLS_ERR_ECP_XXX or MBEDTLS_MPI_XXX error code on failure for any other reason.

See also:

ecp.h

int mbedtls_ecdsa_write_signature (*mbedtls_ecdsa_context* * ctx, *mbedtls_md_type_t* md_alg, const unsigned char * hash, size_t hlen, unsigned char * sig, size_t * slen, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

This function computes the ECDSA signature and writes it to a buffer, serialized as defined in RFC-4492: Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) .

Warning

It is not thread-safe to use the same context in multiple threads.

Note

- The deterministic version is used if MBEDTLS_ECDSA_DETERMINISTIC is defined. For more information, see RFC-6979: Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA) .
- The sig buffer must be at least twice as large as the size of the curve used, plus 9. For example, 73 Bytes if a 256-bit curve is used. A buffer length of *MBEDTLS_ECDSA_MAX_LEN* is always safe.
- If the bitlength of the message hash is larger than the bitlength of the group order, then the hash is truncated as defined in Standards for Efficient Cryptography Group (SECG): SEC1 Elliptic Curve Cryptography , section 4.1.3, step 5.

Parameters:

Parameter	Description
`ctx`	The ECDSA context.
`md_alg`	The message digest that was used to hash the message.
`hash`	The message hash.
`hlen`	The length of the hash.
`sig`	The buffer that holds the signature.

Parameter	Description
`slen`	The length of the signature written.
`f_rng`	The RNG function.
`p_rng`	The RNG parameter.

Returns:

0 on success, or an MBEDTLS_ERR_ECP_XXX , MBEDTLS_ERR_MPI_XXX or MBEDTLS_ERR_ASN1_XXX error code on failure.

See also:

ecp.h

1.7.42 ecp.h File Reference

This file contains the Mbed TLS elliptic curves over GF(p) APIs.

#include "bignum.h"

Data structures

- struct *mbedtls_ecp_curve_info*
- struct *mbedtls_ecp_point*

  ECP point structure (jacobian coordinates)
- struct *mbedtls_ecp_group*

  ECP group structure.
- struct *mbedtls_ecp_keypair*

  ECP key pair structure.

Macros

- #define *MBEDTLS_ERR_ECP_BAD_INPUT_DATA* -0x4F80
- #define *MBEDTLS_ERR_ECP_BUFFER_TOO_SMALL* -0x4F00
- #define *MBEDTLS_ERR_ECP_FEATURE_UNAVAILABLE* -0x4E80
- #define *MBEDTLS_ERR_ECP_VERIFY_FAILED* -0x4E00
- #define *MBEDTLS_ERR_ECP_ALLOC_FAILED* -0x4D80
- #define *MBEDTLS_ERR_ECP_RANDOM_FAILED* -0x4D00
- #define *MBEDTLS_ERR_ECP_INVALID_KEY* -0x4C80
- #define *MBEDTLS_ERR_ECP_SIG_LEN_MISMATCH* -0x4C00
- #define *MBEDTLS_ERR_ECP_HW_ACCEL_FAILED* -0x4B80
- #define *MBEDTLS_ECP_DP_MAX* 12
- #define *MBEDTLS_ECP_PF_UNCOMPRESSED* 0
- #define *MBEDTLS_ECP_PF_COMPRESSED* 1
- #define *MBEDTLS_ECP_TLS_NAMED_CURVE* 3

Module settings

The configuration options you can set for this module are in this section. Either change them in config.h or define them on the compiler command line.

- #define *MBEDTLS_ECP_MAX_BITS* 521
- #define MBEDTLS_ECP_MAX_BYTES ((*MBEDTLS_ECP_MAX_BITS* + 7) / 8)
- #define MBEDTLS_ECP_MAX_PT_LEN (2 * MBEDTLS_ECP_MAX_BYTES + 1)
- #define *MBEDTLS_ECP_WINDOW_SIZE* 6
- #define *MBEDTLS_ECP_FIXED_POINT_OPTIM* 1

Enumerations

- enum *mbedtls_ecp_group_id* { MBEDTLS_ECP_DP_NONE = 0, *MBEDTLS_ECP_DP_SECP192R1*, *MBEDTLS_ECP_DP_SECP224R1*, *MBEDTLS_ECP_DP_SECP256R1*, *MBEDTLS_ECP_DP_SECP384R1*, *MBEDTLS_ECP_DP_SECP521R1*, *MBEDTLS_ECP_DP_BP256R1*, *MBEDTLS_ECP_DP_BP384R1*, *MBEDTLS_ECP_DP_BP512R1*, *MBEDTLS_ECP_DP_CURVE25519*, *MBEDTLS_ECP_DP_SECP192K1*, *MBEDTLS_ECP_DP_SECP224K1*, *MBEDTLS_ECP_DP_SECP256K1* }

Functions

- const *mbedtls_ecp_curve_info* * *mbedtls_ecp_curve_list* (void)

  Get the list of supported curves in order of preferrence (full information)
- const *mbedtls_ecp_group_id* * *mbedtls_ecp_grp_id_list* (void)

  Get the list of supported curves in order of preferrence (grp_id only)
- const *mbedtls_ecp_curve_info* * *mbedtls_ecp_curve_info_from_grp_id* (*mbedtls_ecp_group_id* grp_id)

  Get curve information from an internal group identifier.
- const *mbedtls_ecp_curve_info* * *mbedtls_ecp_curve_info_from_tls_id* (uint16_t tls_id)

  Get curve information from a TLS NamedCurve value.
- const *mbedtls_ecp_curve_info* * *mbedtls_ecp_curve_info_from_name* (const char *name)

  Get curve information from a human-readable name.
- void *mbedtls_ecp_point_init* (*mbedtls_ecp_point* *pt)

  Initialize a point (as zero)
- void *mbedtls_ecp_group_init* (*mbedtls_ecp_group* *grp)

  Initialize a group (to something meaningless)
- void *mbedtls_ecp_keypair_init* (*mbedtls_ecp_keypair* *key)

  Initialize a key pair (as an invalid one)
- void *mbedtls_ecp_point_free* (*mbedtls_ecp_point* *pt)

  Free the components of a point.
- void *mbedtls_ecp_group_free* (*mbedtls_ecp_group* *grp)

  Free the components of an ECP group.
- void *mbedtls_ecp_keypair_free* (*mbedtls_ecp_keypair* *key)

  Free the components of a key pair.

- int mbedtls_ecp_copy (mbedtls_ecp_point *P, const mbedtls_ecp_point *Q)

  Copy the contents of point Q into P.
- int mbedtls_ecp_group_copy (mbedtls_ecp_group *dst, const mbedtls_ecp_group *src)

  Copy the contents of a group object.
- int mbedtls_ecp_set_zero (mbedtls_ecp_point *pt)

  Set a point to zero.
- int mbedtls_ecp_is_zero (mbedtls_ecp_point *pt)

  Tell if a point is zero.
- int mbedtls_ecp_point_cmp (const mbedtls_ecp_point *P, const mbedtls_ecp_point *Q)

  Compare two points.
- int mbedtls_ecp_point_read_string (mbedtls_ecp_point *P, int radix, const char *x, const char *y)

  Import a non-zero point from two ASCII strings.
- int mbedtls_ecp_point_write_binary (const mbedtls_ecp_group *grp, const mbedtls_ecp_point *P, int format, size_t *olen, unsigned char *buf, size_t buflen)

  Export a point into unsigned binary data.
- int mbedtls_ecp_point_read_binary (const mbedtls_ecp_group *grp, mbedtls_ecp_point *P, const unsigned char *buf, size_t ilen)

  Import a point from unsigned binary data.
- int mbedtls_ecp_tls_read_point (const mbedtls_ecp_group *grp, mbedtls_ecp_point *pt, const unsigned char **buf, size_t len)

  Import a point from a TLS ECPoint record.
- int mbedtls_ecp_tls_write_point (const mbedtls_ecp_group *grp, const mbedtls_ecp_point *pt, int format, size_t *olen, unsigned char *buf, size_t blen)

  Export a point as a TLS ECPoint record.
- int mbedtls_ecp_group_load (mbedtls_ecp_group *grp, mbedtls_ecp_group_id id)

  Set a group using well-known domain parameters.
- int mbedtls_ecp_tls_read_group (mbedtls_ecp_group *grp, const unsigned char **buf, size_t len)

  Set a group from a TLS ECParameters record.
- int mbedtls_ecp_tls_write_group (const mbedtls_ecp_group *grp, size_t *olen, unsigned char *buf, size_t blen)

  Write the TLS ECParameters record for a group.
- int mbedtls_ecp_mul (mbedtls_ecp_group *grp, mbedtls_ecp_point *R, const mbedtls_mpi *m, const mbedtls_ecp_point *P, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  Multiplication by an integer: R = m * P (Not thread-safe to use same group in multiple threads)
- int mbedtls_ecp_muladd (mbedtls_ecp_group *grp, mbedtls_ecp_point *R, const mbedtls_mpi *m, const mbedtls_ecp_point *P, const mbedtls_mpi *n, const mbedtls_ecp_point *Q)

Multiplication and addition of two points by integers: R = m * P + n * Q (Not thread-safe to use same group in multiple threads)

- int *mbedtls_ecp_check_pubkey* (const *mbedtls_ecp_group* *grp, const *mbedtls_ecp_point* *pt)

  Check that a point is a valid public key on this curve.

- int *mbedtls_ecp_check_privkey* (const *mbedtls_ecp_group* *grp, const mbedtls_mpi *d)

  Check that an mbedtls_mpi is a valid private key for this curve.

- int *mbedtls_ecp_gen_keypair_base* (*mbedtls_ecp_group* *grp, const *mbedtls_ecp_point* *G, mbedtls_mpi *d, *mbedtls_ecp_point* *Q, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  Generate a keypair with configurable base point.

- int *mbedtls_ecp_gen_keypair* (*mbedtls_ecp_group* *grp, mbedtls_mpi *d, *mbedtls_ecp_point* *Q, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  Generate a keypair.

- int *mbedtls_ecp_gen_key* (*mbedtls_ecp_group_id* grp_id, *mbedtls_ecp_keypair* *key, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  Generate a keypair.

- int *mbedtls_ecp_check_pub_priv* (const *mbedtls_ecp_keypair* *pub, const *mbedtls_ecp_keypair* *prv)

  Check a public-private key pair.

Detailed description

Elliptic curves over GF(p)

Macro definition documentation

#define MBEDTLS_ECP_DP_MAX   12

Number of supported curves (plus one for NONE).

(Montgomery curves excluded for now.)

#define MBEDTLS_ECP_FIXED_POINT_OPTIM   1

Enable fixed-point speed-up

#define MBEDTLS_ECP_MAX_BITS   521

Maximum size of the groups (that is, of N and P)Maximum bit size of groups

#define MBEDTLS_ECP_PF_COMPRESSED   1

Compressed point format

#define MBEDTLS_ECP_PF_UNCOMPRESSED   0

Uncompressed point format

#define MBEDTLS_ECP_TLS_NAMED_CURVE   3

ECCurveType's named_curve

#define MBEDTLS_ECP_WINDOW_SIZE   6

Maximum window size used

#define MBEDTLS_ERR_ECP_ALLOC_FAILED   -0x4D80

Memory allocation failed.

#define MBEDTLS_ERR_ECP_BAD_INPUT_DATA   -0x4F80

Bad input parameters to function.

#define MBEDTLS_ERR_ECP_BUFFER_TOO_SMALL   -0x4F00

The buffer is too small to write to.

#define MBEDTLS_ERR_ECP_FEATURE_UNAVAILABLE   -0x4E80

Requested curve not available.

#define MBEDTLS_ERR_ECP_HW_ACCEL_FAILED   -0x4B80

ECP hardware accelerator failed.

#define MBEDTLS_ERR_ECP_INVALID_KEY   -0x4C80

Invalid private or public key.

#define MBEDTLS_ERR_ECP_RANDOM_FAILED   -0x4D00

Generation of random value, such as (ephemeral) key, failed.

#define MBEDTLS_ERR_ECP_SIG_LEN_MISMATCH   -0x4C00

Signature is valid but shorter than the user-supplied length.

#define MBEDTLS_ERR_ECP_VERIFY_FAILED   -0x4E00

The signature is not valid.

Enumeration type documentation

enum *mbedtls_ecp_group_id*

Domain parameters (curve, subgroup and generator) identifiers.

Only curves over prime fields are supported.

Warning

This library does not support validation of arbitrary domain parameters. Therefore, only well-known domain parameters from trusted sources should be used. See *mbedtls_ecp_group_load()*.

Enumerator:

Enum	Description
MBEDTLS_ECP_DP_SECP192R1	192-bits NIST curve
MBEDTLS_ECP_DP_SECP224R1	224-bits NIST curve
MBEDTLS_ECP_DP_SECP256R1	256-bits NIST curve
MBEDTLS_ECP_DP_SECP384R1	384-bits NIST curve
MBEDTLS_ECP_DP_SECP521R1	521-bits NIST curve

Enum	Description
MBEDTLS_ECP_DP_BP256R1	256-bits Brainpool curve
MBEDTLS_ECP_DP_BP384R1	384-bits Brainpool curve
MBEDTLS_ECP_DP_BP512R1	512-bits Brainpool curve
MBEDTLS_ECP_DP_CURVE25519	Curve25519
MBEDTLS_ECP_DP_SECP192K1	192-bits "Koblitz" curve
MBEDTLS_ECP_DP_SECP224K1	224-bits "Koblitz" curve
MBEDTLS_ECP_DP_SECP256K1	256-bits "Koblitz" curve

Function documentation

int mbedtls_ecp_check_privkey (const mbedtls_ecp_group * grp, const mbedtls_mpi * d)

Check that an mbedtls_mpi is a valid private key for this curve.

Note

Uses bare components rather than an mbedtls_ecp_keypair structure in order to ease use with other structures such as mbedtls_ecdh_context of mbedtls_ecdsa_context.

Parameters:

Parameter	Description
grp	Group used
d	Integer to check

Returns:

0 if point is a valid private key, MBEDTLS_ERR_ECP_INVALID_KEY otherwise.

int mbedtls_ecp_check_pub_priv (const mbedtls_ecp_keypair * pub, const mbedtls_ecp_keypair * prv)

Check a public-private key pair.

Parameters:

Parameter	Description
pub	Keypair structure holding a public key
prv	Keypair structure holding a private (plus public) key

Returns:

0 if successful (keys are valid and match), or MBEDTLS_ERR_ECP_BAD_INPUT_DATA, or a MBEDTLS_ERR_ECP_XXX or MBEDTLS_ERR_MPI_XXX code.

int mbedtls_ecp_check_pubkey (const *mbedtls_ecp_group* * grp, const *mbedtls_ecp_point* * pt)

Check that a point is a valid public key on this curve.

Note

This function only checks the point is non-zero, has valid coordinates and lies on the curve, but not that it is indeed a multiple of G. This is additional check is more expensive, isn't required by standards, and shouldn't be necessary if the group used has a small cofactor. In particular, it is useless for the NIST groups which all have a cofactor of 1.

Uses bare components rather than an *mbedtls_ecp_keypair* structure in order to ease use with other structures such as *mbedtls_ecdh_context* of mbedtls_ecdsa_context.

Parameters:

Parameter	Description
`grp`	Curve/group the point should belong to
`pt`	Point to check

Returns:

0 if point is a valid public key, MBEDTLS_ERR_ECP_INVALID_KEY otherwise.

int mbedtls_ecp_copy (*mbedtls_ecp_point* * P, const *mbedtls_ecp_point* * Q)

Copy the contents of point Q into P.

Parameters:

Parameter	Description
`P`	Destination point
`Q`	Source point

Returns:

0 if successful, MBEDTLS_ERR_MPI_ALLOC_FAILED if memory allocation failed

const *mbedtls_ecp_curve_info mbedtls_ecp_curve_info_from_grp_id (*mbedtls_ecp_group_id* grp_id)**

Get curve information from an internal group identifier.

Parameters:

Parameter	Description
`grp_id`	A MBEDTLS_ECP_DP_XXX value

Returns:

The associated curve information or NULL

const *mbedtls_ecp_curve_info** mbedtls_ecp_curve_info_from_name (const char * name)

Get curve information from a human-readable name.

Parameters:

Parameter	Description
`name`	The name

Returns:

The associated curve information or NULL

const *mbedtls_ecp_curve_info** mbedtls_ecp_curve_info_from_tls_id (uint16_t tls_id)

Get curve information from a TLS NamedCurve value.

Parameters:

Parameter	Description
`tls_id`	A MBEDTLS_ECP_DP_XXX value

Returns:

The associated curve information or NULL

const *mbedtls_ecp_curve_info** mbedtls_ecp_curve_list (void)

Get the list of supported curves in order of preferrence (full information)

Returns:

A statically allocated array, the last entry is 0.

int mbedtls_ecp_gen_key (*mbedtls_ecp_group_id* grp_id, *mbedtls_ecp_keypair* * key, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

Generate a keypair.

Parameters:

Parameter	Description
`grp_id`	ECP group identifier
`key`	Destination keypair
`f_rng`	RNG function
`p_rng`	RNG parameter

Returns:

0 if successful, or a MBEDTLS_ERR_ECP_XXX or MBEDTLS_MPI_XXX error code

int mbedtls_ecp_gen_keypair (*mbedtls_ecp_group* * grp, mbedtls_mpi * d, *mbedtls_ecp_point* * Q, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

Generate a keypair.

Note

Uses bare components rather than an *mbedtls_ecp_keypair* structure in order to ease use with other structures such as *mbedtls_ecdh_context* of mbedtls_ecdsa_context.

Parameters:

Parameter	Description
`grp`	ECP group
`d`	Destination MPI (secret part)
`Q`	Destination point (public part)
`f_rng`	RNG function
`p_rng`	RNG parameter

Returns:

0 if successful, or a MBEDTLS_ERR_ECP_XXX or MBEDTLS_MPI_XXX error code

int mbedtls_ecp_gen_keypair_base (*mbedtls_ecp_group* * grp, const *mbedtls_ecp_point* * G, mbedtls_mpi * d, *mbedtls_ecp_point* * Q, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

Generate a keypair with configurable base point.

Note

Uses bare components rather than an *mbedtls_ecp_keypair* structure in order to ease use with other structures such as *mbedtls_ecdh_context* of mbedtls_ecdsa_context.

Parameters:

Parameter	Description
`grp`	ECP group
`G`	Chosen base point
`d`	Destination MPI (secret part)
`Q`	Destination point (public part)
`f_rng`	RNG function
`p_rng`	RNG parameter

Returns:

0 if successful, or a MBEDTLS_ERR_ECP_XXX or MBEDTLS_MPI_XXX error code

int mbedtls_ecp_group_copy (*mbedtls_ecp_group* * dst, const *mbedtls_ecp_group* * src)

Copy the contents of a group object.

Parameters:

Parameter	Description
`dst`	Destination group
`src`	Source group

Returns:

0 if successful, MBEDTLS_ERR_MPI_ALLOC_FAILED if memory allocation failed

int mbedtls_ecp_group_load (*mbedtls_ecp_group* * grp, *mbedtls_ecp_group_id* id)

Set a group using well-known domain parameters.

Parameters:

Parameter	Description
grp	Destination group
id	Index in the list of well-known domain parameters

Returns:

0 if successful, MBEDTLS_ERR_MPI_XXX if initialization failed MBEDTLS_ERR_ECP_FEATURE_UNAVAILABLE for unkownn groups

Note

Index should be a value of RFC 4492's enum NamedCurve, usually in the form of a MBEDTLS_ECP_DP_XXX macro.

const *mbedtls_ecp_group_id** mbedtls_ecp_grp_id_list (void)

Get the list of supported curves in order of preferrence (grp_id only)

Returns:

A statically allocated array, terminated with MBEDTLS_ECP_DP_NONE.

int mbedtls_ecp_is_zero (*mbedtls_ecp_point* * pt)

Tell if a point is zero.

Parameters:

Parameter	Description
pt	Point to test

Returns:

1 if point is zero, 0 otherwise

int mbedtls_ecp_mul (*mbedtls_ecp_group* * grp, *mbedtls_ecp_point* * R, const mbedtls_mpi * m, const *mbedtls_ecp_point* * P, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng)

Multiplication by an integer: R = m * P (Not thread-safe to use same group in multiple threads)

Note

In order to prevent timing attacks, this function executes the exact same sequence of (base field) operations for any valid m. It avoids any if-branch or array index depending on the value of m.

If f_rng is not NULL, it is used to randomize intermediate results in order to prevent potential timing attacks targeting these results. It is recommended to always provide a non-NULL f_rng (the overhead is negligible).

Parameters:

Parameter	Description
grp	ECP group

Parameter	Description
R	Destination point
m	Integer by which to multiply
P	Point to multiply
f_rng	RNG function (see notes)
p_rng	RNG parameter

Returns:

0 if successful, MBEDTLS_ERR_ECP_INVALID_KEY if m is not a valid privkey or P is not a valid pubkey, MBEDTLS_ERR_MPI_ALLOC_FAILED if memory allocation failed

int mbedtls_ecp_muladd (*mbedtls_ecp_group* * grp, *mbedtls_ecp_point* * R, const mbedtls_mpi * m, const *mbedtls_ecp_point* * P, const mbedtls_mpi * n, const *mbedtls_ecp_point* * Q)

Multiplication and addition of two points by integers: R = m * P + n * Q (Not thread-safe to use same group in multiple threads)

Note

In contrast to *mbedtls_ecp_mul()*, this function does not guarantee a constant execution flow and timing.

Parameters:

Parameter	Description
grp	ECP group
R	Destination point
m	Integer by which to multiply P
P	Point to multiply by m
n	Integer by which to multiply Q
Q	Point to be multiplied by n

Returns:

0 if successful, MBEDTLS_ERR_ECP_INVALID_KEY if m or n is not a valid privkey or P or Q is not a valid pubkey, MBEDTLS_ERR_MPI_ALLOC_FAILED if memory allocation failed

int mbedtls_ecp_point_cmp (const *mbedtls_ecp_point* * P, const *mbedtls_ecp_point* * Q)

Compare two points.

Note

This assumes the points are normalized. Otherwise, they may compare as "not equal" even if they are.

Parameters:

Parameter	Description
`P`	First point to compare
`Q`	Second point to compare

Returns:

0 if the points are equal, MBEDTLS_ERR_ECP_BAD_INPUT_DATA otherwise

int mbedtls_ecp_point_read_binary (const *mbedtls_ecp_group* * grp, *mbedtls_ecp_point* * P, const unsigned char * buf, size_t ilen)

Import a point from unsigned binary data.

Parameters:

Parameter	Description
`grp`	Group to which the point should belong
`P`	Point to import
`buf`	Input buffer
`ilen`	Actual length of input

Returns:

0 if successful, MBEDTLS_ERR_ECP_BAD_INPUT_DATA if input is invalid, MBEDTLS_ERR_MPI_ALLOC_FAILED if memory allocation failed, MBEDTLS_ERR_ECP_FEATURE_UNAVAILABLE if the point format is not implemented.

Note

This function does NOT check that the point actually belongs to the given group, see *mbedtls_ecp_check_pubkey()* for that.

int mbedtls_ecp_point_read_string (*mbedtls_ecp_point* * P, int radix, const char * x, const char * y)

Import a non-zero point from two ASCII strings.

Parameters:

Parameter	Description
`P`	Destination point
`radix`	Input numeric base
`x`	First affine coordinate as a null-terminated string
`y`	Second affine coordinate as a null-terminated string

Returns:

0 if successful, or a MBEDTLS_ERR_MPI_XXX error code

int mbedtls_ecp_point_write_binary (const *mbedtls_ecp_group* * grp, const *mbedtls_ecp_point* * P, int format, size_t * olen, unsigned char * buf, size_t buflen)

Export a point into unsigned binary data.

Parameters:

Parameter	Description
`grp`	Group to which the point should belong
`P`	Point to export
`format`	Point format, should be a MBEDTLS_ECP_PF_XXX macro
`olen`	Length of the actual output
`buf`	Output buffer
`buflen`	Length of the output buffer

Returns:

0 if successful, or MBEDTLS_ERR_ECP_BAD_INPUT_DATA or MBEDTLS_ERR_ECP_BUFFER_TOO_SMALL

int mbedtls_ecp_set_zero (*mbedtls_ecp_point* * pt)

Set a point to zero.

Parameters:

Parameter	Description
`pt`	Destination point

Returns:

0 if successful, MBEDTLS_ERR_MPI_ALLOC_FAILED if memory allocation failed

int mbedtls_ecp_tls_read_group (*mbedtls_ecp_group* * grp, const unsigned char ** buf, size_t len)

Set a group from a TLS ECParameters record.

Note

buf is updated to point right after ECParameters on exit

Parameters:

Parameter	Description
`grp`	Destination group
`buf`	&(Start of input buffer)
`len`	Buffer length

Returns:

0 if successful, MBEDTLS_ERR_MPI_XXX if initialization failed
MBEDTLS_ERR_ECP_BAD_INPUT_DATA if input is invalid

int mbedtls_ecp_tls_read_point (const *mbedtls_ecp_group* * grp, *mbedtls_ecp_point* * pt, const unsigned char ** buf, size_t len)

Import a point from a TLS ECPoint record.

Note

buf is updated to point right after the ECPoint on exit

Parameters:

Parameter	Description
`grp`	ECP group used
`pt`	Destination point
`buf`	$(Start of input buffer)
`len`	Buffer length

Returns:

0 if successful, MBEDTLS_ERR_MPI_XXX if initialization failed
MBEDTLS_ERR_ECP_BAD_INPUT_DATA if input is invalid

int mbedtls_ecp_tls_write_group (const *mbedtls_ecp_group* * grp, size_t * olen, unsigned char * buf, size_t blen)

Write the TLS ECParameters record for a group.

Parameters:

Parameter	Description
`grp`	ECP group used
`olen`	Number of bytes actually written
`buf`	Buffer to write to
`blen`	Buffer length

Returns:

0 if successful, or MBEDTLS_ERR_ECP_BUFFER_TOO_SMALL

int mbedtls_ecp_tls_write_point (const *mbedtls_ecp_group* * grp, const *mbedtls_ecp_point* * pt, int format, size_t * olen, unsigned char * buf, size_t blen)

Export a point as a TLS ECPoint record.

Parameters:

Parameter	Description
`grp`	ECP group used
`pt`	Point to export
`format`	Export format
`olen`	length of data written
`buf`	Buffer to write to
`blen`	Buffer length

Returns:

0 if successful, or MBEDTLS_ERR_ECP_BAD_INPUT_DATA or MBEDTLS_ERR_ECP_BUFFER_TOO_SMALL

1.7.43 gcm.h File Reference

This file contains the Mbed TLS Galois/Counter Mode (GCM) APIs.

Galois/Counter Mode (GCM) for 128-bit block ciphers, as defined in D. McGrew, J. Viega, The Galois/Counter Mode of Operation (GCM), Natl. Inst. Stand. Technol.

#include "cipher.h"

#include <stdint.h>

Data structures

- struct *mbedtls_gcm_context*

The GCM context structure. Macros

- #define *MBEDTLS_GCM_ENCRYPT* 1
- #define *MBEDTLS_GCM_DECRYPT* 0
- #define *MBEDTLS_ERR_GCM_AUTH_FAILED* -0x0012
- #define *MBEDTLS_ERR_GCM_HW_ACCEL_FAILED* -0x0013
- #define *MBEDTLS_ERR_GCM_BAD_INPUT* -0x0014

Functions

- void *mbedtls_gcm_init* (*mbedtls_gcm_context* *ctx)

  This function initializes the specified GCM context, to make references valid, and prepares the context for *mbedtls_gcm_setkey()* or *mbedtls_gcm_free()*.

- int *mbedtls_gcm_setkey* (*mbedtls_gcm_context* *ctx, *mbedtls_cipher_id_t* cipher, const unsigned char *key, unsigned int keybits)

  This function associates a GCM context with a cipher algorithm and a key.

- int *mbedtls_gcm_crypt_and_tag* (*mbedtls_gcm_context* *ctx, int mode, size_t length, const unsigned char *iv, size_t iv_len, const unsigned char *add, size_t add_len, const unsigned char *input, unsigned char *output, size_t tag_len, unsigned char *tag)

  This function performs GCM encryption or decryption of a buffer.

- int *mbedtls_gcm_auth_decrypt* (*mbedtls_gcm_context* *ctx, size_t length, const unsigned char *iv, size_t iv_len, const unsigned char *add, size_t add_len, const unsigned char *tag, size_t tag_len, const unsigned char *input, unsigned char *output)

  This function performs a GCM authenticated decryption of a buffer.

- int *mbedtls_gcm_starts* (*mbedtls_gcm_context* *ctx, int mode, const unsigned char *iv, size_t iv_len, const unsigned char *add, size_t add_len)

  This function starts a GCM encryption or decryption operation.

- int *mbedtls_gcm_update* (*mbedtls_gcm_context* *ctx, size_t length, const unsigned char *input, unsigned char *output)

  This function feeds an input buffer into an ongoing GCM encryption or decryption operation.

- int *mbedtls_gcm_finish* (*mbedtls_gcm_context* *ctx, unsigned char *tag, size_t tag_len)

  This function finishes the GCM operation and generates the authentication tag.

- void *mbedtls_gcm_free* (*mbedtls_gcm_context* *ctx)

This function clears a GCM context and the underlying cipher sub-context.

- int *mbedtls_gcm_self_test* (int verbose)

  The GCM checkup routine.

Detailed description

Galois/Counter Mode (GCM) for 128-bit block ciphers, as defined in D. McGrew, J. Viega, The Galois/Counter Mode of Operation (GCM), Natl. Inst. Stand. Technol.

For more information on GCM, see NIST SP 800-38D: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC .

Macro definition documentation

#define MBEDTLS_ERR_GCM_AUTH_FAILED -0x0012

Authenticated decryption failed.

#define MBEDTLS_ERR_GCM_BAD_INPUT -0x0014

Bad input parameters to function.

#define MBEDTLS_ERR_GCM_HW_ACCEL_FAILED -0x0013

GCM hardware accelerator failed.

#define MBEDTLS_GCM_DECRYPT 0

GCM decrypt operation.

#define MBEDTLS_GCM_ENCRYPT 1

GCM encrypt operation.

Function documentation

int mbedtls_gcm_auth_decrypt (*mbedtls_gcm_context* * ctx, size_t length, const unsigned char * iv, size_t iv_len, const unsigned char * add, size_t add_len, const unsigned char * tag, size_t tag_len, const unsigned char * input, unsigned char * output)

This function performs a GCM authenticated decryption of a buffer.

Note

For decryption, the output buffer cannot be the same as input buffer. If the buffers overlap, the output buffer must trail at least 8 Bytes behind the input buffer.

Parameters:

Parameter	Description
`ctx`	The GCM context.
`length`	The length of the input data. This must be a multiple of 16 except in the last call before *mbedtls_gcm_finish()*.
`iv`	The initialization vector.
`iv_len`	The length of the IV.
`add`	The buffer holding the additional data.
`add_len`	The length of the additional data.

Parameter	Description
`tag`	The buffer holding the tag.
`tag_len`	The length of the tag.
`input`	The buffer holding the input data.
`output`	The buffer for holding the output data.

Returns:

0 if successful and authenticated, or *MBEDTLS_ERR_GCM_AUTH_FAILED* if tag does not match.

int mbedtls_gcm_crypt_and_tag (*mbedtls_gcm_context* * ctx, int mode, size_t length, const unsigned char * iv, size_t iv_len, const unsigned char * add, size_t add_len, const unsigned char * input, unsigned char * output, size_t tag_len, unsigned char * tag)

This function performs GCM encryption or decryption of a buffer.

Note

For encryption, the output buffer can be the same as the input buffer. For decryption, the output buffer cannot be the same as input buffer. If the buffers overlap, the output buffer must trail at least 8 Bytes behind the input buffer.

Parameters:

Parameter	Description
`ctx`	The GCM context to use for encryption or decryption.
`mode`	The operation to perform: *MBEDTLS_GCM_ENCRYPT* or *MBEDTLS_GCM_DECRYPT*.
`length`	The length of the input data. This must be a multiple of 16 except in the last call before *mbedtls_gcm_finish()*.
`iv`	The initialization vector.
`iv_len`	The length of the IV.
`add`	The buffer holding the additional data.
`add_len`	The length of the additional data.
`input`	The buffer holding the input data.
`output`	The buffer for holding the output data.
`tag_len`	The length of the tag to generate.
`tag`	The buffer for holding the tag.

Returns:

0 on success.

int mbedtls_gcm_finish (*mbedtls_gcm_context* * ctx, unsigned char * tag, size_t tag_len)

This function finishes the GCM operation and generates the authentication tag.

It wraps up the GCM stream, and generates the tag. The tag can have a maximum length of 16 Bytes.

Parameters:

Parameter	Description
`ctx`	The GCM context.
`tag`	The buffer for holding the tag.
`tag_len`	The length of the tag to generate. Must be at least four.

Returns:

0 on success, or *MBEDTLS_ERR_GCM_BAD_INPUT* on failure.

void mbedtls_gcm_free (*mbedtls_gcm_context* * ctx)

This function clears a GCM context and the underlying cipher sub-context.

Parameters:

Parameter	Description
`ctx`	The GCM context to clear.

void mbedtls_gcm_init (*mbedtls_gcm_context* * ctx)

This function initializes the specified GCM context, to make references valid, and prepares the context for *mbedtls_gcm_setkey()* or *mbedtls_gcm_free()*.

The function does not bind the GCM context to a particular cipher, nor set the key. For this purpose, use *mbedtls_gcm_setkey()*.

Parameters:

Parameter	Description
`ctx`	The GCM context to initialize.

int mbedtls_gcm_self_test (int verbose)

The GCM checkup routine.

Returns:

0 on success, or 1 on failure.

int mbedtls_gcm_setkey (*mbedtls_gcm_context* * ctx, *mbedtls_cipher_id_t* cipher, const unsigned char * key, unsigned int keybits)

This function associates a GCM context with a cipher algorithm and a key.

Parameters:

Parameter	Description
`ctx`	The GCM context to initialize.
`cipher`	The 128-bit block cipher to use.
`key`	The encryption key.
`keybits`	The key size in bits. Valid options are: • 128 bits • 192 bits • 256 bits

Returns:

0 on success, or a cipher specific error code.

int mbedtls_gcm_starts (*mbedtls_gcm_context* * ctx, int mode, const unsigned char * iv, size_t iv_len, const unsigned char * add, size_t add_len)

This function starts a GCM encryption or decryption operation.

Parameters:

Parameter	Description
`ctx`	The GCM context.
`mode`	The operation to perform: *MBEDTLS_GCM_ENCRYPT* or *MBEDTLS_GCM_DECRYPT*.
`iv`	The initialization vector.
`iv_len`	The length of the IV.
`add`	The buffer holding the additional data, or NULL if add_len is 0.
`add_len`	The length of the additional data. If 0, add is NULL.

Returns:

0 on success.

int mbedtls_gcm_update (*mbedtls_gcm_context* * ctx, size_t length, const unsigned char * input, unsigned char * output)

This function feeds an input buffer into an ongoing GCM encryption or decryption operation.

` The function expects input to be a multiple of 16 Bytes. Only the last call before calling *mbedtls_gcm_finish()* can be less than 16 Bytes.

Note

For decryption, the output buffer cannot be the same as input buffer. If the buffers overlap, the output buffer must trail at least 8 Bytes behind the input buffer.

Parameters:

Parameter	Description
`ctx`	The GCM context.
`length`	The length of the input data. This must be a multiple of 16 except in the last call before *mbedtls_gcm_finish()*.
`input`	The buffer holding the input data.
`output`	The buffer for holding the output data.

Returns:

0 on success, or *MBEDTLS_ERR_GCM_BAD_INPUT* on failure.

1.7.44 mbedtls_aes_ext_dma.h File Reference

This file contains all the CryptoCell AES external DMA APIs, their enums and definitions.

#include "cc_aes_defs_proj.h"

#include "cc_pal_types.h"

Functions

- int *mbedtls_aes_ext_dma_init* (unsigned int keybits, int encryptDecryptFlag, *CCAesOperationMode_t* operationMode, size_t data_size)

  This function initializes the external DMA Control. It configures the AES mode, the direction(encryption or decryption), and the data size.

- int *mbedtls_aes_ext_dma_set_key* (const unsigned char *key, unsigned int keybits)

  This function configures the key.

- int *mbedtls_aes_ext_dma_set_iv* (*CCAesOperationMode_t* operationMode, unsigned char *iv, unsigned int iv_size)

  This function configures the IV.

- int *mbedtls_aes_ext_dma_finish* (*CCAesOperationMode_t* operationMode, unsigned char *iv, unsigned int iv_size)

  This function returns the IV after an AES CMAC or a CBCMAC operation.

Detailed description

This file contains all the CryptoCell AES external DMA APIs, their enums and definitions.

1.7.45 mbedtls_cc_aes_crypt_additional.h File Reference

This file contains all CryptoCell AES APIs that are currently not supported by Mbed TLS.

#include "mbedtls/aes.h"

Functions

- int *mbedtls_aes_crypt_ofb* (*mbedtls_aes_context* *ctx, size_t length, size_t *nc_off, unsigned char nonce_counter[16], unsigned char stream_block[16], const unsigned char *input, unsigned char *output)

  This function encrypts or decrypts AES-OFB buffer.

Detailed description

This file contains all CryptoCell AES APIs that are currently not supported by Mbed TLS.

1.7.46 mbedtls_cc_aes_key_wrap.h File Reference

This file contains all of the CryptoCell key-wrapping APIs, their enums and definitions.

#include "cc_pal_types.h"

#include "cc_error.h"

Macros

- #define CC_AES_KEYWRAP_SEMIBLOCK_SIZE_BYTES (CC_AES_BLOCK_SIZE_IN_BYTES >> 1)
- #define CC_AES_KEYWRAP_SEMIBLOCK_SIZE_WORDS (CC_AES_KEYWRAP_SEMIBLOCK_SIZE_BYTES >> 2)
- #define CC_AES_KEYWRAP_SEMIBLOCK_TO_BYTES_SHFT 3
- #define CC_AES_KEYWRAP_MAX_PAD_LEN 7
- #define CC_AES_KEYWRAP_ICV1 {0xA6A6A6A6, 0xA6A6A6A6}
- #define CC_AES_KEYWRAP_ICV2 {0xA65959A6, 0x00000000}

Typedefs

- typedef enum keyWrapMode mbedtls_keywrap_mode_t

Enumerations

- enum keyWrapMode { CC_AES_KEYWRAP_KW_MODE = 0, CC_AES_KEYWRAP_KWP_MODE = 1, CC_AES_KEYWRAP_NUM_OF_MODES = 2, CC_AES_KEYWRAP_RESERVE32B = INT32_MAX }

Functions

- CCError_t mbedtls_aes_key_wrap (mbedtls_keywrap_mode_t keyWrapFlag, uint8_t *keyBuf, size_t keySize, uint8_t *pPlainText, size_t plainTextSize, uint8_t *pCipherText, size_t *pCipherTextSize)

  This is the AES wrapping or encryption function.

- CCError_t mbedtls_aes_key_unwrap (mbedtls_keywrap_mode_t keyWrapFlag, uint8_t *keyBuf, size_t keySize, uint8_t *pCipherText, size_t cipherTextSize, uint8_t *pPlainText, size_t *pPlainTextSize)

  This is the AES unwrapping or decryption function.

Detailed description

This file contains all of the CryptoCell key-wrapping APIs, their enums and definitions.

The APIs support AES key wrapping as defined in NIST SP 800-38F: Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping .

1.7.47 mbedtls_cc_aes_key_wrap_error.h File Reference

This file contains the error definitions of the CryptoCell AES key-wrapping APIs.

#include "cc_error.h"

Macros

- #define CC_AES_KEYWRAP_DATA_IN_POINTER_INVALID_ERROR (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0x00UL)
- #define CC_AES_KEYWRAP_DATA_OUT_POINTER_INVALID_ERROR (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0x01UL)
- #define CC_AES_KEYWRAP_INVALID_KEY_POINTER_ERROR (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0x02UL)
- #define CC_AES_KEYWRAP_ILLEGAL_KEY_SIZE_ERROR (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0x03UL)
- #define CC_AES_KEYWRAP_SEMIBLOCKS_NUM_ILLEGAL (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0x04UL)
- #define CC_AES_KEYWRAP_ILLEGAL_PARAMETER_PTR_ERROR (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0x05UL)
- #define CC_AES_KEYWRAP_INVALID_ENCRYPT_MODE_ERROR (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0x06UL)
- #define CC_AES_KEYWRAP_DATA_IN_SIZE_ILLEGAL (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0x07UL)
- #define CC_AES_KEYWRAP_DATA_OUT_SIZE_ILLEGAL (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0x08UL)
- #define CC_AES_KEYWRAP_INVALID_KEYWRAP_MODE_ERROR (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0x09UL)
- #define CC_AES_KEYWRAP_UNWRAP_COMPARISON_ERROR (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0x0AUL)
- #define CC_AES_KEYWRAP_IS_NOT_SUPPORTED (CC_AES_KEYWRAP_MODULE_ERROR_BASE + 0xFFUL)

Detailed description

This file contains the error definitions of the CryptoCell AES key-wrapping APIs.

1.7.48 mbedtls_cc_ccm_star.h File Reference

This file contains the CryptoCell AES-CCM star APIs, their enums and definitions.

#include "cc_pal_types.h"

#include "cc_error.h"

#include "mbedtls/ccm.h"

#include "mbedtls_ccm_common.h"

Functions

- void *mbedtls_ccm_star_init* (*mbedtls_ccm_context* *ctx)

  This function initializes the CCM star context.

- int *mbedtls_ccm_star_setkey* (*mbedtls_ccm_context* *ctx, *mbedtls_cipher_id_t* cipher, const unsigned char *key, unsigned int keybits)

  This function initializes the CCM star context set in the ctx parameter and sets the encryption or decryption key.

- void *mbedtls_ccm_star_free* (*mbedtls_ccm_context* *ctx)

  This function releases and clears the specified CCM star context and underlying cipher sub-context.

- int *mbedtls_ccm_star_encrypt_and_tag* (*mbedtls_ccm_context* *ctx, size_t length, const unsigned char *iv, size_t iv_len, const unsigned char *add, size_t add_len, const unsigned char *input, unsigned char *output, unsigned char *tag, size_t tag_len)

  This function encrypts a buffer using CCM star.

- int *mbedtls_ccm_star_auth_decrypt* (*mbedtls_ccm_context* *ctx, size_t length, const unsigned char *iv, size_t iv_len, const unsigned char *add, size_t add_len, const unsigned char *input, unsigned char *output, const unsigned char *tag, size_t tag_len)

  This function performs a CCM star authenticated decryption of a buffer.

- int *mbedtls_ccm_star_nonce_generate* (unsigned char *src_addr, uint32_t frame_counter, uint8_t size_of_t, unsigned char *nonce_buf)

  This function receives the MAC source address, the frame counter, and the MAC size, and returns the required nonce for AES-CCM*, as defined in IEEE 802.15.4: IEEE Standard for Local and metropolitan area networks— Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs) .

Detailed description

This file contains the CryptoCell AES-CCM star APIs, their enums and definitions.

This API supports AES-CCM*, as defined in IEEE 802.15.4: IEEE Standard for Local and metropolitan area networks— Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs) , with the instantiations defined in section B.3.2, and the nonce defined in section 7.3.2.

1.7.49 mbedtls_cc_chacha.h File Reference

This file contains all of the CryptoCell ChaCha APIs, their enums and definitions.

#include "cc_pal_types.h"

#include "cc_error.h"

Data structures

- struct *mbedtls_chacha_user_context*

The context prototype of the user. Macros

- #define *CC_CHACHA_USER_CTX_SIZE_IN_WORDS* 17
- #define *CC_CHACHA_BLOCK_SIZE_IN_WORDS* 16
- #define *CC_CHACHA_BLOCK_SIZE_IN_BYTES* (*CC_CHACHA_BLOCK_SIZE_IN_WORDS* * sizeof(uint32_t))
- #define *CC_CHACHA_NONCE_MAX_SIZE_IN_WORDS* 3
- #define *CC_CHACHA_NONCE_MAX_SIZE_IN_BYTES* (*CC_CHACHA_NONCE_MAX_SIZE_IN_WORDS* * sizeof(uint32_t))
- #define *CC_CHACHA_KEY_MAX_SIZE_IN_WORDS* 8
- #define *CC_CHACHA_KEY_MAX_SIZE_IN_BYTES* (*CC_CHACHA_KEY_MAX_SIZE_IN_WORDS* * sizeof(uint32_t))

Typedefs

- typedef uint8_t *mbedtls_chacha_nonce*[*CC_CHACHA_NONCE_MAX_SIZE_IN_BYTES*]
- typedef uint8_t *mbedtls_chacha_key*[*CC_CHACHA_KEY_MAX_SIZE_IN_BYTES*]
- typedef struct *mbedtls_chacha_user_context* *mbedtls_chacha_user_context*

  The context prototype of the user.

Enumerations

- enum *mbedtls_chacha_encrypt_mode_t* { *CC_CHACHA_Encrypt* = 0, *CC_CHACHA_Decrypt* = 1, *CC_CHACHA_EncryptNumOfOptions*, *CC_CHACHA_EncryptModeLast* = 0x7FFFFFFF }
- enum *mbedtls_chacha_nonce_size_t* { *CC_CHACHA_Nonce64BitSize* = 0, *CC_CHACHA_Nonce96BitSize* = 1, *CC_CHACHA_NonceSizeNumOfOptions*, *CC_CHACHA_NonceSizeLast* = 0x7FFFFFFF }

Functions

- CIMPORT_C CCError_t *mbedtls_chacha_init* (*mbedtls_chacha_user_context* *pContextID, *mbedtls_chacha_nonce* pNonce, *mbedtls_chacha_nonce_size_t* nonceSize, *mbedtls_chacha_key* pKey, uint32_t initialCounter, *mbedtls_chacha_encrypt_mode_t* EncryptDecryptFlag)

  This function initializes the context for ChaCha-engine operations.

- CIMPORT_C CCError_t *mbedtls_chacha_block* (*mbedtls_chacha_user_context* *pContextID, uint8_t *pDataIn, size_t dataInSize, uint8_t *pDataOut)

  This function processes aligned blocks of the ChaCha engine.

- CIMPORT_C CCError_t *mbedtls_chacha_finish* (*mbedtls_chacha_user_context* *pContextID, uint8_t *pDataIn, size_t dataInSize, uint8_t *pDataOut)

  This function processes the remaining ChaCha data.

- CIMPORT_C CCError_t *mbedtls_chacha_free* (*mbedtls_chacha_user_context* *pContextID)

  This function frees the context used for ChaCha operations.

- CIMPORT_C CCError_t *mbedtls_chacha* (*mbedtls_chacha_nonce* pNonce, *mbedtls_chacha_nonce_size_t* nonceSize, *mbedtls_chacha_key* pKey, uint32_t initialCounter, *mbedtls_chacha_encrypt_mode_t* encryptDecryptFlag, uint8_t *pDataIn, size_t dataInSize, uint8_t *pDataOut)

  This function performs the ChaCha operation in one integrated process.

Detailed description

This file contains all of the CryptoCell ChaCha APIs, their enums and definitions.

1.7.50 mbedtls_cc_chacha_error.h File Reference

This file contains the error definitions of the CryptoCell ChaCha APIs.

#include "cc_error.h"

Macros

- #define CC_CHACHA_INVALID_NONCE_ERROR (CC_CHACHA_MODULE_ERROR_BASE + 0x01UL)
- #define CC_CHACHA_ILLEGAL_KEY_SIZE_ERROR (CC_CHACHA_MODULE_ERROR_BASE + 0x02UL)
- #define CC_CHACHA_INVALID_KEY_POINTER_ERROR (CC_CHACHA_MODULE_ERROR_BASE + 0x03UL)
- #define CC_CHACHA_INVALID_ENCRYPT_MODE_ERROR (CC_CHACHA_MODULE_ERROR_BASE + 0x04UL)
- #define CC_CHACHA_DATA_IN_POINTER_INVALID_ERROR (CC_CHACHA_MODULE_ERROR_BASE + 0x05UL)
- #define CC_CHACHA_DATA_OUT_POINTER_INVALID_ERROR (CC_CHACHA_MODULE_ERROR_BASE + 0x06UL)
- #define CC_CHACHA_INVALID_USER_CONTEXT_POINTER_ERROR (CC_CHACHA_MODULE_ERROR_BASE + 0x07UL)
- #define CC_CHACHA_CTX_SIZES_ERROR (CC_CHACHA_MODULE_ERROR_BASE + 0x08UL)
- #define CC_CHACHA_INVALID_NONCE_PTR_ERROR (CC_CHACHA_MODULE_ERROR_BASE + 0x09UL)
- #define CC_CHACHA_DATA_IN_SIZE_ILLEGAL (CC_CHACHA_MODULE_ERROR_BASE + 0x0AUL)
- #define CC_CHACHA_GENERAL_ERROR (CC_CHACHA_MODULE_ERROR_BASE + 0x0BUL)
- #define CC_CHACHA_IS_NOT_SUPPORTED (CC_CHACHA_MODULE_ERROR_BASE + 0xFFUL)

Detailed description

This file contains the error definitions of the CryptoCell ChaCha APIs.

1.7.51 mbedtls_cc_chacha_poly.h File Reference

This file contains all of the CryptoCell ChaCha-POLY APIs, their enums and definitions.

#include "cc_pal_types.h"

#include "cc_error.h"

#include "mbedtls_cc_chacha.h"

#include "mbedtls_cc_poly.h"

Functions

- CIMPORT_C CCError_t *mbedtls_chacha_poly* (*mbedtls_chacha_nonce* pNonce, *mbedtls_chacha_key* pKey, *mbedtls_chacha_encrypt_mode_t* encryptDecryptFlag, uint8_t *pAddData, size_t addDataSize, uint8_t *pDataIn, size_t dataInSize, uint8_t *pDataOut, *mbedtls_poly_mac* macRes)

  This function performs the ChaCha-POLY encryption and authentication operation.

Detailed description

This file contains all of the CryptoCell ChaCha-POLY APIs, their enums and definitions.

1.7.52 mbedtls_cc_chacha_poly_error.h File Reference

This file contains the errors definitions of the CryptoCell ChaCha-POLY APIs.

#include "cc_error.h"

Macros

- #define *CC_CHACHA_POLY_ADATA_INVALID_ERROR* (*CC_CHACHA_POLY_MODULE_ERROR_BASE* + 0x01UL)
- #define *CC_CHACHA_POLY_DATA_INVALID_ERROR* (*CC_CHACHA_POLY_MODULE_ERROR_BASE* + 0x02UL)
- #define *CC_CHACHA_POLY_ENC_MODE_INVALID_ERROR* (*CC_CHACHA_POLY_MODULE_ERROR_BASE* + 0x03UL)
- #define *CC_CHACHA_POLY_DATA_SIZE_INVALID_ERROR* (*CC_CHACHA_POLY_MODULE_ERROR_BASE* + 0x04UL)
- #define *CC_CHACHA_POLY_GEN_KEY_ERROR* (*CC_CHACHA_POLY_MODULE_ERROR_BASE* + 0x05UL)
- #define *CC_CHACHA_POLY_ENCRYPTION_ERROR* (*CC_CHACHA_POLY_MODULE_ERROR_BASE* + 0x06UL)
- #define *CC_CHACHA_POLY_AUTH_ERROR* (*CC_CHACHA_POLY_MODULE_ERROR_BASE* + 0x07UL)
- #define *CC_CHACHA_POLY_MAC_ERROR* (*CC_CHACHA_POLY_MODULE_ERROR_BASE* + 0x08UL)

Detailed description

This file contains the errors definitions of the CryptoCell ChaCha-POLY APIs.

1.7.53 mbedtls_cc_ecies.h File Reference

This file contains the CryptoCell Elliptic Curve Integrated Encryption Scheme (ECIES) APIs.

#include "cc_ecpki_types.h"

#include "cc_pal_types_plat.h"

#include "cc_kdf.h"

#include "mbedtls_cc_hkdf.h"

#include "mbedtls/ecp.h"

Macros

- #define *MBEDTLS_ECIES_MAX_CIPHER_LEN_BYTES* ((2**CC_ECPKI_MODUL_MAX_LENGTH_IN_WORDS* + 1) * sizeof(int))
- #define *MBEDTLS_ECIES_MIN_BUFF_LEN_BYTES* (sizeof(*CCEciesTempData_t*))
- #define *mbedtls_ecies_kem_encrypt*(pGrp, pRecipPublKey, kdfDerivMode, kdfHashMode, isSingleHashMode, pSecrKey, secrKeySize, pCipherData, pCipherDataSize, pBuff, buffLen, f_rng, p_rng)

  A macro for creating and encrypting a secret key.

Functions

- CCError_t *mbedtls_ecies_kem_encrypt_full* (*mbedtls_ecp_group* *pGrp, *mbedtls_ecp_point* *pRecipUzPublKey, CCKdfDerivFuncMode_t kdfDerivMode, *mbedtls_hkdf_hashmode_t* kdfHashMode, uint32_t isSingleHashMode, *mbedtls_ecp_point* *pExtEphUzPublicKey, mbedtls_mpi *pExtEphUzPrivateKey, uint8_t *pSecrKey, size_t secrKeySize, uint8_t *pCipherData, size_t *pCipherDataSize, void *pBuff, size_t buffLen, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng)

  This function creates and encrypts (encapsulates) the secret key of required size, according to ISO/IEC 18033-2:2006: Information technology – Security techniques – Encryption algorithms – Part 2: Asymmetric ciphers , ECIES-KEM Encryption.

- CCError_t *mbedtls_ecies_kem_decrypt* (*mbedtls_ecp_group* *pGrp, mbedtls_mpi *pRecipUzPrivKey, CCKdfDerivFuncMode_t kdfDerivMode, *mbedtls_hkdf_hashmode_t* kdfHashMode, uint32_t isSingleHashMode, uint8_t *pCipherData, size_t cipherDataSize, uint8_t *pSecrKey, size_t secrKeySize, void *pBuff, size_t buffLen)

  This function decrypts the encapsulated secret key passed by the sender, according to ISO/IEC 18033-2:2006: Information technology – Security techniques – Encryption algorithms – Part 2: Asymmetric ciphers , sec. 10.2.4 - ECIES-KEM Decryption.

Detailed description

This file contains the CryptoCell Elliptic Curve Integrated Encryption Scheme (ECIES) APIs.

1.7.54 mbedtls_cc_hkdf.h File Reference

This file contains the CryptoCell HMAC key-derivation function API.

#include "cc_pal_types.h"

Macros

- #define *CC_HKDF_MAX_HASH_KEY_SIZE_IN_BYTES* 512
- #define *CC_HKDF_MAX_HASH_DIGEST_SIZE_IN_BYTES* *CC_HASH_SHA512_DIGEST_SIZE_IN_BYTES*

Enumerations

- enum *mbedtls_hkdf_hashmode_t* { *CC_HKDF_HASH_SHA1_mode* = 0, *CC_HKDF_HASH_SHA224_mode* = 1, *CC_HKDF_HASH_SHA256_mode* = 2, *CC_HKDF_HASH_SHA384_mode* = 3, *CC_HKDF_HASH_SHA512_mode* = 4, *CC_HKDF_HASH_NumOfModes*, *CC_HKDF_HASH_OpModeLast* = 0x7FFFFFFF }

Functions

- CCError_t *mbedtls_hkdf_key_derivation* (*mbedtls_hkdf_hashmode_t* HKDFhashMode, uint8_t *Salt_ptr, size_t SaltLen, uint8_t *Ikm_ptr, uint32_t IkmLen, uint8_t *Info, uint32_t InfoLen, uint8_t *Okm, uint32_t OkmLen, *CCBool* IsStrongKey)

  mbedtls_hkdf_key_derivation() performs the HMAC-based key derivation, as define by RFC-5869: HMAC-based Extract-and-Expand Key Derivation Function (HKDF) .

Detailed description

This file contains the CryptoCell HMAC key-derivation function API.

This function is as defined in RFC-5869: HMAC-based Extract-and-Expand Key Derivation Function (HKDF) .

1.7.55 mbedtls_cc_hkdf_error.h File Reference

This file contains the error definitions of the CryptoCell HKDF APIs.

#include "cc_error.h"

Macros

- #define *CC_HKDF_INVALID_ARGUMENT_POINTER_ERROR* (*CC_HKDF_MODULE_ERROR_BASE* + 0x0UL)
- #define *CC_HKDF_INVALID_ARGUMENT_SIZE_ERROR* (*CC_HKDF_MODULE_ERROR_BASE* + 0x1UL)
- #define *CC_HKDF_INVALID_ARGUMENT_HASH_MODE_ERROR* (*CC_HKDF_MODULE_ERROR_BASE* + 0x3UL)
- #define *CC_HKDF_IS_NOT_SUPPORTED* (*CC_HKDF_MODULE_ERROR_BASE* + 0xFFUL)

Detailed description

This file contains the error definitions of the CryptoCell HKDF APIs.

1.7.56 mbedtls_cc_mng.h File Reference

This file contains all the CryptoCell Management APIs, their enums and definitions.

#include "cc_pal_types_plat.h"

Data structures

- union *mbedtls_mng_apbcconfig*

Macros

- #define *CC_MNG_LCS_CM* 0x0
- #define *CC_MNG_LCS_DM* 0x1
- #define *CC_MNG_LCS_SEC_ENABLED* 0x5
- #define *CC_MNG_LCS_RMA* 0x7

Typedefs

- typedef union *mbedtls_mng_apbcconfig* *mbedtls_mng_apbcconfig*

Enumerations

- enum *mbedtls_mng_rmastatus* { CC_MNG_NON_RMA = 0, *CC_MNG_PENDING_RMA* = 1, *CC_MNG_ILLEGAL_STATE* = 2, *CC_MNG_RMA* = 3 }
- enum *mbedtls_mng_keytype* { CC_MNG_HUK_KEY = 0, *CC_MNG_RTL_KEY* = 1, *CC_MNG_PROV_KEY* = 2, *CC_MNG_CE_KEY* = 3, *CC_MNG_ICV_PROV_KEY* = 4, *CC_MNG_ICV_CE_KEY* = 5, *CC_MNG_TOTAL_HW_KEYS* = 6, *CC_MNG_END_OF_KEY_TYPE* = 0x7FFFFFFF }

Functions

- int *mbedtls_mng_pending_rma_status_get* (uint32_t *rmaStatus)

  This function reads the OTP word of the OEM flags, and returns the OEM RMA flag status: TRUE or FALSE.

- int *mbedtls_mng_hw_version_get* (uint32_t *partNumber, uint32_t *revision)

  This function verifies and returns the CryptoCell HW version.

- int *mbedtls_mng_cc_sec_mode_set* (CCBool_t isSecAccessMode, CCBool_t isSecModeLock)

  This function sets CryptoCell to Secure mode.

- int *mbedtls_mng_cc_priv_mode_set* (CCBool_t isPrivAccessMode, CCBool_t isPrivModeLock)

  This function sets CryptoCell to Privilege mode.

- int *mbedtls_mng_debug_key_set* (*mbedtls_mng_keytype* keyType, uint32_t *pHwKey, size_t keySize)

  This function sets the shadow register of one of the HW Keys when the device is in CM LCS or DM LCS.

- int *mbedtls_mng_gen_config_get* (uint32_t *pOtpWord)

This function retrieves the general configuration from the OTP. See Arm TrustZone CryptoCell-312 Software Integrators Manual.

- int *mbedtls_mng_oem_key_lock* (CCBool_t kcpLock, CCBool_t kceLock)

  This function locks the usage of either Kcp, Kce, or both during runtime, in either Secure LCS or RMA LCS.

- int *mbedtls_mng_apbc_config_set* (*mbedtls_mng_apbcconfig* apbcConfig)

  This function sets the CryptoCell APB-C into one of the following modes:

- int *mbedtls_mng_apbc_access* (CCBool_t isApbcAccessUsed)

  This function requests usage of or releases the APB-C.

- int *mbedtls_mng_suspend* (uint8_t *pBackupBuffer, size_t backupSize)

  This function is called once the external PMU decides to power-down CryptoCell.

- int *mbedtls_mng_resume* (uint8_t *pBackupBuffer, size_t backupSize)

  This function is called once the external PMU decides to power-up CryptoCell.

Detailed description

This file contains all the CryptoCell Management APIs, their enums and definitions.

The following terms, used throughout this module, are defined in Arm Architecture Reference Manual Armv8:

- Privileged and unprivileged modes.
- Secure and Non-secure modes.

1.7.57 mbedtls_cc_mng_error.h File Reference

This file contains the error definitions of the CryptoCell management APIs.

#include "cc_error.h"

Macros

- #define CC_MNG_ILLEGAL_INPUT_PARAM_ERR (CC_MNG_MODULE_ERROR_BASE + 0x00UL)
- #define CC_MNG_ILLEGAL_OPERATION_ERR (CC_MNG_MODULE_ERROR_BASE + 0x01UL)
- #define CC_MNG_ILLEGAL_PIDR_ERR (CC_MNG_MODULE_ERROR_BASE + 0x02UL)
- #define CC_MNG_ILLEGAL_CIDR_ERR (CC_MNG_MODULE_ERROR_BASE + 0x03UL)
- #define CC_MNG_APB_SECURE_IS_LOCKED_ERR (CC_MNG_MODULE_ERROR_BASE + 0x04UL)
- #define CC_MNG_APB_PRIVILEGE_IS_LOCKED_ERR (CC_MNG_MODULE_ERROR_BASE + 0x05UL)
- #define CC_MNG_APBC_SECURE_IS_LOCKED_ERR (CC_MNG_MODULE_ERROR_BASE + 0x06UL)
- #define CC_MNG_APBC_PRIVILEGE_IS_LOCKED_ERR (CC_MNG_MODULE_ERROR_BASE + 0x07UL)
- #define CC_MNG_APBC_INSTRUCTION_IS_LOCKED_ERR (CC_MNG_MODULE_ERROR_BASE + 0x08UL)
- #define CC_MNG_INVALID_KEY_TYPE_ERROR (CC_MNG_MODULE_ERROR_BASE + 0x09UL)
- #define CC_MNG_ILLEGAL_HUK_SIZE_ERR (CC_MNG_MODULE_ERROR_BASE + 0x0AUL)
- #define CC_MNG_ILLEGAL_HW_KEY_SIZE_ERR (CC_MNG_MODULE_ERROR_BASE + 0x0BUL)
- #define CC_MNG_HW_KEY_IS_LOCKED_ERR (CC_MNG_MODULE_ERROR_BASE + 0x0CUL)
- #define CC_MNG_KCP_IS_LOCKED_ERR (CC_MNG_MODULE_ERROR_BASE + 0x0DUL)
- #define CC_MNG_KCE_IS_LOCKED_ERR (CC_MNG_MODULE_ERROR_BASE + 0x0EUL)
- #define CC_MNG_RMA_ILLEGAL_STATE_ERR (CC_MNG_MODULE_ERROR_BASE + 0x0FUL)
- #define CC_MNG_AO_WRITE_FAILED_ERR (CC_MNG_MODULE_ERROR_BASE + 0x10UL)
- #define CC_MNG_APBC_ACCESS_FAILED_ERR (CC_MNG_MODULE_ERROR_BASE + 0x11UL)
- #define CC_MNG_PM_SUSPEND_RESUME_FAILED_ERR (CC_MNG_MODULE_ERROR_BASE + 0x12UL)

- #define *CC_MNG_ILLEGAL_SW_VERSION_ERR* (*CC_MNG_MODULE_ERROR_BASE* + 0x13UL)
- #define *CC_MNG_HASH_NOT_PROGRAMMED_ERR* (*CC_MNG_MODULE_ERROR_BASE* + 0x14UL)
- #define *CC_MNG_HBK_ZERO_COUNT_ERR* (*CC_MNG_MODULE_ERROR_BASE* + 0x15UL)

Detailed description

This file contains the error definitions of the CryptoCell management APIs.

1.7.58 mbedtls_cc_poly.h File Reference

This file contains all of the CryptoCell POLY APIs, their enums and definitions.

#include "cc_pal_types.h"

#include "cc_error.h"

Macros

- #define *CC_POLY_KEY_SIZE_IN_WORDS* 8
- #define *CC_POLY_KEY_SIZE_IN_BYTES* (*CC_POLY_KEY_SIZE_IN_WORDS***CC_32BIT_WORD_SIZE*)
- #define *CC_POLY_MAC_SIZE_IN_WORDS* 4
- #define *CC_POLY_MAC_SIZE_IN_BYTES* (*CC_POLY_MAC_SIZE_IN_WORDS***CC_32BIT_WORD_SIZE*)

Typedefs

- typedef uint32_t *mbedtls_poly_mac*[*CC_POLY_MAC_SIZE_IN_WORDS*]
- typedef uint32_t *mbedtls_poly_key*[*CC_POLY_KEY_SIZE_IN_WORDS*]

Functions

- CIMPORT_C CCError_t *mbedtls_poly* (*mbedtls_poly_key* pKey, uint8_t *pDataIn, size_t dataInSize, *mbedtls_poly_mac* macRes)

  This function performs the POLY MAC Calculation.

Detailed description

This file contains all of the CryptoCell POLY APIs, their enums and definitions.

1.7.59 mbedtls_cc_poly_error.h File Reference

This file contains the error definitions of the CryptoCell POLY APIs.

#include "cc_error.h"

Macros

- #define *CC_POLY_KEY_INVALID_ERROR* (*CC_POLY_MODULE_ERROR_BASE* + 0x01UL)
- #define *CC_POLY_DATA_INVALID_ERROR* (*CC_POLY_MODULE_ERROR_BASE* + 0x02UL)
- #define *CC_POLY_DATA_SIZE_INVALID_ERROR* (*CC_POLY_MODULE_ERROR_BASE* + 0x03UL)
- #define *CC_POLY_RESOURCES_ERROR* (*CC_POLY_MODULE_ERROR_BASE* + 0x04UL)

Detailed description

This file contains the error definitions of the CryptoCell POLY APIs.

1.7.60 mbedtls_cc_sbrt.h File Reference

This file contains CryptoCell Secure Boot certificate-chain processing APIs.

#include "secureboot_defs.h"

#include "secureboot_gen_defs.h"

Functions

- CCError_t *mbedtls_sb_cert_chain_cerification_init* (*CCSbCertInfo_t* *certPkgInfo)

  This function initializes the Secure Boot certificate-chain processing.

- CCError_t *mbedtls_sb_cert_verify_single* (*CCSbFlashReadFunc* flashReadFunc, void *userContext, *CCAddr_t* certStoreAddress, *CCSbCertInfo_t* *pCertPkgInfo, uint32_t *pHeader, uint32_t headerSize, uint32_t *pWorkspace, uint32_t workspaceSize)

  This function verifies a single certificate package containing either a key or content certificate.

- CCError_t *mbedtls_sb_sw_image_store_address_change* (uint32_t *pCert, uint32_t maxCertSizeWords, *CCAddr_t* address, uint32_t indexOfAddress)

  This function changes the storage address of a specific SW image in the content certificate.

Detailed description

This file contains CryptoCell Secure Boot certificate-chain processing APIs.

1.7.61 mbedtls_cc_sha512_t.h File Reference

This file contains all of the CryptoCell SHA-512 truncated APIs, their enums and definitions.

#include <sha512.h>

Functions

- void *mbedtls_sha512_t_init* (*mbedtls_sha512_context* *ctx)

  This function initializes the SHA-512_t context.
- void *mbedtls_sha512_t_free* (*mbedtls_sha512_context* *ctx)

  This function clears the SHA-512_t context.
- void *mbedtls_sha512_t_starts* (*mbedtls_sha512_context* *ctx, int is224)

  This function starts a SHA-512_t checksum calculation.
- void *mbedtls_sha512_t_update* (*mbedtls_sha512_context* *ctx, const unsigned char *input, size_t ilen)

  This function feeds an input buffer into an ongoing SHA-512_t checksum calculation.
- void *mbedtls_sha512_t_finish* (*mbedtls_sha512_context* *ctx, unsigned char output[32], int is224)

  This function finishes the SHA-512_t operation, and writes the result to the output buffer.
- void *mbedtls_sha512_t* (const unsigned char *input, size_t ilen, unsigned char output[32], int is224)

  This function calculates the SHA-512 checksum of a buffer.

Detailed description

This file contains all of the CryptoCell SHA-512 truncated APIs, their enums and definitions.

1.7.62 mbedtls_cc_srp.h File Reference

This file contains all of the CryptoCell SRP APIs, their enums and definitions.

#include "cc_pal_types.h"

#include "cc_error.h"

#include "cc_pka_defs_hw.h"

#include "cc_hash_defs.h"

#include "cc_rnd_common.h"

Data structures

- struct *mbedtls_srp_group_param*
- Group parameters for the SRP. struct *mbedtls_srp_context*

Macros

- #define *CC_SRP_MODULUS_SIZE_1024_BITS* 1024
- #define *CC_SRP_MODULUS_SIZE_1536_BITS* 1536
- #define *CC_SRP_MODULUS_SIZE_2048_BITS* 2048
- #define *CC_SRP_MODULUS_SIZE_3072_BITS* 3072
- #define *CC_SRP_MAX_MODULUS_IN_BITS* *CC_SRP_MODULUS_SIZE_3072_BITS*
- #define *CC_SRP_MAX_MODULUS* (*CC_SRP_MAX_MODULUS_IN_BITS*/*CC_BITS_IN_BYTE*)
- #define *CC_SRP_MAX_MODULUS_IN_WORDS* (*CC_SRP_MAX_MODULUS_IN_BITS*/*CC_BITS_IN_32BIT_WORD*)
- #define *CC_SRP_PRIV_NUM_MIN_SIZE_IN_BITS* (256)
- #define *CC_SRP_PRIV_NUM_MIN_SIZE* (*CC_SRP_PRIV_NUM_MIN_SIZE_IN_BITS*/*CC_BITS_IN_BYTE*)
- #define *CC_SRP_PRIV_NUM_MIN_SIZE_IN_WORDS* (*CC_SRP_PRIV_NUM_MIN_SIZE_IN_BITS*/*CC_BITS_IN_32BIT_WORD*)
- #define *CC_SRP_PRIV_NUM_MAX_SIZE_IN_BITS* (*CC_SRP_MAX_MODULUS_IN_BITS*)
- #define *CC_SRP_PRIV_NUM_MAX_SIZE* (*CC_SRP_PRIV_NUM_MAX_SIZE_IN_BITS*/*CC_BITS_IN_BYTE*)
- #define *CC_SRP_PRIV_NUM_MAX_SIZE_IN_WORDS* (*CC_SRP_PRIV_NUM_MAX_SIZE_IN_BITS*/*CC_BITS_IN_32BIT_WORD*)
- #define *CC_SRP_MAX_DIGEST_IN_WORDS* *CC_HASH_RESULT_SIZE_IN_WORDS*
- #define *CC_SRP_MAX_DIGEST* (*CC_SRP_MAX_DIGEST_IN_WORDS***CC_32BIT_WORD_SIZE*)
- #define *CC_SRP_MIN_SALT_SIZE* (8)

- #define *CC_SRP_MIN_SALT_SIZE_IN_WORDS* (*CC_SRP_MIN_SALT_SIZE*/*CC_32BIT_WORD_SIZE*)
- #define *CC_SRP_MAX_SALT_SIZE* (64)
- #define *CC_SRP_MAX_SALT_SIZE_IN_WORDS* (*CC_SRP_MAX_SALT_SIZE*/*CC_32BIT_WORD_SIZE*)
- #define *CC_SRP_HK_INIT*(srpType, srpModulus, srpGen, modSizeInBits, pUserName, userNameSize, pPwd, pwdSize, pRndCtx, pCtx) *mbedtls_srp_init*(srpType, *CC_SRP_VER_HK*, srpModulus, srpGen, modSizeInBits, *CC_HASH_SHA512_mode*, pUserName, userNameSize, pPwd, pwdSize, pRndCtx, pCtx)

Typedefs

- typedef uint8_t *mbedtls_srp_modulus*[*CC_SRP_MAX_MODULUS*]
- typedef uint8_t *mbedtls_srp_digest*[*CC_SRP_MAX_DIGEST*]
- typedef uint8_t *mbedtls_srp_secret*[2 **CC_SRP_MAX_DIGEST*]
- typedef struct *mbedtls_srp_group_param* *mbedtls_srp_group_param*

  Group parameters for the SRP.
- typedef struct *mbedtls_srp_context* *mbedtls_srp_context*

Enumerations

- enum *mbedtls_srp_version_t* { *CC_SRP_VER_3* = 0, *CC_SRP_VER_6* = 1, *CC_SRP_VER_6A* = 2, *CC_SRP_VER_HK* = 3, CC_SRP_NumOfVersions, *CC_SRP_VersionLast* = 0x7FFFFFFF }
- enum *mbedtls_srp_entity_t* { *CC_SRP_HOST* = 1, *CC_SRP_USER* = 2, CC_SRP_NumOfEntityType, *CC_SRP_EntityLast* = 0x7FFFFFFF }

Functions

- CIMPORT_C CCError_t *mbedtls_srp_init* (*mbedtls_srp_entity_t* srpType, *mbedtls_srp_version_t* srpVer, *mbedtls_srp_modulus* srpModulus, uint8_t srpGen, size_t modSizeInBits, *CCHashOperationMode_t* hashMode, uint8_t *pUserName, size_t userNameSize, uint8_t *pPwd, size_t pwdSize, *CCRndContext_t* *pRndCtx, *mbedtls_srp_context* *pCtx)

  This function initiates the SRP context.
- CIMPORT_C CCError_t *mbedtls_srp_pwd_ver_create* (size_t saltSize, uint8_t *pSalt, *mbedtls_srp_modulus* pwdVerifier, *mbedtls_srp_context* *pCtx)

  This function calculates pSalt and pwdVerifier .
- CIMPORT_C CCError_t *mbedtls_srp_clear* (*mbedtls_srp_context* *pCtx)

  This function clears the SRP context.
- CIMPORT_C CCError_t *mbedtls_srp_host_pub_key_create* (size_t ephemPrivSize, *mbedtls_srp_modulus* pwdVerifier, *mbedtls_srp_modulus* hostPubKeyB, *mbedtls_srp_context* *pCtx)

  This function generates the public and private host ephemeral keys, known as B and b in RFC 5054 Using the Secure Remote Password (SRP) Protocol for TLS Authentication .

- CIMPORT_C CCError_t *mbedtls_srp_host_proof_verify_and_calc* (size_t saltSize, uint8_t *pSalt, *mbedtls_srp_modulus* pwdVerifier, *mbedtls_srp_modulus* userPubKeyA, *mbedtls_srp_modulus* hostPubKeyB, *mbedtls_srp_digest* userProof, *mbedtls_srp_digest* hostProof, *mbedtls_srp_secret* sharedSecret, *mbedtls_srp_context* *pCtx)

  This function verifies the user proof, and calculates the host-message proof.

- CIMPORT_C CCError_t *mbedtls_srp_user_pub_key_create* (size_t ephemPrivSize, *mbedtls_srp_modulus* userPubKeyA, *mbedtls_srp_context* *pCtx)

  This function generates public and private user ephemeral keys, known as A and a in RFC 5054 Using the Secure Remote Password (SRP) Protocol for TLS Authentication .

- CIMPORT_C CCError_t *mbedtls_srp_user_proof_calc* (size_t saltSize, uint8_t *pSalt, *mbedtls_srp_modulus* userPubKeyA, *mbedtls_srp_modulus* hostPubKeyB, *mbedtls_srp_digest* userProof, *mbedtls_srp_secret* sharedSecret, *mbedtls_srp_context* *pCtx)

  This function calculates the user proof.

- CIMPORT_C CCError_t *mbedtls_srp_user_proof_verify* (*mbedtls_srp_secret* sharedSecret, *mbedtls_srp_modulus* userPubKeyA, *mbedtls_srp_digest* userProof, *mbedtls_srp_digest* hostProof, *mbedtls_srp_context* *pCtx)

  This function verifies the host proof.

Detailed description

This file contains all of the CryptoCell SRP APIs, their enums and definitions.

1.7.63 mbedtls_cc_srp_error.h File Reference

This file contains the error definitions of the CryptoCell SRP APIs.

#include "cc_error.h"

Macros

- #define *CC_SRP_PARAM_INVALID_ERROR* (*CC_SRP_MODULE_ERROR_BASE* + 0x01UL)
- #define *CC_SRP_MOD_SIZE_INVALID_ERROR* (*CC_SRP_MODULE_ERROR_BASE* + 0x02UL)
- #define *CC_SRP_STATE_UNINITIALIZED_ERROR* (*CC_SRP_MODULE_ERROR_BASE* + 0x03UL)
- #define *CC_SRP_RESULT_ERROR* (*CC_SRP_MODULE_ERROR_BASE* + 0x04UL)
- #define *CC_SRP_PARAM_ERROR* (*CC_SRP_MODULE_ERROR_BASE* + 0x05UL)
- #define *CC_SRP_INTERNAL_ERROR* (*CC_SRP_MODULE_ERROR_BASE* + 0x06UL)

Detailed description

This file contains the error definitions of the CryptoCell SRP APIs.

1.7.64 mbedtls_cc_util_asset_prov.h File Reference

This file contains CryptoCell runtime-library ICV and OEM asset-provisioning APIs and definitions.

#include "cc_pal_types_plat.h"

Macros

- #define *CC_ASSET_PROV_MAX_ASSET_PKG_SIZE*   560

Enumerations

- enum *CCAssetProvKeyType_t* { *ASSET_PROV_KEY_TYPE_KPICV* = 1, *ASSET_PROV_KEY_TYPE_KCP* = 2, *ASSET_PROV_KEY_TYPE_RESERVED* = 0x7FFFFFFF }

Functions

- CCError_t *mbedtls_util_asset_pkg_unpack* (*CCAssetProvKeyType_t* keyType, uint32_t assetId, uint32_t *pAssetPackage, size_t assetPackageLen, uint8_t *pAssetData, size_t *pAssetDataLen)

  This API securely provisions ICV or OEM assets to devices, using CryptoCell.

Detailed description

This file contains CryptoCell runtime-library ICV and OEM asset-provisioning APIs and definitions.

1.7.65 mbedtls_cc_util_defs.h File Reference

This file contains general definitions of the CryptoCell utility APIs.

#include "cc_pal_types_plat.h"

#include "mbedtls_cc_util_key_derivation_defs.h"

Data structures

- struct *mbedtls_util_keydata*

Macros

- #define *CC_UTIL_AES_128BIT_SIZE* 16
- #define *CC_UTIL_AES_192BIT_SIZE* 24
- #define *CC_UTIL_AES_256BIT_SIZE* 32
- #define *CC_UTIL_CMAC_DERV_MIN_DATA_IN_SIZE* *CC_UTIL_FIX_DATA_MIN_SIZE_IN_BYTES*+2
- #define *CC_UTIL_CMAC_DERV_MAX_DATA_IN_SIZE* *CC_UTIL_MAX_KDF_SIZE_IN_BYTES*
- #define *CC_UTIL_AES_CMAC_RESULT_SIZE_IN_BYTES* 0x10UL
- #define *CC_UTIL_AES_CMAC_RESULT_SIZE_IN_WORDS* (*CC_UTIL_AES_CMAC_RESULT_SIZE_IN_BYTES*/sizeof(uint32_t))

Typedefs

- typedef uint32_t *CCUtilError_t*
- typedef struct *mbedtls_util_keydata* *mbedtls_util_keydata*

Detailed description

This file contains general definitions of the CryptoCell utility APIs.

1.7.66 mbedtls_cc_util_key_derivation.h File Reference

This file contains the CryptoCell utility key-derivation function APIs.

#include "mbedtls_cc_util_defs.h"

#include "mbedtls_cc_util_key_derivation_defs.h"

#include "cc_hash_defs.h"

Macros

- #define *mbedtls_util_key_derivation_cmac*(keyType, pUserKey, pLabel, labelSize, pContextData, contextSize, pDerivedKey, derivedKeySize) *mbedtls_util_key_derivation*(keyType, pUserKey, *CC_UTIL_PRF_CMAC*, *CC_HASH_OperationModeLast*, pLabel, labelSize, pContextData, contextSize, pDerivedKey, derivedKeySize)

  This function performs key derivation using CMAC.

- #define *mbedtls_util_key_derivation_hmac*(keyType, pUserKey, hashMode, pLabel, labelSize, pContextData, contextSize, pDerivedKey, derivedKeySize) *mbedtls_util_key_derivation*(keyType, pUserKey, *CC_UTIL_PRF_HMAC*, hashMode, pLabel, labelSize, pContextData, contextSize, pDerivedKey, derivedKeySize)

  This function performs key derivation using HMAC.

Enumerations

- enum *mbedtls_util_keytype_t* { *CC_UTIL_USER_KEY* = 0, *CC_UTIL_ROOT_KEY* = 1, *CC_UTIL_TOTAL_KEYS* = 2, *CC_UTIL_END_OF_KEY_TYPE* = 0x7FFFFFFF }
- enum *mbedtls_util_prftype_t* { *CC_UTIL_PRF_CMAC* = 0, *CC_UTIL_PRF_HMAC* = 1, *CC_UTIL_TOTAL_PRFS* = 2, *CC_UTIL_END_OF_PRF_TYPE* = 0x7FFFFFFF }

Functions

- *CCUtilError_t* *mbedtls_util_key_derivation* (*mbedtls_util_keytype_t* keyType, *mbedtls_util_keydata* *pUserKey, *mbedtls_util_prftype_t* prfType, *CCHashOperationMode_t* hashMode, const uint8_t *pLabel, size_t labelSize, const uint8_t *pContextData, size_t contextSize, uint8_t *pDerivedKey, size_t derivedKeySize)

  This function performs key derivation using AES-CMAC.

Detailed description

This file contains the CryptoCell utility key-derivation function APIs.

The key-derivation function is defined as specified in the KDF in Counter Mode section in NIST Special Publication 800-108: Recommendation for Key Derivation Using Pseudorandom Functions .

1.7.67 mbedtls_cc_util_key_derivation_defs.h File Reference

This file contains the definitions for the key-derivation API.

Macros

- #define *CC_UTIL_MAX_LABEL_LENGTH_IN_BYTES* 64
- #define *CC_UTIL_MAX_CONTEXT_LENGTH_IN_BYTES* 64
- #define *CC_UTIL_FIX_DATA_MIN_SIZE_IN_BYTES* 3
- #define *CC_UTIL_FIX_DATA_MAX_SIZE_IN_BYTES* 4
- #define *CC_UTIL_MAX_KDF_SIZE_IN_BYTES* (*CC_UTIL_MAX_LABEL_LENGTH_IN_BYTES*+*CC_UTIL_MAX_CONTEXT_LENGTH_IN_BYTES*+*CC_UTIL_FIX_DATA_MAX_SIZE_IN_BYTES*)
- #define *CC_UTIL_MAX_DERIVED_KEY_SIZE_IN_BYTES* 4080

Detailed description

This file contains the definitions for the key-derivation API.

1.7.68 mbedtls_ccm_common.h File Reference

This file contains the common definitions of the CryptoCell AES-CCM star APIs.

Macros

- #define *MBEDTLS_AESCCM_STAR_NONCE_SIZE_BYTES* 13
- #define *MBEDTLS_AESCCM_STAR_SOURCE_ADDRESS_SIZE_BYTES* 8
- #define *MBEDTLS_AESCCM_MODE_CCM* 0
- #define *MBEDTLS_AESCCM_MODE_STAR* 1

Detailed description

This file contains the common definitions of the CryptoCell AES-CCM star APIs.

1.7.69 mbedtls_chacha_ext_dma.h File Reference

This file contains all the CryptoCell ChaCha external DMA APIs, their enums and definitions.

#include "cc_pal_types.h"

#include "mbedtls_cc_chacha.h"

Functions

- int *mbedtls_ext_dma_chacha_init* (uint8_t *pNonce, *mbedtls_chacha_nonce_size_t* nonceSizeFlag, uint8_t *pKey, uint32_t keySizeBytes, uint32_t initialCounter, *mbedtls_chacha_encrypt_mode_t* EncryptDecryptFlag)

  This function initializes the external DMA control. It configures the ChaCha mode, the initial hash value, and other configurations in the ChaCha engine.

- int *mbedtls_chacha_ext_dma_finish* (void)

  This function frees used resources.

Detailed description

This file contains all the CryptoCell ChaCha external DMA APIs, their enums and definitions.

1.7.70 mbedtls_ext_dma_error.h File Reference

This file contains the error definitions of the CryptoCell external DMA APIs.

#include "cc_error.h"

Macros

- #define *EXT_DMA_AES_ILLEGAL_OPERATION_MODE_ERROR* (*CC_EXT_DMA_MODULE_ERROR_BASE* + 0x00UL)
- #define *EXT_DMA_AES_INVALID_ENCRYPT_MODE_ERROR* (*CC_EXT_DMA_MODULE_ERROR_BASE* + 0x01UL)
- #define *EXT_DMA_AES_DECRYPTION_NOT_ALLOWED_ON_THIS_MODE* (*CC_EXT_DMA_MODULE_ERROR_BASE* + 0x02UL)
- #define *EXT_DMA_AES_ILLEGAL_KEY_SIZE_ERROR* (*CC_EXT_DMA_MODULE_ERROR_BASE* + 0x03UL)
- #define *EXT_DMA_AES_INVALID_IV_OR_TWEAK_PTR_ERROR* (*CC_EXT_DMA_MODULE_ERROR_BASE* + 0x04UL)
- #define *EXT_DMA_HASH_ILLEGAL_OPERATION_MODE_ERROR* (*CC_EXT_DMA_MODULE_ERROR_BASE* + 0x05UL)
- #define *EXT_DMA_HASH_INVALID_RESULT_BUFFER_POINTER_ERROR* (*CC_EXT_DMA_MODULE_ERROR_BASE* + 0x06UL)
- #define *EXT_DMA_HASH_ILLEGAL_PARAMS_ERROR* (*CC_EXT_DMA_MODULE_ERROR_BASE* + 0x07UL)
- #define *EXT_DMA_CHACHA_INVALID_NONCE_PTR_ERROR* (*CC_EXT_DMA_MODULE_ERROR_BASE* + 0x08UL)
- #define *EXT_DMA_CHACHA_INVALID_ENCRYPT_MODE_ERROR* (*CC_EXT_DMA_MODULE_ERROR_BASE* + 0x09UL)
- #define *EXT_DMA_CHACHA_INVALID_KEY_POINTER_ERROR* (*CC_EXT_DMA_MODULE_ERROR_BASE* + 0xAUL)
- #define *EXT_DMA_CHACHA_ILLEGAL_KEY_SIZE_ERROR* (*CC_EXT_DMA_MODULE_ERROR_BASE* + 0xBUL)
- #define *EXT_DMA_CHACHA_INVALID_NONCE_ERROR* (*CC_EXT_DMA_MODULE_ERROR_BASE* + 0xCUL)

Detailed description

This file contains the error definitions of the CryptoCell external DMA APIs.

1.7.71 mbedtls_hash_ext_dma.h File Reference

This file contains all the CryptoCell hash external DMA APIs, their enums and definitions.

#include "cc_pal_types.h"

#include "cc_hash_defs.h"

Functions

- int *mbedtls_hash_ext_dma_init* (*CCHashOperationMode_t* operationMode)

  This function initializes the External DMA Control.
- int *mbedtls_hash_ext_dma_finish* (*CCHashOperationMode_t* operationMode, uint32_t digestBufferSize, uint32_t *digestBuffer)

  This function returns the digest after the hash operation, and frees used resources.

Detailed description

This file contains all the CryptoCell hash external DMA APIs, their enums and definitions.

1.7.72 md.h File Reference

This file contains the Mbed TLS generic message-digest wrapper.

#include <stddef.h>

#include "config.h"

Data structures

- struct *mbedtls_md_context_t*

Macros

- #define *MBEDTLS_ERR_MD_FEATURE_UNAVAILABLE* -0x5080
- #define *MBEDTLS_ERR_MD_BAD_INPUT_DATA* -0x5100
- #define *MBEDTLS_ERR_MD_ALLOC_FAILED* -0x5180
- #define *MBEDTLS_ERR_MD_FILE_IO_ERROR* -0x5200
- #define *MBEDTLS_ERR_MD_HW_ACCEL_FAILED* -0x5280
- #define *MBEDTLS_MD_MAX_SIZE* 32 /* longest known is SHA256 or less */
- #define MBEDTLS_DEPRECATED

Typedefs

- typedef struct *mbedtls_md_info_t* *mbedtls_md_info_t*

Enumerations

- enum *mbedtls_md_type_t* { MBEDTLS_MD_NONE =0, MBEDTLS_MD_MD2, MBEDTLS_MD_MD4, MBEDTLS_MD_MD5, MBEDTLS_MD_SHA1, MBEDTLS_MD_SHA224, MBEDTLS_MD_SHA256, MBEDTLS_MD_SHA384, MBEDTLS_MD_SHA512, MBEDTLS_MD_RIPEMD160 }Enumeration of supported message digests.

Functions

- const int * *mbedtls_md_list* (void)

  This function returns the list of digests supported by the generic digest module.

- const *mbedtls_md_info_t* * *mbedtls_md_info_from_string* (const char *md_name)

  This function returns the message-digest information associated with the given digest name.

- const *mbedtls_md_info_t* * *mbedtls_md_info_from_type* (*mbedtls_md_type_t* md_type)

  This function returns the message-digest information associated with the given digest type.

- void *mbedtls_md_init* (*mbedtls_md_context_t* *ctx)

  This function initializes a message-digest context without binding it to a particular message-digest algorithm.

- void *mbedtls_md_free* (*mbedtls_md_context_t* *ctx)

  This function clears the internal structure of ctx and frees any embedded internal structure, but does not free ctx itself.

- int *mbedtls_md_init_ctx* (*mbedtls_md_context_t* *ctx, const *mbedtls_md_info_t* *md_info) MBEDTLS_DEPRECATED

  This function selects the message digest algorithm to use, and allocates internal structures.
- int *mbedtls_md_setup* (*mbedtls_md_context_t* *ctx, const *mbedtls_md_info_t* *md_info, int hmac)

  This function selects the message digest algorithm to use, and allocates internal structures.
- int *mbedtls_md_clone* (*mbedtls_md_context_t* *dst, const *mbedtls_md_context_t* *src)

  This function clones the state of an message-digest context.
- unsigned char *mbedtls_md_get_size* (const *mbedtls_md_info_t* *md_info)

  This function extracts the message-digest size from the message-digest information structure.
- *mbedtls_md_type_t* *mbedtls_md_get_type* (const *mbedtls_md_info_t* *md_info)

  This function extracts the message-digest type from the message-digest information structure.
- const char * *mbedtls_md_get_name* (const *mbedtls_md_info_t* *md_info)

  This function extracts the message-digest name from the message-digest information structure.
- int *mbedtls_md_starts* (*mbedtls_md_context_t* *ctx)

  This function starts a message-digest computation.
- int *mbedtls_md_update* (*mbedtls_md_context_t* *ctx, const unsigned char *input, size_t ilen)

  This function feeds an input buffer into an ongoing message-digest computation.
- int *mbedtls_md_finish* (*mbedtls_md_context_t* *ctx, unsigned char *output)

  This function finishes the digest operation, and writes the result to the output buffer.
- int *mbedtls_md* (const *mbedtls_md_info_t* *md_info, const unsigned char *input, size_t ilen, unsigned char *output)

  This function calculates the message-digest of a buffer, with respect to a configurable message-digest algorithm in a single call.
- int *mbedtls_md_hmac_starts* (*mbedtls_md_context_t* *ctx, const unsigned char *key, size_t keylen)

  This function sets the HMAC key and prepares to authenticate a new message.
- int *mbedtls_md_hmac_update* (*mbedtls_md_context_t* *ctx, const unsigned char *input, size_t ilen)

  This function feeds an input buffer into an ongoing HMAC computation.
- int *mbedtls_md_hmac_finish* (*mbedtls_md_context_t* *ctx, unsigned char *output)

  This function finishes the HMAC operation, and writes the result to the output buffer.
- int *mbedtls_md_hmac_reset* (*mbedtls_md_context_t* *ctx)

  This function prepares to authenticate a new message with the same key as the previous HMAC operation.
- int *mbedtls_md_hmac* (const *mbedtls_md_info_t* *md_info, const unsigned char *key, size_t keylen, const unsigned char *input, size_t ilen, unsigned char *output)

  This function calculates the full generic HMAC on the input buffer with the provided key.
- int *mbedtls_md_process* (*mbedtls_md_context_t* *ctx, const unsigned char *data)

This function processes a single data block within the ongoing message-digest computation. This function is for internal use only.

Detailed description

The generic message-digest wrapper.

Author:

Adriaan de Jong dejong@fox-it.com

Macro definition documentation

#define MBEDTLS_ERR_MD_ALLOC_FAILED   -0x5180

Failed to allocate memory.

#define MBEDTLS_ERR_MD_BAD_INPUT_DATA   -0x5100

Bad input parameters to function.

#define MBEDTLS_ERR_MD_FEATURE_UNAVAILABLE   -0x5080

The selected feature is not available.

#define MBEDTLS_ERR_MD_FILE_IO_ERROR   -0x5200

Opening or reading of file failed.

#define MBEDTLS_ERR_MD_HW_ACCEL_FAILED   -0x5280

MD hardware accelerator failed.

#define MBEDTLS_MD_MAX_SIZE   32   /* longest known is SHA256 or less */

The maximal size of a message digest.

Typedef documentation

typedef struct mbedtls_md_info_t mbedtls_md_info_t

Opaque struct defined in md_internal.h.

Enumeration type documentation

enum mbedtls_md_type_t

Enumeration of supported message digests.

Warning

MD2, MD4, MD5 and SHA-1 are considered weak message digests and their use constitutes a security risk. We recommend considering stronger message digests instead.

Function documentation

int mbedtls_md (const mbedtls_md_info_t *   md_info, const unsigned char *   input, size_t   ilen, unsigned char *   output)

This function calculates the message-digest of a buffer, with respect to a configurable message-digest algorithm in a single call.

The result is calculated as Output = message_digest(input buffer).

Parameters:

Parameter	Description
`md_info`	The information structure of the message-digest algorithm to use.
`input`	The buffer holding the data.
`ilen`	The length of the input data.
`output`	The generic message-digest checksum result.

Returns:

0 on success, or *MBEDTLS_ERR_MD_BAD_INPUT_DATA* if parameter verification fails.

int mbedtls_md_clone (*mbedtls_md_context_t* * dst, const *mbedtls_md_context_t* * src)

This function clones the state of an message-digest context.

Note

You must call *mbedtls_md_setup()* on dst before calling this function.

The two contexts must have the same type, for example, both are SHA-256.

Warning

This function clones the message-digest state, not the HMAC state.

Parameters:

Parameter	Description
`dst`	The destination context.
`src`	The context to be cloned.

Returns:

0 on success, *MBEDTLS_ERR_MD_BAD_INPUT_DATA* on parameter failure.

int mbedtls_md_finish (*mbedtls_md_context_t* * ctx, unsigned char * output)

This function finishes the digest operation, and writes the result to the output buffer.

Call this function after a call to *mbedtls_md_starts()*, followed by any number of calls to *mbedtls_md_update()*. Afterwards, you may either clear the context with *mbedtls_md_free()*, or call *mbedtls_md_starts()* to reuse the context for another digest operation with the same algorithm.

Parameters:

Parameter	Description
`ctx`	The generic message-digest context.
`output`	The buffer for the generic message-digest checksum result.

Returns:

0 on success, or *MBEDTLS_ERR_MD_BAD_INPUT_DATA* if parameter verification fails.

void mbedtls_md_free (mbedtls_md_context_t * ctx)

This function clears the internal structure of ctx and frees any embedded internal structure, but does not free ctx itself.

If you have called *mbedtls_md_setup()* on ctx , you must call *mbedtls_md_free()* when you are no longer using the context. Calling this function if you have previously called *mbedtls_md_init()* and nothing else is optional. You must not call this function if you have not called *mbedtls_md_init()*.

const char* mbedtls_md_get_name (const mbedtls_md_info_t * md_info)

This function extracts the message-digest name from the message-digest information structure.

Parameters:

Parameter	Description
md_info	The information structure of the message-digest algorithm to use.

Returns:

The name of the message digest.

unsigned char mbedtls_md_get_size (const mbedtls_md_info_t * md_info)

This function extracts the message-digest size from the message-digest information structure.

Parameters:

Parameter	Description
md_info	The information structure of the message-digest algorithm to use.

Returns:

The size of the message-digest output in Bytes.

mbedtls_md_type_t mbedtls_md_get_type (const mbedtls_md_info_t * md_info)

This function extracts the message-digest type from the message-digest information structure.

Parameters:

Parameter	Description
md_info	The information structure of the message-digest algorithm to use.

Returns:

The type of the message digest.

int mbedtls_md_hmac (const mbedtls_md_info_t * md_info, const unsigned char * key, size_t keylen, const unsigned char * input, size_t ilen, unsigned char * output)

This function calculates the full generic HMAC on the input buffer with the provided key.

The function allocates the context, performs the calculation, and frees the context.

The HMAC result is calculated as output = generic HMAC(hmac key, input buffer).

Parameters:

Parameter	Description
`md_info`	The information structure of the message-digest algorithm to use.
`key`	The HMAC secret key.
`keylen`	The length of the HMAC secret key in Bytes.
`input`	The buffer holding the input data.
`ilen`	The length of the input data.
`output`	The generic HMAC result.

Returns:

0 on success, or *MBEDTLS_ERR_MD_BAD_INPUT_DATA* if parameter verification fails.

int mbedtls_md_hmac_finish (*mbedtls_md_context_t* * ctx, unsigned char * output)

This function finishes the HMAC operation, and writes the result to the output buffer.

Call this function after *mbedtls_md_hmac_starts()* and *mbedtls_md_hmac_update()* to get the HMAC value. Afterwards you may either call *mbedtls_md_free()* to clear the context, or call *mbedtls_md_hmac_reset()* to reuse the context with the same HMAC key.

Parameters:

Parameter	Description
`ctx`	The message digest context containing an embedded HMAC context.
`output`	The generic HMAC checksum result.

Returns:

0 on success, or *MBEDTLS_ERR_MD_BAD_INPUT_DATA* if parameter verification fails.

int mbedtls_md_hmac_reset (*mbedtls_md_context_t* * ctx)

This function prepares to authenticate a new message with the same key as the previous HMAC operation.

You may call this function after *mbedtls_md_hmac_finish()*. Afterwards call *mbedtls_md_hmac_update()* to pass the new input.

Parameters:

Parameter	Description
`ctx`	The message digest context containing an embedded HMAC context.

Returns:

0 on success, or *MBEDTLS_ERR_MD_BAD_INPUT_DATA* if parameter verification fails.

int mbedtls_md_hmac_starts (*mbedtls_md_context_t* * ctx, const unsigned char * key, size_t keylen)

This function sets the HMAC key and prepares to authenticate a new message.

Call this function after *mbedtls_md_setup()*, to use the MD context for an HMAC calculation, then call *mbedtls_md_hmac_update()* to provide the input data, and *mbedtls_md_hmac_finish()* to get the HMAC value.

Parameters:

Parameter	Description
`ctx`	The message digest context containing an embedded HMAC context.
`key`	The HMAC secret key.
`keylen`	The length of the HMAC key in Bytes.

Returns:

0 on success, or *MBEDTLS_ERR_MD_BAD_INPUT_DATA* if parameter verification fails.

int mbedtls_md_hmac_update (*mbedtls_md_context_t* * ctx, const unsigned char * input, size_t ilen)

This function feeds an input buffer into an ongoing HMAC computation.

Call *mbedtls_md_hmac_starts()* or *mbedtls_md_hmac_reset()* before calling this function. You may call this function multiple times to pass the input piecewise. Afterwards, call *mbedtls_md_hmac_finish()*.

Parameters:

Parameter	Description
`ctx`	The message digest context containing an embedded HMAC context.
`input`	The buffer holding the input data.
`ilen`	The length of the input data.

Returns:

0 on success, or *MBEDTLS_ERR_MD_BAD_INPUT_DATA* if parameter verification fails.

const *mbedtls_md_info_t** mbedtls_md_info_from_string (const char * md_name)

This function returns the message-digest information associated with the given digest name.

Parameters:

Parameter	Description
`md_name`	The name of the digest to search for.

Returns:

The message-digest information associated with md_name , or NULL if not found.

const *mbedtls_md_info_t** mbedtls_md_info_from_type (*mbedtls_md_type_t* md_type)

This function returns the message-digest information associated with the given digest type.

Parameters:

Parameter	Description
`md_type`	The type of digest to search for.

Returns:

The message-digest information associated with md_type , or NULL if not found.

void mbedtls_md_init (*mbedtls_md_context_t* * ctx)

This function initializes a message-digest context without binding it to a particular message-digest algorithm.

This function should always be called first. It prepares the context for *mbedtls_md_setup()* for binding it to a message-digest algorithm.

int mbedtls_md_init_ctx (*mbedtls_md_context_t* * ctx, const *mbedtls_md_info_t* * md_info)

This function selects the message digest algorithm to use, and allocates internal structures.

It should be called after *mbedtls_md_init()* or *mbedtls_md_free()*. Makes it necessary to call *mbedtls_md_free()* later.

***Deprecated*:**

Superseded by *mbedtls_md_setup()* in 2.0.0

Parameters:

Parameter	Description
`ctx`	The context to set up.
`md_info`	The information structure of the message-digest algorithm to use.

Returns:

0 on success, *MBEDTLS_ERR_MD_BAD_INPUT_DATA* on parameter failure, *MBEDTLS_ERR_MD_ALLOC_FAILED* memory allocation failure.

const int* mbedtls_md_list (void)

This function returns the list of digests supported by the generic digest module.

Returns:

A statically allocated array of digests. Each element in the returned list is an integer belonging to the message-digest enumeration *mbedtls_md_type_t*. The last entry is 0.

int mbedtls_md_process (*mbedtls_md_context_t* * ctx, const unsigned char * data)

This function processes a single data block within the ongoing message-digest computation. This function is for internal use only.

Parameters:

Parameter	Description
`ctx`	The message-digest context.
`data`	The buffer holding one block of data.

Returns:

0 on success.

int mbedtls_md_setup (*mbedtls_md_context_t* * ctx, const *mbedtls_md_info_t* * md_info, int hmac)

This function selects the message digest algorithm to use, and allocates internal structures.

It should be called after *mbedtls_md_init()* or *mbedtls_md_free()*. Makes it necessary to call *mbedtls_md_free()* later.

Parameters:

Parameter	Description
`ctx`	The context to set up.
`md_info`	The information structure of the message-digest algorithm to use.
`hmac`	• 0: HMAC is not used. Saves some memory. • non-zero: HMAC is used with this context.

Returns:

0 on success, *MBEDTLS_ERR_MD_BAD_INPUT_DATA* on parameter failure, or *MBEDTLS_ERR_MD_ALLOC_FAILED* on memory allocation failure.

int mbedtls_md_starts (*mbedtls_md_context_t* * ctx)

This function starts a message-digest computation.

You must call this function after setting up the context with *mbedtls_md_setup()*, and before passing data with *mbedtls_md_update()*.

Parameters:

Parameter	Description
`ctx`	The generic message-digest context.

Returns:

0 on success, *MBEDTLS_ERR_MD_BAD_INPUT_DATA* if parameter verification fails.

int mbedtls_md_update (*mbedtls_md_context_t* * ctx, const unsigned char * input, size_t ilen)

This function feeds an input buffer into an ongoing message-digest computation.

You must call *mbedtls_md_starts()* before calling this function. You may call this function multiple times. Afterwards, call *mbedtls_md_finish()*.

Parameters:

Parameter	Description
`ctx`	The generic message-digest context.
`input`	The buffer holding the input data.
`ilen`	The length of the input data.

Returns:

0 on success, *MBEDTLS_ERR_MD_BAD_INPUT_DATA* if parameter verification fails.

1.7.73 platform.h File Reference

The Mbed TLS platform abstraction layer.

#include "config.h"

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

Data structures

- struct *mbedtls_platform_context*

The platform context structure. Macros

- #define mbedtls_free free
- #define mbedtls_calloc calloc
- #define mbedtls_fprintf fprintf
- #define mbedtls_printf printf
- #define mbedtls_snprintf snprintf
- #define mbedtls_exit exit
- #define MBEDTLS_EXIT_SUCCESS *MBEDTLS_PLATFORM_STD_EXIT_SUCCESS*
- #define MBEDTLS_EXIT_FAILURE *MBEDTLS_PLATFORM_STD_EXIT_FAILURE*

Module settings

The configuration options you can set for this module are in this section. Either change them in config.h or define them on the compiler command line.

  - #define *MBEDTLS_PLATFORM_STD_SNPRINTF* snprintf
  - #define *MBEDTLS_PLATFORM_STD_PRINTF* printf
  - #define *MBEDTLS_PLATFORM_STD_FPRINTF* fprintf
  - #define *MBEDTLS_PLATFORM_STD_CALLOC* calloc
  - #define *MBEDTLS_PLATFORM_STD_FREE* free
  - #define *MBEDTLS_PLATFORM_STD_EXIT* exit
  - #define *MBEDTLS_PLATFORM_STD_TIME* time
  - #define *MBEDTLS_PLATFORM_STD_EXIT_SUCCESS* EXIT_SUCCESS
  - #define *MBEDTLS_PLATFORM_STD_EXIT_FAILURE* EXIT_FAILURE

Functions

- int *mbedtls_platform_setup* (*mbedtls_platform_context* *ctx)

  This function performs any platform initialization operations.

- void *mbedtls_platform_teardown* (*mbedtls_platform_context* *ctx)

  This function performs any platform teardown operations.

Detailed description

The Mbed TLS platform abstraction layer.

Macro definition documentation

#define MBEDTLS_PLATFORM_STD_CALLOC calloc

The default calloc function to use.

#define MBEDTLS_PLATFORM_STD_EXIT exit

The default exit function to use.

#define MBEDTLS_PLATFORM_STD_EXIT_FAILURE EXIT_FAILURE

The default exit value to use.

#define MBEDTLS_PLATFORM_STD_EXIT_SUCCESS EXIT_SUCCESS

The default exit value to use.

#define MBEDTLS_PLATFORM_STD_FPRINTF fprintf

The default fprintf function to use.

#define MBEDTLS_PLATFORM_STD_FREE free

The default free function to use.

#define MBEDTLS_PLATFORM_STD_PRINTF printf

The default printf function to use.

#define MBEDTLS_PLATFORM_STD_SNPRINTF snprintf

The default snprintf function to use.

#define MBEDTLS_PLATFORM_STD_TIME time

The default time function to use.

Function documentation

int mbedtls_platform_setup (*mbedtls_platform_context* * ctx)

This function performs any platform initialization operations.

Parameters:

Parameter	Description
`ctx`	The Mbed TLS context.

Returns:

0 on success.

Note

This function is intended to allow platform-specific initialization, and should be called before any other library functions. Its implementation is platform-specific, and unless platform-specific code is provided, it does nothing.

Its use and whether it is necessary to call it is dependent on the platform.

void mbedtls_platform_teardown (*mbedtls_platform_context* * ctx)

This function performs any platform teardown operations.

Parameters:

Parameter	Description
`ctx`	The Mbed TLS context.

Note

This function should be called after every other Mbed TLS module has been correctly freed using the appropriate free function. Its implementation is platform-specific, and unless platform-specific code is provided, it does nothing.

Its use and whether it is necessary to call it is dependent on the platform.

1.7.74 rsa.h File Reference

This file contains the Mbed TLS RSA public-key cryptosystem APIs.

#include "config.h"

#include "bignum.h"

#include "md.h"

Data structures

- struct *mbedtls_rsa_context*

The RSA context structure. Macros

- #define *MBEDTLS_ERR_RSA_BAD_INPUT_DATA* -0x4080
- #define *MBEDTLS_ERR_RSA_INVALID_PADDING* -0x4100
- #define *MBEDTLS_ERR_RSA_KEY_GEN_FAILED* -0x4180
- #define *MBEDTLS_ERR_RSA_KEY_CHECK_FAILED* -0x4200
- #define *MBEDTLS_ERR_RSA_PUBLIC_FAILED* -0x4280
- #define *MBEDTLS_ERR_RSA_PRIVATE_FAILED* -0x4300
- #define *MBEDTLS_ERR_RSA_VERIFY_FAILED* -0x4380
- #define *MBEDTLS_ERR_RSA_OUTPUT_TOO_LARGE* -0x4400
- #define *MBEDTLS_ERR_RSA_RNG_FAILED* -0x4480
- #define *MBEDTLS_ERR_RSA_UNSUPPORTED_OPERATION* -0x4500
- #define *MBEDTLS_ERR_RSA_HW_ACCEL_FAILED* -0x4580
- #define *MBEDTLS_RSA_PUBLIC* 0
- #define *MBEDTLS_RSA_PRIVATE* 1
- #define *MBEDTLS_RSA_PKCS_V15* 0
- #define *MBEDTLS_RSA_PKCS_V21* 1
- #define *MBEDTLS_RSA_SIGN* 1
- #define *MBEDTLS_RSA_CRYPT* 2
- #define MBEDTLS_RSA_SALT_LEN_ANY -1 /*! The length of the salt used in padding. */

Functions

- void *mbedtls_rsa_init* (*mbedtls_rsa_context* *ctx, int padding, int hash_id)

  This function initializes an RSA context.

- int *mbedtls_rsa_import* (*mbedtls_rsa_context* *ctx, const mbedtls_mpi *N, const mbedtls_mpi *P, const mbedtls_mpi *Q, const mbedtls_mpi *D, const mbedtls_mpi *E)

  This function imports a set of core parameters into an RSA context.

- int *mbedtls_rsa_import_raw* (*mbedtls_rsa_context* *ctx, unsigned char const *N, size_t N_len, unsigned char const *P, size_t P_len, unsigned char const *Q, size_t Q_len, unsigned char const *D, size_t D_len, unsigned char const *E, size_t E_len)

  This function imports core RSA parameters, in raw big-endian binary format, into an RSA context.

- int *mbedtls_rsa_complete* (*mbedtls_rsa_context* *ctx)

  This function completes an RSA context from a set of imported core parameters.

- int *mbedtls_rsa_export* (const *mbedtls_rsa_context* *ctx, mbedtls_mpi *N, mbedtls_mpi *P, mbedtls_mpi *Q, mbedtls_mpi *D, mbedtls_mpi *E)

  This function exports the core parameters of an RSA key.

- int *mbedtls_rsa_export_raw* (const *mbedtls_rsa_context* *ctx, unsigned char *N, size_t N_len, unsigned char *P, size_t P_len, unsigned char *Q, size_t Q_len, unsigned char *D, size_t D_len, unsigned char *E, size_t E_len)

  This function exports core parameters of an RSA key in raw big-endian binary format.

- int *mbedtls_rsa_export_crt* (const *mbedtls_rsa_context* *ctx, mbedtls_mpi *DP, mbedtls_mpi *DQ, mbedtls_mpi *QP)

  This function exports CRT parameters of a private RSA key.

- void *mbedtls_rsa_set_padding* (*mbedtls_rsa_context* *ctx, int padding, int hash_id)

  This function sets padding for an already initialized RSA context. See *mbedtls_rsa_init()* for details.

- size_t *mbedtls_rsa_get_len* (const *mbedtls_rsa_context* *ctx)

  This function retrieves the length of RSA modulus in Bytes.

- int *mbedtls_rsa_gen_key* (*mbedtls_rsa_context* *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, unsigned int nbits, int exponent)

  This function generates an RSA keypair.

- int *mbedtls_rsa_check_pubkey* (const *mbedtls_rsa_context* *ctx)

  This function checks if a context contains at least an RSA public key.

- int *mbedtls_rsa_check_privkey* (const *mbedtls_rsa_context* *ctx)

  This function checks if a context contains an RSA private key and perform basic consistency checks.

- int *mbedtls_rsa_check_pub_priv* (const *mbedtls_rsa_context* *pub, const *mbedtls_rsa_context* *prv)

  This function checks a public-private RSA key pair.

- int *mbedtls_rsa_public* (*mbedtls_rsa_context* *ctx, const unsigned char *input, unsigned char *output)

  This function performs an RSA public key operation.

- int *mbedtls_rsa_private* (*mbedtls_rsa_context* *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, const unsigned char *input, unsigned char *output)

This function performs an RSA private key operation.

- int *mbedtls_rsa_pkcs1_encrypt* (*mbedtls_rsa_context* *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, size_t ilen, const unsigned char *input, unsigned char *output)

  This function adds the message padding, then performs an RSA operation.

- int *mbedtls_rsa_rsaes_pkcs1_v15_encrypt* (*mbedtls_rsa_context* *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, size_t ilen, const unsigned char *input, unsigned char *output)

  This function performs a PKCS#1 v1.5 encryption operation (RSAES-PKCS1-v1_5-ENCRYPT).

- int *mbedtls_rsa_rsaes_oaep_encrypt* (*mbedtls_rsa_context* *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, const unsigned char *label, size_t label_len, size_t ilen, const unsigned char *input, unsigned char *output)

  This function performs a PKCS#1 v2.1 OAEP encryption operation (RSAES-OAEP-ENCRYPT).

- int *mbedtls_rsa_pkcs1_decrypt* (*mbedtls_rsa_context* *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, size_t *olen, const unsigned char *input, unsigned char *output, size_t output_max_len)

  This function performs an RSA operation, then removes the message padding.

- int *mbedtls_rsa_rsaes_pkcs1_v15_decrypt* (*mbedtls_rsa_context* *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, size_t *olen, const unsigned char *input, unsigned char *output, size_t output_max_len)

  This function performs a PKCS#1 v1.5 decryption operation (RSAES-PKCS1-v1_5-DECRYPT).

- int *mbedtls_rsa_rsaes_oaep_decrypt* (*mbedtls_rsa_context* *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, const unsigned char *label, size_t label_len, size_t *olen, const unsigned char *input, unsigned char *output, size_t output_max_len)

  This function performs a PKCS#1 v2.1 OAEP decryption operation (RSAES-OAEP-DECRYPT).

- int *mbedtls_rsa_pkcs1_sign* (*mbedtls_rsa_context* *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, *mbedtls_md_type_t* md_alg, unsigned int hashlen, const unsigned char *hash, unsigned char *sig)

  This function performs a private RSA operation to sign a message digest using PKCS#1.

- int *mbedtls_rsa_rsassa_pkcs1_v15_sign* (*mbedtls_rsa_context* *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, *mbedtls_md_type_t* md_alg, unsigned int hashlen, const unsigned char *hash, unsigned char *sig)

  This function performs a PKCS#1 v1.5 signature operation (RSASSA-PKCS1-v1_5-SIGN).

- int *mbedtls_rsa_rsassa_pss_sign* (*mbedtls_rsa_context* *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, *mbedtls_md_type_t* md_alg, unsigned int hashlen, const unsigned char *hash, unsigned char *sig)

  This function performs a PKCS#1 v2.1 PSS signature operation (RSASSA-PSS-SIGN).

- int *mbedtls_rsa_pkcs1_verify* (*mbedtls_rsa_context* *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, *mbedtls_md_type_t* md_alg, unsigned int hashlen, const unsigned char *hash, const unsigned char *sig)

  This function performs a public RSA operation and checks the message digest.

- int *mbedtls_rsa_rsassa_pkcs1_v15_verify* (*mbedtls_rsa_context* *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, *mbedtls_md_type_t* md_alg, unsigned int hashlen, const unsigned char *hash, const unsigned char *sig)

  This function performs a PKCS#1 v1.5 verification operation (RSASSA-PKCS1-v1_5-VERIFY).
- int *mbedtls_rsa_rsassa_pss_verify* (*mbedtls_rsa_context* *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, *mbedtls_md_type_t* md_alg, unsigned int hashlen, const unsigned char *hash, const unsigned char *sig)

  This function performs a PKCS#1 v2.1 PSS verification operation (RSASSA-PSS-VERIFY).
- int *mbedtls_rsa_rsassa_pss_verify_ext* (*mbedtls_rsa_context* *ctx, int(*f_rng)(void *, unsigned char *, size_t), void *p_rng, int mode, *mbedtls_md_type_t* md_alg, unsigned int hashlen, const unsigned char *hash, *mbedtls_md_type_t* mgf1_hash_id, int expected_salt_len, const unsigned char *sig)

  This function performs a PKCS#1 v2.1 PSS verification operation (RSASSA-PSS-VERIFY).
- int *mbedtls_rsa_copy* (*mbedtls_rsa_context* *dst, const *mbedtls_rsa_context* *src)

  This function copies the components of an RSA context.
- void *mbedtls_rsa_free* (*mbedtls_rsa_context* *ctx)

  This function frees the components of an RSA key.
- int *mbedtls_rsa_self_test* (int verbose)

  The RSA checkup routine.

Detailed description

The RSA public-key cryptosystem.

For more information, see Public-Key Cryptography Standards (PKCS) #1 v1.5: RSA Encryption and Public-Key Cryptography Standards (PKCS) #1 v2.1: RSA Cryptography Specifications .

Macro definition documentation

#define MBEDTLS_ERR_RSA_BAD_INPUT_DATA -0x4080

Bad input parameters to function.

#define MBEDTLS_ERR_RSA_HW_ACCEL_FAILED -0x4580

RSA hardware accelerator failed.

#define MBEDTLS_ERR_RSA_INVALID_PADDING -0x4100

Input data contains invalid padding and is rejected.

#define MBEDTLS_ERR_RSA_KEY_CHECK_FAILED -0x4200

Key failed to pass the validity check of the library.

#define MBEDTLS_ERR_RSA_KEY_GEN_FAILED -0x4180

Something failed during generation of a key.

#define MBEDTLS_ERR_RSA_OUTPUT_TOO_LARGE -0x4400

The output buffer for decryption is not large enough.

#define MBEDTLS_ERR_RSA_PRIVATE_FAILED -0x4300

The private key operation failed.

#define MBEDTLS_ERR_RSA_PUBLIC_FAILED -0x4280

The public key operation failed.

#define MBEDTLS_ERR_RSA_RNG_FAILED -0x4480

The random generator failed to generate non-zeros.

#define MBEDTLS_ERR_RSA_UNSUPPORTED_OPERATION -0x4500

The implementation does not offer the requested operation, for example, because of security violations or lack of functionality.

#define MBEDTLS_ERR_RSA_VERIFY_FAILED -0x4380

The PKCS#1 verification failed.

#define MBEDTLS_RSA_CRYPT 2

Identifier for RSA encryption and decryption operations.

#define MBEDTLS_RSA_PKCS_V15 0

Use PKCS-1 v1.5 encoding.

#define MBEDTLS_RSA_PKCS_V21 1

Use PKCS-1 v2.1 encoding.

#define MBEDTLS_RSA_PRIVATE 1

Request public key operation.

#define MBEDTLS_RSA_PUBLIC 0

Request private key operation.

#define MBEDTLS_RSA_SIGN 1

Identifier for RSA signature operations.

Function documentation

int mbedtls_rsa_check_privkey (const *mbedtls_rsa_context* * ctx)

This function checks if a context contains an RSA private key and perform basic consistency checks.

Parameters:

Parameter	Description
`ctx`	The RSA context to check.

Returns:

0 on success, or an MBEDTLS_ERR_RSA_XXX error code on failure.

Note

The consistency checks performed by this function not only ensure that *mbedtls_rsa_private()* can be called successfully on the given context, but that the various parameters are mutually consistent with high probability, in the sense that *mbedtls_rsa_public()* and *mbedtls_rsa_private()* are inverses.

Warning

This function should catch accidental misconfigurations like swapping of parameters, but it cannot establish full trust in neither the quality nor the consistency of the key material that was used to setup the given RSA context:

- Consistency: Imported parameters that are irrelevant for the implementation might be silently dropped. If dropped, the current function does not have access to them, and therefore cannot check them. See *mbedtls_rsa_complete()*. If you want to check the consistency of the entire content of an PKCS1-encoded RSA private key, for example, you should use mbedtls_rsa_validate_params() before setting up the RSA context. Additionally, if the implementation performs empirical checks, these checks substantiate but do not guarantee consistency.
- Quality: This function is not expected to perform extended quality assessments like checking that the prime factors are safe. Additionally, it is the responsibility of the user to ensure the trustworthiness of the source of his RSA parameters, which goes beyond what is effectively checkable by the library.

---

int mbedtls_rsa_check_pub_priv (const *mbedtls_rsa_context* * pub, const *mbedtls_rsa_context* * prv)

This function checks a public-private RSA key pair.

It checks each of the contexts, and makes sure they match.

Parameters:

Parameter	Description
`pub`	The RSA context holding the public key.
`prv`	The RSA context holding the private key.

Returns:

0 on success, or an MBEDTLS_ERR_RSA_XXX error code on failure.

int mbedtls_rsa_check_pubkey (const *mbedtls_rsa_context* * ctx)

This function checks if a context contains at least an RSA public key.

If the function runs successfully, it is guaranteed that enough information is present to perform an RSA public key operation using *mbedtls_rsa_public()*.

Parameters:

Parameter	Description
`ctx`	The RSA context to check.

Returns:

0 on success, or an MBEDTLS_ERR_RSA_XXX error code on failure.

int mbedtls_rsa_complete (*mbedtls_rsa_context* * ctx)

This function completes an RSA context from a set of imported core parameters.

To setup an RSA public key, precisely N and E must have been imported.

To setup an RSA private key, sufficient information must be present for the other parameters to be derivable.

The default implementation supports the following:

- Derive P , Q from N , D , E .

- Derive N , D from P , Q , E .

Alternative implementations need not support these.

If this function runs successfully, it guarantees that the RSA context can be used for RSA operations without the risk of failure or crash.

Parameters:

Parameter	Description
`ctx`	The initialized RSA context holding imported parameters.

Returns:

0 on success, or *MBEDTLS_ERR_RSA_BAD_INPUT_DATA* if the attempted derivations failed.

Warning

This function need not perform consistency checks for the imported parameters. In particular, parameters that are not needed by the implementation might be silently discarded and left unchecked. To check the consistency of the key material, see *mbedtls_rsa_check_privkey()*.

int mbedtls_rsa_copy (*mbedtls_rsa_context* * dst, const *mbedtls_rsa_context* * src)

This function copies the components of an RSA context.

Parameters:

Parameter	Description
`dst`	The destination context.
`src`	The source context.

Returns:

0 on success, MBEDTLS_ERR_MPI_ALLOC_FAILED on memory allocation failure.

int mbedtls_rsa_export (const *mbedtls_rsa_context* * ctx, mbedtls_mpi * N, mbedtls_mpi * P, mbedtls_mpi * Q, mbedtls_mpi * D, mbedtls_mpi * E)

This function exports the core parameters of an RSA key.

If this function runs successfully, the non-NULL buffers pointed to by N , P , Q , D , and E are fully written, with additional unused space filled leading by zero Bytes.

Possible reasons for returning *MBEDTLS_ERR_RSA_UNSUPPORTED_OPERATION*:

- An alternative RSA implementation is in use, which stores the key externally, and either cannot or should not export it into RAM.
- A SW or HW implementation might not support a certain deduction. For example, P , Q from N , D , and E if the former are not part of the implementation.

If the function fails due to an unsupported operation, the RSA context stays intact and remains usable.

Parameters:

Parameter	Description
`ctx`	The initialized RSA context.

Parameter	Description
`N`	The MPI to hold the RSA modulus, or NULL.
`P`	The MPI to hold the first prime factor of N , or NULL.
`Q`	The MPI to hold the second prime factor of N , or NULL.
`D`	The MPI to hold the private exponent, or NULL.
`E`	The MPI to hold the public exponent, or NULL.

Returns:

0 on success, *MBEDTLS_ERR_RSA_UNSUPPORTED_OPERATION* if exporting the requested parameters cannot be done due to missing functionality or because of security policies, or a non-zero return code on any other failure.

int mbedtls_rsa_export_crt (const *mbedtls_rsa_context* * ctx, mbedtls_mpi * DP, mbedtls_mpi * DQ, mbedtls_mpi * QP)

This function exports CRT parameters of a private RSA key.

Parameters:

Parameter	Description
`ctx`	The initialized RSA context.
`DP`	The MPI to hold D modulo P-1, or NULL.
`DQ`	The MPI to hold D modulo Q-1, or NULL.
`QP`	The MPI to hold modular inverse of Q modulo P, or NULL.

Returns:

0 on success, non-zero error code otherwise.

Note

Alternative RSA implementations not using CRT-parameters internally can implement this function based on mbedtls_rsa_deduce_opt().

int mbedtls_rsa_export_raw (const *mbedtls_rsa_context* * ctx, unsigned char * N, size_t N_len, unsigned char * P, size_t P_len, unsigned char * Q, size_t Q_len, unsigned char * D, size_t D_len, unsigned char * E, size_t E_len)

This function exports core parameters of an RSA key in raw big-endian binary format.

If this function runs successfully, the non-NULL buffers pointed to by N , P , Q , D , and E are fully written, with additional unused space filled leading by zero Bytes.

Possible reasons for returning *MBEDTLS_ERR_RSA_UNSUPPORTED_OPERATION*:

- An alternative RSA implementation is in use, which stores the key externally, and either cannot or should not export it into RAM.
- A SW or HW implementation might not support a certain deduction. For example, P , Q from N , D , and E if the former are not part of the implementation.

If the function fails due to an unsupported operation, the RSA context stays intact and remains usable.

Note

The length fields are ignored if the corresponding buffer pointers are NULL.

Parameters:

Parameter	Description
`ctx`	The initialized RSA context.
`N`	The Byte array to store the RSA modulus, or NULL.
`N_len`	The size of the buffer for the modulus.
`P`	The Byte array to hold the first prime factor of N , or NULL.
`P_len`	The size of the buffer for the first prime factor.
`Q`	The Byte array to hold the second prime factor of N , or NULL.
`Q_len`	The size of the buffer for the second prime factor.
`D`	The Byte array to hold the private exponent, or NULL.
`D_len`	The size of the buffer for the private exponent.
`E`	The Byte array to hold the public exponent, or NULL.
`E_len`	The size of the buffer for the public exponent.

Returns:

0 on success, *MBEDTLS_ERR_RSA_UNSUPPORTED_OPERATION* if exporting the requested parameters cannot be done due to missing functionality or because of security policies, or a non-zero return code on any other failure.

void mbedtls_rsa_free (*mbedtls_rsa_context* * ctx)

This function frees the components of an RSA key.

Parameters:

Parameter	Description
`ctx`	The RSA Context to free.

int mbedtls_rsa_gen_key (*mbedtls_rsa_context* * ctx, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, unsigned int nbits, int exponent)

This function generates an RSA keypair.

Note

mbedtls_rsa_init() must be called before this function, to set up the RSA context.

Parameters:

Parameter	Description
`ctx`	The RSA context used to hold the key.
`f_rng`	The RNG function.
`p_rng`	The RNG parameter.
`nbits`	The size of the public key in bits.
`exponent`	The public exponent. For example, 65537.

Returns:

0 on success, or an MBEDTLS_ERR_RSA_XXX error code on failure.

size_t mbedtls_rsa_get_len (const *mbedtls_rsa_context* * ctx)

This function retrieves the length of RSA modulus in Bytes.

Parameters:

Parameter	Description
`ctx`	The initialized RSA context.

Returns:

The length of the RSA modulus in Bytes.

int mbedtls_rsa_import (*mbedtls_rsa_context* * ctx, const mbedtls_mpi * N, const mbedtls_mpi * P, const mbedtls_mpi * Q, const mbedtls_mpi * D, const mbedtls_mpi * E)

This function imports a set of core parameters into an RSA context.

Note

This function can be called multiple times for successive imports, if the parameters are not simultaneously present.

Any sequence of calls to this function should be followed by a call to *mbedtls_rsa_complete()*, which checks and completes the provided information to a ready-for-use public or private RSA key.

Note

See *mbedtls_rsa_complete()* for more information on which parameters are necessary to set up a private or public RSA key.

The imported parameters are copied and need not be preserved for the lifetime of the RSA context being set up.

Parameters:

Parameter	Description
`ctx`	The initialized RSA context to store the parameters in.
`N`	The RSA modulus, or NULL.
`P`	The first prime factor of N , or NULL.
`Q`	The second prime factor of N , or NULL.
`D`	The private exponent, or NULL.
`E`	The public exponent, or NULL.

Returns:

0 on success, or a non-zero error code on failure.

int mbedtls_rsa_import_raw (*mbedtls_rsa_context* * ctx, unsigned char const * N, size_t N_len, unsigned char const * P, size_t P_len, unsigned char const * Q, size_t Q_len, unsigned char const * D, size_t D_len, unsigned char const * E, size_t E_len)

This function imports core RSA parameters, in raw big-endian binary format, into an RSA context.

Note

This function can be called multiple times for successive imports, if the parameters are not simultaneously present.

Any sequence of calls to this function should be followed by a call to *mbedtls_rsa_complete()*, which checks and completes the provided information to a ready-for-use public or private RSA key.

Note

See *mbedtls_rsa_complete()* for more information on which parameters are necessary to set up a private or public RSA key.

The imported parameters are copied and need not be preserved for the lifetime of the RSA context being set up.

Parameters:

Parameter	Description
`ctx`	The initialized RSA context to store the parameters in.
`N`	The RSA modulus, or NULL.
`N_len`	The Byte length of N , ignored if N == NULL.
`P`	The first prime factor of N , or NULL.
`P_len`	The Byte length of P , ignored if P == NULL.
`Q`	The second prime factor of N , or NULL.
`Q_len`	The Byte length of Q , ignored if Q == NULL.
`D`	The private exponent, or NULL.
`D_len`	The Byte length of D , ignored if D == NULL.
`E`	The public exponent, or NULL.
`E_len`	The Byte length of E , ignored if E == NULL.

Returns:

0 on success, or a non-zero error code on failure.

void mbedtls_rsa_init (*mbedtls_rsa_context* * ctx, int padding, int hash_id)

This function initializes an RSA context.

Note

The hash_id parameter is ignored when using *MBEDTLS_RSA_PKCS_V15* padding.

The choice of padding mode is strictly enforced for private key operations, since there might be security concerns in mixing padding modes. For public key operations it is a default value, which can be overriden by calling specific rsa_rsaes_xxx or rsa_rsassa_xxx functions.

The hash selected in hash_id is always used for OEAP encryption. For PSS signatures, it is always used for making signatures, but can be overriden for verifying them. If set to MBEDTLS_MD_NONE , it is always overriden.

Note

Set padding to *MBEDTLS_RSA_PKCS_V21* for the RSAES-OAEP encryption scheme and the RSASSA-PSS signature scheme.

Parameters:

Parameter	Description
`ctx`	The RSA context to initialize.
`padding`	Selects padding mode: *MBEDTLS_RSA_PKCS_V15* or *MBEDTLS_RSA_PKCS_V21*.
`hash_id`	The hash identifier of *mbedtls_md_type_t* type, if padding is *MBEDTLS_RSA_PKCS_V21*.

int mbedtls_rsa_pkcs1_decrypt (*mbedtls_rsa_context* * ctx, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, int mode, size_t * olen, const unsigned char * input, unsigned char * output, size_t output_max_len)

This function performs an RSA operation, then removes the message padding.

It is the generic wrapper for performing a PKCS#1 decryption operation using the mode from the context.

Parameters:

Parameter	Description
`ctx`	The RSA context.
`f_rng`	The RNG function. Only needed for *MBEDTLS_RSA_PRIVATE*.
`p_rng`	The RNG parameter.
`mode`	*MBEDTLS_RSA_PUBLIC* or *MBEDTLS_RSA_PRIVATE*.
`olen`	The length of the plaintext.
`input`	The buffer holding the encrypted data.
`output`	The buffer used to hold the plaintext.
`output_max_len`	The maximum length of the output buffer.

***Deprecated*:**

It is deprecated and discouraged to call this function in *MBEDTLS_RSA_PUBLIC* mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to *MBEDTLS_RSA_PRIVATE*.

Note

Alternative implementations of RSA need not support mode being set to *MBEDTLS_RSA_PUBLIC* and might instead return *MBEDTLS_ERR_RSA_UNSUPPORTED_OPERATION*.

Returns:

0 on success, or an MBEDTLS_ERR_RSA_XXX error code on failure.

Note

The output buffer length output_max_len should be as large as the size ctx->len of ctx->N (for example, 128 Bytes if RSA-1024 is used) to be able to hold an arbitrary decrypted message. If it is not large enough to hold the decryption of the particular ciphertext provided, the function returns MBEDTLS_ERR_RSA_OUTPUT_TOO_LARGE .

The input buffer must be as large as the size of ctx->N . For example, 128 Bytes if RSA-1024 is used.

int mbedtls_rsa_pkcs1_encrypt (*mbedtls_rsa_context* * ctx, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, int mode, size_t ilen, const unsigned char * input, unsigned char * output)

This function adds the message padding, then performs an RSA operation.

It is the generic wrapper for performing a PKCS#1 encryption operation using the mode from the context.

***Deprecated*:**

It is deprecated and discouraged to call this function in *MBEDTLS_RSA_PRIVATE* mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to *MBEDTLS_RSA_PUBLIC*.

Note

Alternative implementations of RSA need not support mode being set to *MBEDTLS_RSA_PRIVATE* and might instead return *MBEDTLS_ERR_RSA_UNSUPPORTED_OPERATION*.

Note

The input and output buffers must be as large as the size of ctx->N . For example, 128 Bytes if RSA-1024 is used.

Parameters:

Parameter	Description
`ctx`	The RSA context.
`f_rng`	The RNG function. Needed for padding, PKCS#1 v2.1 encoding, and *MBEDTLS_RSA_PRIVATE*.
`p_rng`	The RNG parameter.
`mode`	*MBEDTLS_RSA_PUBLIC* or *MBEDTLS_RSA_PRIVATE*.
`ilen`	The length of the plaintext.
`input`	The buffer holding the data to encrypt.
`output`	The buffer used to hold the ciphertext.

Returns:

0 on success, or an MBEDTLS_ERR_RSA_XXX error code on failure.

int mbedtls_rsa_pkcs1_sign (*mbedtls_rsa_context* * ctx, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, int mode, *mbedtls_md_type_t* md_alg, unsigned int hashlen, const unsigned char * hash, unsigned char * sig)

This function performs a private RSA operation to sign a message digest using PKCS#1.

It is the generic wrapper for performing a PKCS#1 signature using the mode from the context.

***Deprecated*:**

It is deprecated and discouraged to call this function in *MBEDTLS_RSA_PUBLIC* mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to *MBEDTLS_RSA_PRIVATE*.

Note

Alternative implementations of RSA need not support mode being set to *MBEDTLS_RSA_PUBLIC* and might instead return *MBEDTLS_ERR_RSA_UNSUPPORTED_OPERATION*.

Note

The sig buffer must be as large as the size of ctx->N . For example, 128 Bytes if RSA-1024 is used.

For PKCS#1 v2.1 encoding, see comments on *mbedtls_rsa_rsassa_pss_sign()* for details on md_alg and hash_id .

Parameters:

Parameter	Description
`ctx`	The RSA context.
`f_rng`	The RNG function. Needed for PKCS#1 v2.1 encoding and for *MBEDTLS_RSA_PRIVATE*.
`p_rng`	The RNG parameter.
`mode`	*MBEDTLS_RSA_PUBLIC* or *MBEDTLS_RSA_PRIVATE*.
`md_alg`	The message-digest algorithm used to hash the original data. Use MBEDTLS_MD_NONE for signing raw data.
`hashlen`	The length of the message digest. Only used if md_alg is MBEDTLS_MD_NONE .
`hash`	The buffer holding the message digest.
`sig`	The buffer to hold the ciphertext.

Returns:

0 if the signing operation was successful, or an MBEDTLS_ERR_RSA_XXX error code on failure.

int mbedtls_rsa_pkcs1_verify (*mbedtls_rsa_context* * ctx, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, int mode, *mbedtls_md_type_t* md_alg, unsigned int hashlen, const unsigned char * hash, const unsigned char * sig)

This function performs a public RSA operation and checks the message digest.

This is the generic wrapper for performing a PKCS#1 verification using the mode from the context.

***Deprecated*:**

It is deprecated and discouraged to call this function in *MBEDTLS_RSA_PRIVATE* mode. Future versions of the library are likely to remove the mode argument and have it set to *MBEDTLS_RSA_PUBLIC*.

Note

Alternative implementations of RSA need not support mode being set to *MBEDTLS_RSA_PRIVATE* and might instead return *MBEDTLS_ERR_RSA_UNSUPPORTED_OPERATION*.

Note

The sig buffer must be as large as the size of ctx->N . For example, 128 Bytes if RSA-1024 is used.

For PKCS#1 v2.1 encoding, see comments on *mbedtls_rsa_rsassa_pss_verify()* about md_alg and hash_id .

Parameters:

Parameter	Description
`ctx`	The RSA public key context.
`f_rng`	The RNG function. Only needed for *MBEDTLS_RSA_PRIVATE*.
`p_rng`	The RNG parameter.
`mode`	*MBEDTLS_RSA_PUBLIC* or *MBEDTLS_RSA_PRIVATE*.
`md_alg`	The message-digest algorithm used to hash the original data. Use MBEDTLS_MD_NONE for signing raw data.
`hashlen`	The length of the message digest. Only used if md_alg is MBEDTLS_MD_NONE .
`hash`	The buffer holding the message digest.
`sig`	The buffer holding the ciphertext.

Returns:

0 if the verify operation was successful, or an MBEDTLS_ERR_RSA_XXX error code on failure.

int mbedtls_rsa_private (*mbedtls_rsa_context* * ctx, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, const unsigned char * input, unsigned char * output)

This function performs an RSA private key operation.

Note

The input and output buffers must be large enough. For example, 128 Bytes if RSA-1024 is used.

Parameters:

Parameter	Description
`ctx`	The RSA context.
`f_rng`	The RNG function. Needed for blinding.
`p_rng`	The RNG parameter.
`input`	The input buffer.
`output`	The output buffer.

Returns:

0 on success, or an MBEDTLS_ERR_RSA_XXX error code on failure.

int mbedtls_rsa_public (*mbedtls_rsa_context* * ctx, const unsigned char * input, unsigned char * output)

This function performs an RSA public key operation.

Note

This function does not handle message padding.

Make sure to set input [0] = 0 or ensure that input is smaller than N .

The input and output buffers must be large enough. For example, 128 Bytes if RSA-1024 is used.

Parameters:

Parameter	Description
`ctx`	The RSA context.
`input`	The input buffer.
`output`	The output buffer.

Returns:

0 on success, or an MBEDTLS_ERR_RSA_XXX error code on failure.

int mbedtls_rsa_rsaes_oaep_decrypt (*mbedtls_rsa_context* * ctx, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, int mode, const unsigned char * label, size_t label_len, size_t * olen, const unsigned char * input, unsigned char * output, size_t output_max_len)

This function performs a PKCS#1 v2.1 OAEP decryption operation (RSAES-OAEP-DECRYPT).

***Deprecated*:**

It is deprecated and discouraged to call this function in *MBEDTLS_RSA_PUBLIC* mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to *MBEDTLS_RSA_PRIVATE*.

Note

Alternative implementations of RSA need not support mode being set to *MBEDTLS_RSA_PUBLIC* and might instead return *MBEDTLS_ERR_RSA_UNSUPPORTED_OPERATION*.

Note

The output buffer length output_max_len should be as large as the size ctx->len of ctx->N , for example, 128 Bytes if RSA-1024 is used, to be able to hold an arbitrary decrypted message. If it is not large enough to hold the decryption of the particular ciphertext provided, the function returns *MBEDTLS_ERR_RSA_OUTPUT_TOO_LARGE*.

The input buffer must be as large as the size of ctx->N . For example, 128 Bytes if RSA-1024 is used.

Parameters:

Parameter	Description
`ctx`	The RSA context.
`f_rng`	The RNG function. Only needed for *MBEDTLS_RSA_PRIVATE*.
`p_rng`	The RNG parameter.
`mode`	*MBEDTLS_RSA_PUBLIC* or *MBEDTLS_RSA_PRIVATE*.
`label`	The buffer holding the custom label to use.
`label_len`	The length of the label.

Parameter	Description
`olen`	The length of the plaintext.
`input`	The buffer holding the encrypted data.
`output`	The buffer to hold the plaintext.
`output_max_len`	The maximum length of the output buffer.

Returns:

0 on success, or an MBEDTLS_ERR_RSA_XXX error code on failure.

int mbedtls_rsa_rsaes_oaep_encrypt (mbedtls_rsa_context * ctx, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, int mode, const unsigned char * label, size_t label_len, size_t ilen, const unsigned char * input, unsigned char * output)

This function performs a PKCS#1 v2.1 OAEP encryption operation (RSAES-OAEP-ENCRYPT).

***Deprecated*:**

It is deprecated and discouraged to call this function in *MBEDTLS_RSA_PRIVATE* mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to *MBEDTLS_RSA_PUBLIC*.

Note

Alternative implementations of RSA need not support mode being set to *MBEDTLS_RSA_PRIVATE* and might instead return *MBEDTLS_ERR_RSA_UNSUPPORTED_OPERATION*.

Note

The output buffer must be as large as the size of ctx->N. For example, 128 Bytes if RSA-1024 is used.

Parameters:

Parameter	Description
`ctx`	The RSA context.
`f_rng`	The RNG function. Needed for padding and PKCS#1 v2.1 encoding and *MBEDTLS_RSA_PRIVATE*.
`p_rng`	The RNG parameter.
`mode`	*MBEDTLS_RSA_PUBLIC* or *MBEDTLS_RSA_PRIVATE*.
`label`	The buffer holding the custom label to use.
`label_len`	The length of the label.
`ilen`	The length of the plaintext.
`input`	The buffer holding the data to encrypt.
`output`	The buffer used to hold the ciphertext.

Returns:

0 on success, or an MBEDTLS_ERR_RSA_XXX error code on failure.

int mbedtls_rsa_rsaes_pkcs1_v15_decrypt (*mbedtls_rsa_context* * ctx, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, int mode, size_t * olen, const unsigned char * input, unsigned char * output, size_t output_max_len)

This function performs a PKCS#1 v1.5 decryption operation (RSAES-PKCS1-v1_5-DECRYPT).

***Deprecated*:**

It is deprecated and discouraged to call this function in *MBEDTLS_RSA_PUBLIC* mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to *MBEDTLS_RSA_PRIVATE*.

Note

Alternative implementations of RSA need not support mode being set to *MBEDTLS_RSA_PUBLIC* and might instead return *MBEDTLS_ERR_RSA_UNSUPPORTED_OPERATION*.

Note

The output buffer length output_max_len should be as large as the size ctx->len of ctx->N , for example, 128 Bytes if RSA-1024 is used, to be able to hold an arbitrary decrypted message. If it is not large enough to hold the decryption of the particular ciphertext provided, the function returns *MBEDTLS_ERR_RSA_OUTPUT_TOO_LARGE*.

The input buffer must be as large as the size of ctx->N . For example, 128 Bytes if RSA-1024 is used.

Parameters:

Parameter	Description
`ctx`	The RSA context.
`f_rng`	The RNG function. Only needed for *MBEDTLS_RSA_PRIVATE*.
`p_rng`	The RNG parameter.
`mode`	*MBEDTLS_RSA_PUBLIC* or *MBEDTLS_RSA_PRIVATE*.
`olen`	The length of the plaintext.
`input`	The buffer holding the encrypted data.
`output`	The buffer to hold the plaintext.
`output_max_len`	The maximum length of the output buffer.

Returns:

0 on success, or an MBEDTLS_ERR_RSA_XXX error code on failure.

int mbedtls_rsa_rsaes_pkcs1_v15_encrypt (*mbedtls_rsa_context* * ctx, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, int mode, size_t ilen, const unsigned char * input, unsigned char * output)

This function performs a PKCS#1 v1.5 encryption operation (RSAES-PKCS1-v1_5-ENCRYPT).

***Deprecated*:**

It is deprecated and discouraged to call this function in *MBEDTLS_RSA_PRIVATE* mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to *MBEDTLS_RSA_PUBLIC*.

Note

Alternative implementations of RSA need not support mode being set to *MBEDTLS_RSA_PRIVATE* and might instead return *MBEDTLS_ERR_RSA_UNSUPPORTED_OPERATION*.

Note

The output buffer must be as large as the size of ctx->N . For example, 128 Bytes if RSA-1024 is used.

Parameters:

Parameter	Description
`ctx`	The RSA context.
`f_rng`	The RNG function. Needed for padding and *MBEDTLS_RSA_PRIVATE*.
`p_rng`	The RNG parameter.
`mode`	*MBEDTLS_RSA_PUBLIC* or *MBEDTLS_RSA_PRIVATE*.
`ilen`	The length of the plaintext.
`input`	The buffer holding the data to encrypt.
`output`	The buffer used to hold the ciphertext.

Returns:

0 on success, or an MBEDTLS_ERR_RSA_XXX error code on failure.

int mbedtls_rsa_rsassa_pkcs1_v15_sign (*mbedtls_rsa_context* * ctx, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, int mode, *mbedtls_md_type_t* md_alg, unsigned int hashlen, const unsigned char * hash, unsigned char * sig)

This function performs a PKCS#1 v1.5 signature operation (RSASSA-PKCS1-v1_5-SIGN).

***Deprecated*:**

It is deprecated and discouraged to call this function in *MBEDTLS_RSA_PUBLIC* mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to *MBEDTLS_RSA_PRIVATE*.

Note

Alternative implementations of RSA need not support mode being set to *MBEDTLS_RSA_PUBLIC* and might instead return *MBEDTLS_ERR_RSA_UNSUPPORTED_OPERATION*.

Note

The sig buffer must be as large as the size of ctx->N . For example, 128 Bytes if RSA-1024 is used.

Parameters:

Parameter	Description
`ctx`	The RSA context.
`f_rng`	The RNG function. Only needed for *MBEDTLS_RSA_PRIVATE*.
`p_rng`	The RNG parameter.
`mode`	*MBEDTLS_RSA_PUBLIC* or *MBEDTLS_RSA_PRIVATE*.

Parameter	Description
`md_alg`	The message-digest algorithm used to hash the original data. Use MBEDTLS_MD_NONE for signing raw data.
`hashlen`	The length of the message digest. Only used if md_alg is MBEDTLS_MD_NONE .
`hash`	The buffer holding the message digest.
`sig`	The buffer to hold the ciphertext.

Returns:

0 if the signing operation was successful, or an MBEDTLS_ERR_RSA_XXX error code on failure.

int mbedtls_rsa_rsassa_pkcs1_v15_verify (*mbedtls_rsa_context* * ctx, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, int mode, *mbedtls_md_type_t* md_alg, unsigned int hashlen, const unsigned char * hash, const unsigned char * sig)

This function performs a PKCS#1 v1.5 verification operation (RSASSA-PKCS1-v1_5-VERIFY).

***Deprecated*:**

It is deprecated and discouraged to call this function in *MBEDTLS_RSA_PRIVATE* mode. Future versions of the library are likely to remove the mode argument and have it set to *MBEDTLS_RSA_PUBLIC*.

Note

Alternative implementations of RSA need not support mode being set to *MBEDTLS_RSA_PRIVATE* and might instead return *MBEDTLS_ERR_RSA_UNSUPPORTED_OPERATION*.

Note

The sig buffer must be as large as the size of ctx->N . For example, 128 Bytes if RSA-1024 is used.

Parameters:

Parameter	Description
`ctx`	The RSA public key context.
`f_rng`	The RNG function. Only needed for *MBEDTLS_RSA_PRIVATE*.
`p_rng`	The RNG parameter.
`mode`	*MBEDTLS_RSA_PUBLIC* or *MBEDTLS_RSA_PRIVATE*.
`md_alg`	The message-digest algorithm used to hash the original data. Use MBEDTLS_MD_NONE for signing raw data.
`hashlen`	The length of the message digest. Only used if md_alg is MBEDTLS_MD_NONE .
`hash`	The buffer holding the message digest.
`sig`	The buffer holding the ciphertext.

Returns:

0 if the verify operation was successful, or an MBEDTLS_ERR_RSA_XXX error code on failure.

int mbedtls_rsa_rsassa_pss_sign (mbedtls_rsa_context * ctx, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, int mode, mbedtls_md_type_t md_alg, unsigned int hashlen, const unsigned char * hash, unsigned char * sig)

This function performs a PKCS#1 v2.1 PSS signature operation (RSASSA-PSS-SIGN).

Deprecated:

It is deprecated and discouraged to call this function in *MBEDTLS_RSA_PUBLIC* mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to *MBEDTLS_RSA_PRIVATE*.

Note

Alternative implementations of RSA need not support mode being set to *MBEDTLS_RSA_PUBLIC* and might instead return *MBEDTLS_ERR_RSA_UNSUPPORTED_OPERATION*.

Note

The sig buffer must be as large as the size of ctx->N . For example, 128 Bytes if RSA-1024 is used.

The hash_id in the RSA context is the one used for the encoding. md_alg in the function call is the type of hash that is encoded. According to RFC-3447: Public-Key Cryptography Standards (PKCS) #1 v2.1: RSA Cryptography Specifications it is advised to keep both hashes the same.

Parameters:

Parameter	Description
`ctx`	The RSA context.
`f_rng`	The RNG function. Needed for PKCS#1 v2.1 encoding and for *MBEDTLS_RSA_PRIVATE*.
`p_rng`	The RNG parameter.
`mode`	*MBEDTLS_RSA_PUBLIC* or *MBEDTLS_RSA_PRIVATE*.
`md_alg`	The message-digest algorithm used to hash the original data. Use MBEDTLS_MD_NONE for signing raw data.
`hashlen`	The length of the message digest. Only used if md_alg is MBEDTLS_MD_NONE .
`hash`	The buffer holding the message digest.
`sig`	The buffer to hold the ciphertext.

Returns:

0 if the signing operation was successful, or an MBEDTLS_ERR_RSA_XXX error code on failure.

int mbedtls_rsa_rsassa_pss_verify (mbedtls_rsa_context * ctx, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, int mode, mbedtls_md_type_t md_alg, unsigned int hashlen, const unsigned char * hash, const unsigned char * sig)

This function performs a PKCS#1 v2.1 PSS verification operation (RSASSA-PSS-VERIFY).

The hash function for the MGF mask generating function is that specified in the RSA context.

Deprecated:

It is deprecated and discouraged to call this function in *MBEDTLS_RSA_PRIVATE* mode. Future versions of the library are likely to remove the mode argument and have it implicitly set to *MBEDTLS_RSA_PUBLIC*.

Note

Alternative implementations of RSA need not support mode being set to *MBEDTLS_RSA_PRIVATE* and might instead return *MBEDTLS_ERR_RSA_UNSUPPORTED_OPERATION*.

Note

The sig buffer must be as large as the size of ctx->N . For example, 128 Bytes if RSA-1024 is used.

The hash_id in the RSA context is the one used for the verification. md_alg in the function call is the type of hash that is verified. According to RFC-3447: Public-Key Cryptography Standards (PKCS) #1 v2.1: RSA Cryptography Specifications it is advised to keep both hashes the same. If hash_id in the RSA context is unset, the md_alg from the function call is used.

Parameters:

Parameter	Description
`ctx`	The RSA public key context.
`f_rng`	The RNG function. Only needed for *MBEDTLS_RSA_PRIVATE*.
`p_rng`	The RNG parameter.
`mode`	*MBEDTLS_RSA_PUBLIC* or *MBEDTLS_RSA_PRIVATE*.
`md_alg`	The message-digest algorithm used to hash the original data. Use MBEDTLS_MD_NONE for signing raw data.
`hashlen`	The length of the message digest. Only used if md_alg is MBEDTLS_MD_NONE .
`hash`	The buffer holding the message digest.
`sig`	The buffer holding the ciphertext.

Returns:

0 if the verify operation was successful, or an MBEDTLS_ERR_RSA_XXX error code on failure.

int mbedtls_rsa_rsassa_pss_verify_ext (*mbedtls_rsa_context* * ctx, int(*)(void *, unsigned char *, size_t) f_rng, void * p_rng, int mode, *mbedtls_md_type_t* md_alg, unsigned int hashlen, const unsigned char * hash, *mbedtls_md_type_t* mgf1_hash_id, int expected_salt_len, const unsigned char * sig)

This function performs a PKCS#1 v2.1 PSS verification operation (RSASSA-PSS-VERIFY).

The hash function for the MGF mask generating function is that specified in mgf1_hash_id .

The sig buffer must be as large as the size of ctx->N . For example, 128 Bytes if RSA-1024 is used.

The hash_id in the RSA context is ignored.

Parameters:

Parameter	Description
`ctx`	The RSA public key context.
`f_rng`	The RNG function. Only needed for *MBEDTLS_RSA_PRIVATE*.
`p_rng`	The RNG parameter.
`mode`	*MBEDTLS_RSA_PUBLIC* or *MBEDTLS_RSA_PRIVATE*.
`md_alg`	The message-digest algorithm used to hash the original data. Use MBEDTLS_MD_NONE for signing raw data.
`hashlen`	The length of the message digest. Only used if md_alg is MBEDTLS_MD_NONE .
`hash`	The buffer holding the message digest.
`mgf1_hash_id`	The message digest used for mask generation.
`expected_salt_len`	The length of the salt used in padding. Use #MBEDTLS_RSA_SALT_LEN_ANY to accept any salt length.
`sig`	The buffer holding the ciphertext.

Returns:

0 if the verify operation was successful, or an MBEDTLS_ERR_RSA_XXX error code on failure.

Note

int mbedtls_rsa_self_test (int verbose)

The RSA checkup routine.

Returns:

0 on success, or 1 on failure.

void mbedtls_rsa_set_padding (*mbedtls_rsa_context* * ctx, int padding, int hash_id)

This function sets padding for an already initialized RSA context. See *mbedtls_rsa_init()* for details.

Parameters:

Parameter	Description
`ctx`	The RSA context to be set.
`padding`	Selects padding mode: *MBEDTLS_RSA_PKCS_V15* or *MBEDTLS_RSA_PKCS_V21*.
`hash_id`	The *MBEDTLS_RSA_PKCS_V21* hash identifier.

1.7.75 secureboot_basetypes.h File Reference

This file contains basic type definitions for the Secure Boot.

#include "cc_pal_types.h"

#include "cc_pal_types_plat.h"

Detailed description

This file contains basic type definitions for the Secure Boot.

1.7.76 secureboot_defs.h File Reference

This file contains type definitions for the Secure Boot.

#include "cc_crypto_boot_defs.h"

#include "cc_sec_defs.h"

Data structures

- struct *CCSbCertInfo_t*

Macros

- #define *SW_REC_SIGNED_DATA_SIZE_IN_BYTES* 44
- #define *SW_REC_NONE_SIGNED_DATA_SIZE_IN_BYTES* 8
- #define *SW_REC_NONE_SIGNED_DATA_SIZE_IN_WORDS* *SW_REC_NONE_SIGNED_DATA_SIZE_IN_BYTES*/*CC_32BIT_WORD_SIZE*
- #define *CC_SW_COMP_NO_MEM_LOAD_INDICATION* 0xFFFFFFFFUL

Detailed description

This file contains type definitions for the Secure Boot.

1.7.77 secureboot_gen_defs.h File Reference

This file contains all of the definitions and structures used for the Secure Boot and Secure Debug.

#include "secureboot_basetypes.h"

#include "cc_sec_defs.h"

#include "cc_pal_sb_plat.h"

Macros

- #define *CC_SB_MAX_SIZE_ADDITIONAL_DATA_BYTES* 128

Typedefs

- typedef uint32_t *CCSbCertPubKeyHash_t*[HASH_RESULT_SIZE_IN_WORDS]
- typedef uint32_t *CCSbCertSocId_t*[HASH_RESULT_SIZE_IN_WORDS]
- typedef uint32_t(* *CCSbFlashReadFunc*) (*CCAddr_t* flashAddress, uint8_t *memDst, uint32_t sizeToRead, void *context)

  Typedef of the Flash read function pointer, to be implemented by the partner.
- typedef uint32_t(* *CCBsvFlashWriteFunc*) (*CCAddr_t* flashAddress, uint8_t *memSrc, uint32_t sizeToWrite, void *context)

Detailed description

This file contains all of the definitions and structures used for the Secure Boot and Secure Debug.

1.7.78 sha1.h File Reference

The SHA-1 cryptographic hash function.

#include "config.h"

#include <stddef.h>

#include <stdint.h>

Data structures

- struct *mbedtls_sha1_context*

The SHA-1 context structure. Macros

- #define *MBEDTLS_ERR_SHA1_HW_ACCEL_FAILED* -0x0035
- #define MBEDTLS_DEPRECATED
- #define MBEDTLS_DEPRECATED

Functions

- void *mbedtls_sha1_init* (*mbedtls_sha1_context* *ctx)

  This function initializes a SHA-1 context.
- void *mbedtls_sha1_free* (*mbedtls_sha1_context* *ctx)

  This function clears a SHA-1 context.
- void *mbedtls_sha1_clone* (*mbedtls_sha1_context* *dst, const *mbedtls_sha1_context* *src)

  This function clones the state of a SHA-1 context.
- int *mbedtls_sha1_starts_ret* (*mbedtls_sha1_context* *ctx)

  This function starts a SHA-1 checksum calculation.
- int *mbedtls_sha1_update_ret* (*mbedtls_sha1_context* *ctx, const unsigned char *input, size_t ilen)

  This function feeds an input buffer into an ongoing SHA-1 checksum calculation.
- int *mbedtls_sha1_finish_ret* (*mbedtls_sha1_context* *ctx, unsigned char output[20])

  This function finishes the SHA-1 operation, and writes the result to the output buffer.
- int *mbedtls_internal_sha1_process* (*mbedtls_sha1_context* *ctx, const unsigned char data[64])

  This function processes a single data block within the ongoing SHA-1 computation. This function is for internal use only.
- int *mbedtls_sha1_ret* (const unsigned char *input, size_t ilen, unsigned char output[20])

  This function calculates the SHA-1 checksum of a buffer.
- int *mbedtls_sha1_self_test* (int verbose)

  The SHA-1 checkup routine.

Detailed description

The SHA-1 cryptographic hash function.

Warning

SHA-1 is considered a weak message digest and its use constitutes a security risk. We recommend considering stronger message digests instead.

Macro definition documentation

#define MBEDTLS_ERR_SHA1_HW_ACCEL_FAILED -0x0035

SHA-1 hardware accelerator failed

Function documentation

int mbedtls_internal_sha1_process (*mbedtls_sha1_context* * ctx, const unsigned char data[64])

This function processes a single data block within the ongoing SHA-1 computation. This function is for internal use only.

Warning

SHA-1 is considered a weak message digest and its use constitutes a security risk. We recommend considering stronger message digests instead.

Parameters:

Parameter	Description
`ctx`	The SHA-1 context.
`data`	The buffer holding one block of data.

Returns:

0 on success.

void mbedtls_sha1_clone (*mbedtls_sha1_context* * dst, const *mbedtls_sha1_context* * src)

This function clones the state of a SHA-1 context.

Warning

SHA-1 is considered a weak message digest and its use constitutes a security risk. We recommend considering stronger message digests instead.

Parameters:

Parameter	Description
`dst`	The destination context.
`src`	The context to clone.

int mbedtls_sha1_finish_ret (*mbedtls_sha1_context* * ctx, unsigned char output[20])

This function finishes the SHA-1 operation, and writes the result to the output buffer.

Warning

SHA-1 is considered a weak message digest and its use constitutes a security risk. We recommend considering stronger message digests instead.

Parameters:

Parameter	Description
`ctx`	The SHA-1 context.
`output`	The SHA-1 checksum result.

Returns:

0 on success.

void mbedtls_sha1_free (*mbedtls_sha1_context* * ctx)

This function clears a SHA-1 context.

Warning

SHA-1 is considered a weak message digest and its use constitutes a security risk. We recommend considering stronger message digests instead.

Parameters:

Parameter	Description
`ctx`	The SHA-1 context to clear.

void mbedtls_sha1_init (*mbedtls_sha1_context* * ctx)

This function initializes a SHA-1 context.

Warning

SHA-1 is considered a weak message digest and its use constitutes a security risk. We recommend considering stronger message digests instead.

Parameters:

Parameter	Description
`ctx`	The SHA-1 context to initialize.

int mbedtls_sha1_ret (const unsigned char * input, size_t ilen, unsigned char output[20])

This function calculates the SHA-1 checksum of a buffer.

The function allocates the context, performs the calculation, and frees the context.

The SHA-1 result is calculated as output = SHA-1(input buffer).

Warning

SHA-1 is considered a weak message digest and its use constitutes a security risk. We recommend considering stronger message digests instead.

Parameters:

Parameter	Description
`input`	The buffer holding the input data.

Parameter	Description
`ilen`	The length of the input data.
`output`	The SHA-1 checksum result.

Returns:

0 on success.

int mbedtls_sha1_self_test (int verbose)

The SHA-1 checkup routine.

Warning

SHA-1 is considered a weak message digest and its use constitutes a security risk. We recommend considering stronger message digests instead.

Returns:

0 on success, or 1 on failure.

int mbedtls_sha1_starts_ret (*mbedtls_sha1_context* * ctx)

This function starts a SHA-1 checksum calculation.

Warning

SHA-1 is considered a weak message digest and its use constitutes a security risk. We recommend considering stronger message digests instead.

Parameters:

Parameter	Description
`ctx`	The context to initialize.

Returns:

0 on success.

int mbedtls_sha1_update_ret (*mbedtls_sha1_context* * ctx, const unsigned char * input, size_t ilen)

This function feeds an input buffer into an ongoing SHA-1 checksum calculation.

Warning

SHA-1 is considered a weak message digest and its use constitutes a security risk. We recommend considering stronger message digests instead.

Parameters:

Parameter	Description
`ctx`	The SHA-1 context.
`input`	The buffer holding the input data.
`ilen`	The length of the input data.

Returns:

0 on success.

1.7.79 sha256.h File Reference

The SHA-224 and SHA-256 cryptographic hash function.

#include "config.h"

#include <stddef.h>

#include <stdint.h>

Data structures

- struct *mbedtls_sha256_context*

The SHA-256 context structure. Macros

- #define *MBEDTLS_ERR_SHA256_HW_ACCEL_FAILED* -0x0037
- #define MBEDTLS_DEPRECATED
- #define MBEDTLS_DEPRECATED

Functions

- void *mbedtls_sha256_init* (*mbedtls_sha256_context* *ctx)

  This function initializes a SHA-256 context.
- void *mbedtls_sha256_free* (*mbedtls_sha256_context* *ctx)

  This function clears a SHA-256 context.
- void *mbedtls_sha256_clone* (*mbedtls_sha256_context* *dst, const *mbedtls_sha256_context* *src)

  This function clones the state of a SHA-256 context.
- int *mbedtls_sha256_starts_ret* (*mbedtls_sha256_context* *ctx, int is224)

  This function starts a SHA-224 or SHA-256 checksum calculation.
- int *mbedtls_sha256_update_ret* (*mbedtls_sha256_context* *ctx, const unsigned char *input, size_t ilen)

  This function feeds an input buffer into an ongoing SHA-256 checksum calculation.
- int *mbedtls_sha256_finish_ret* (*mbedtls_sha256_context* *ctx, unsigned char output[32])

  This function finishes the SHA-256 operation, and writes the result to the output buffer.
- int *mbedtls_internal_sha256_process* (*mbedtls_sha256_context* *ctx, const unsigned char data[64])

  This function processes a single data block within the ongoing SHA-256 computation. This function is for internal use only.
- int *mbedtls_sha256_ret* (const unsigned char *input, size_t ilen, unsigned char output[32], int is224)

  This function calculates the SHA-224 or SHA-256 checksum of a buffer.
- int *mbedtls_sha256_self_test* (int verbose)

  The SHA-224 and SHA-256 checkup routine.

Detailed description

The SHA-224 and SHA-256 cryptographic hash function.

Macro definition documentation

#define MBEDTLS_ERR_SHA256_HW_ACCEL_FAILED -0x0037

SHA-256 hardware accelerator failed

Function documentation

int mbedtls_internal_sha256_process (*mbedtls_sha256_context* * ctx, const unsigned char data[64])

This function processes a single data block within the ongoing SHA-256 computation. This function is for internal use only.

Parameters:

Parameter	Description
`ctx`	The SHA-256 context.
`data`	The buffer holding one block of data.

Returns:

0 on success.

void mbedtls_sha256_clone (*mbedtls_sha256_context* * dst, const *mbedtls_sha256_context* * src)

This function clones the state of a SHA-256 context.

Parameters:

Parameter	Description
`dst`	The destination context.
`src`	The context to clone.

int mbedtls_sha256_finish_ret (*mbedtls_sha256_context* * ctx, unsigned char output[32])

This function finishes the SHA-256 operation, and writes the result to the output buffer.

Parameters:

Parameter	Description
`ctx`	The SHA-256 context.
`output`	The SHA-224 or SHA-256 checksum result.

Returns:

0 on success.

void mbedtls_sha256_free (*mbedtls_sha256_context* * ctx)

This function clears a SHA-256 context.

Parameters:

Parameter	Description
`ctx`	The SHA-256 context to clear.

void mbedtls_sha256_init (*mbedtls_sha256_context* * ctx)

This function initializes a SHA-256 context.

Parameters:

Parameter	Description
`ctx`	The SHA-256 context to initialize.

int mbedtls_sha256_ret (const unsigned char * input, size_t ilen, unsigned char output[32], int is224)

This function calculates the SHA-224 or SHA-256 checksum of a buffer.

The function allocates the context, performs the calculation, and frees the context.

The SHA-256 result is calculated as output = SHA-256(input buffer).

Parameters:

Parameter	Description
`input`	The buffer holding the input data.
`ilen`	The length of the input data.
`output`	The SHA-224 or SHA-256 checksum result.
`is224`	Determines which function to use. ● 0: Use SHA-256. ● 1: Use SHA-224.

int mbedtls_sha256_self_test (int verbose)

The SHA-224 and SHA-256 checkup routine.

Returns:

0 on success, or 1 on failure.

int mbedtls_sha256_starts_ret (*mbedtls_sha256_context* * ctx, int is224)

This function starts a SHA-224 or SHA-256 checksum calculation.

Parameters:

Parameter	Description
`ctx`	The context to initialize.
`is224`	Determines which function to use. ● 0: Use SHA-256. ● 1: Use SHA-224.

Returns:

0 on success.

int mbedtls_sha256_update_ret (*mbedtls_sha256_context* * ctx, const unsigned char * input, size_t ilen)

This function feeds an input buffer into an ongoing SHA-256 checksum calculation.

Parameters:

Parameter	Description
ctx	The SHA-256 context to initialize.
input	The buffer holding the data.
ilen	The length of the input data.

Returns:

0	on success.

1.7.80 sha512.h File Reference

The SHA-384 and SHA-512 cryptographic hash function.

#include "config.h"

#include <stddef.h>

#include <stdint.h>

Data structures

- struct *mbedtls_sha512_context*

The SHA-512 context structure. Macros

- #define *MBEDTLS_ERR_SHA512_HW_ACCEL_FAILED* -0x0039
- #define MBEDTLS_DEPRECATED
- #define MBEDTLS_DEPRECATED

Functions

- void *mbedtls_sha512_init* (*mbedtls_sha512_context* *ctx)

  This function initializes a SHA-512 context.
- void *mbedtls_sha512_free* (*mbedtls_sha512_context* *ctx)

  This function clears a SHA-512 context.
- void *mbedtls_sha512_clone* (*mbedtls_sha512_context* *dst, const *mbedtls_sha512_context* *src)

  This function clones the state of a SHA-512 context.
- int *mbedtls_sha512_starts_ret* (*mbedtls_sha512_context* *ctx, int is384)

  This function starts a SHA-384 or SHA-512 checksum calculation.
- int *mbedtls_sha512_update_ret* (*mbedtls_sha512_context* *ctx, const unsigned char *input, size_t ilen)

  This function feeds an input buffer into an ongoing SHA-512 checksum calculation.
- int *mbedtls_sha512_finish_ret* (*mbedtls_sha512_context* *ctx, unsigned char output[64])

  This function finishes the SHA-512 operation, and writes the result to the output buffer. This function is for internal use only.

- int *mbedtls_internal_sha512_process* (*mbedtls_sha512_context* *ctx, const unsigned char data[128])

  This function processes a single data block within the ongoing SHA-512 computation.
- int *mbedtls_sha512_ret* (const unsigned char *input, size_t ilen, unsigned char output[64], int is384)

  This function calculates the SHA-512 or SHA-384 checksum of a buffer.
- int *mbedtls_sha512_self_test* (int verbose)

  The SHA-384 or SHA-512 checkup routine.

Detailed description

The SHA-384 and SHA-512 cryptographic hash function.

Macro definition documentation

#define MBEDTLS_ERR_SHA512_HW_ACCEL_FAILED -0x0039

SHA-512 hardware accelerator failed

Function documentation

int mbedtls_internal_sha512_process (*mbedtls_sha512_context* * ctx, const unsigned char data[128])

This function processes a single data block within the ongoing SHA-512 computation.

Parameters:

Parameter	Description
`ctx`	The SHA-512 context.
`data`	The buffer holding one block of data.

Returns:

0 on success.

void mbedtls_sha512_clone (*mbedtls_sha512_context* * dst, const *mbedtls_sha512_context* * src)

This function clones the state of a SHA-512 context.

Parameters:

Parameter	Description
`dst`	The destination context.
`src`	The context to clone.

int mbedtls_sha512_finish_ret (*mbedtls_sha512_context* * ctx, unsigned char output[64])

This function finishes the SHA-512 operation, and writes the result to the output buffer. This function is for internal use only.

Parameters:

Parameter	Description
`ctx`	The SHA-512 context.

Parameter	Description
`output`	The SHA-384 or SHA-512 checksum result.

Returns:

0 on success.

void mbedtls_sha512_free (*mbedtls_sha512_context* * ctx)

This function clears a SHA-512 context.

Parameters:

Parameter	Description
`ctx`	The SHA-512 context to clear.

void mbedtls_sha512_init (*mbedtls_sha512_context* * ctx)

This function initializes a SHA-512 context.

Parameters:

Parameter	Description
`ctx`	The SHA-512 context to initialize.

int mbedtls_sha512_ret (const unsigned char * input, size_t ilen, unsigned char output[64], int is384)

This function calculates the SHA-512 or SHA-384 checksum of a buffer.

The function allocates the context, performs the calculation, and frees the context.

The SHA-512 result is calculated as output = SHA-512(input buffer).

Parameters:

Parameter	Description
`input`	The buffer holding the input data.
`ilen`	The length of the input data.
`output`	The SHA-384 or SHA-512 checksum result.
`is384`	Determines which function to use. • 0: Use SHA-512. • 1: Use SHA-384.

Returns:

0 on success.

int mbedtls_sha512_self_test (int verbose)

The SHA-384 or SHA-512 checkup routine.

Returns:

0 on success, or 1 on failure.

int mbedtls_sha512_starts_ret (*mbedtls_sha512_context* * ctx, int is384)

This function starts a SHA-384 or SHA-512 checksum calculation.

Parameters:

Parameter	Description
`ctx`	The SHA-512 context to initialize.
`is384`	Determines which function to use. • 0: Use SHA-512. • 1: Use SHA-384.

Returns:

0 on success.

int mbedtls_sha512_update_ret (mbedtls_sha512_context * ctx, const unsigned char * input, size_t ilen)

This function feeds an input buffer into an ongoing SHA-512 checksum calculation.

Parameters:

Parameter	Description
`ctx`	The SHA-512 context.
`input`	The buffer holding the input data.
`ilen`	The length of the input data.

Returns:

0 on success.

2 SBROM APIs

This chapter provides an overview of the Arm® TrustZone® CryptoCell-312 SBROM APIs.

2.1 Boot services modules

Here is a list of all modules:

- CryptoCell PAL platform dependent types
- CryptoCell PAL DMA specific definitions

2.2 Boot services data structures

Here are the data structures with brief descriptions:

- *CCSbCertInfo_t*
- *CCSbCertParserSwCompsInfo_t*
- *CCSbSwVersion_t*

2.3 Boot services file list

Here is a list of all documented files with brief descriptions:

Table 2-1: Boot services file list

Filename	Description
bootimagesverifier_api.h	This file contains the set of Secure Boot APIs that are supported by the SBROM library.
bootimagesverifier_def.h	This file contains all definitions that are used for the SBROM service APIs.
bootimagesverifier_error.h	This file contains error code definitions that are used for the SBROM service APIs.
bsv_api.h	This file contains all the SBROM library APIs and definitions.
bsv_crypto_api.h	This file defines crypto ROM APIs: SH256, CMAC KDF, and CCM.
bsv_crypto_defs.h	This file contains crypto ROM API definitions.
bsv_defs.h	This file contains definitions that are used for the Secure Boot ROM service APIs.
bsv_error.h	This file defines the error return code types of the error codes from Secure Boot APIs.
bsv_otp_api.h	This file contains functions that access the OTP memory for read and write operations.
cc_bitops.h	This file defines bit fields operations macros.
cc_crypto_boot_defs.h	This file contains SBROM definitions.
cc_hal_sb.h	This file contains the functions that are used for the SBROM HAL layer.
cc_hal_sb_plat.h	This file contains definitions that are used for the SBROM HAL layer.
cc_pal_dma_defs.h	This file contains the platform-dependent DMA definitions.
cc_pal_sb_plat.h	This file contains the platform dependent definitions that are used in the SBROM code.
cc_pal_types.h	This file contains platform-dependent definitions and types.
cc_pal_types_plat.h	This file contains basic type definitions that are platform-dependent.
cc_sec_defs.h	This file contains general hash definitions and types.
sbrom_sec_debug_api.h	This file contains a secure debug function that defines the allowed debug domains.
secureboot_basetypes.h	This file contains basic type definitions for the Secure Boot.
secureboot_defs.h	This file contains basic type definitions for the Secure Boot in TrustZone platform.
secureboot_error.h	This file defines the error code types returned from the Secure Boot code.
secureboot_gen_defs.h	This file contains all of the definitions and structures that are used for the Secure Boot.

2.4 Boot services module documentation

2.4.1 CryptoCell PAL platform dependent types

Macros

- #define *CC_SUCCESS* 0UL
- #define *CC_FAIL* 1UL
- #define *CC_OK* 0
- #define *CC_UNUSED_PARAM*(prm) ((void)prm)
- #define *CC_MAX_UINT32_VAL* (0xFFFFFFFF)
- #define *CC_MIN*(a, b) (((a) < (b)) ? (a) : (b))
- #define *CC_MAX*(a, b) (((a) > (b)) ? (a) : (b))
- #define *CALC_FULL_BYTES*(numBits) ((numBits)/*CC_BITS_IN_BYTE* + (((numBits) & (*CC_BITS_IN_BYTE*-1)) > 0))
- #define *CALC_FULL_32BIT_WORDS*(numBits) ((numBits)/*CC_BITS_IN_32BIT_WORD* + (((numBits) & (*CC_BITS_IN_32BIT_WORD*-1)) > 0))
- #define *CALC_32BIT_WORDS_FROM_BYTES*(sizeBytes) ((sizeBytes)/*CC_32BIT_WORD_SIZE* + (((sizeBytes) & (*CC_32BIT_WORD_SIZE*-1)) > 0))
- #define *ROUNDUP_BITS_TO_32BIT_WORD*(numBits) (*CALC_FULL_32BIT_WORDS*(numBits) * *CC_BITS_IN_32BIT_WORD*)
- #define *ROUNDUP_BITS_TO_BYTES*(numBits) (*CALC_FULL_BYTES*(numBits) * *CC_BITS_IN_BYTE*)
- #define *ROUNDUP_BYTES_TO_32BIT_WORD*(sizeBytes) (*CALC_32BIT_WORDS_FROM_BYTES*(sizeBytes) * *CC_32BIT_WORD_SIZE*)

Enumerations

- enum *CCBool* { *CC_FALSE* = 0, *CC_TRUE* = 1 }

Macro definition documentation

#define CALC_32BIT_WORDS_FROM_BYTES(sizeBytes) ((sizeBytes)/*CC_32BIT_WORD_SIZE* + (((sizeBytes) & (*CC_32BIT_WORD_SIZE*-1)) > 0))

Macro that calculates number of full 32bits words from bytes (i.e. 3 bytes are 1 word).

#define CALC_FULL_32BIT_WORDS(numBits) ((numBits)/*CC_BITS_IN_32BIT_WORD* + (((numBits) & (*CC_BITS_IN_32BIT_WORD*-1)) > 0))

Macro that calculates number of full 32bits words from bits (i.e. 31 bits are 1 word).

#define CALC_FULL_BYTES(numBits) ((numBits)/*CC_BITS_IN_BYTE* + (((numBits) & (*CC_BITS_IN_BYTE*-1)) > 0))

Macro that calculates number of full bytes from bits (i.e. 7 bits are 1 byte).

#define CC_FAIL 1UL

Failure definition.

#define CC_MAX(a, b) (((a) > (b)) ? (a) : (b))

Definition for maximum.

#define CC_MAX_UINT32_VAL (0xFFFFFFFF)

Maximal uint32 value.

#define CC_MIN(a, b) (((a) < (b)) ? (a) : (b))

Definition for minimum.

#define CC_OK 0

Success (OK) defintion.

#define CC_SUCCESS 0UL

Success definition.

#define CC_UNUSED_PARAM(prm) ((void)prm)

Macro that handles unused parameters in the code (to avoid compilation warnings).

#define ROUNDUP_BITS_TO_32BIT_WORD(numBits) (*CALC_FULL_32BIT_WORDS*(numBits) * *CC_BITS_IN_32BIT_WORD*)

Macro that round up bits to 32bits words.

#define ROUNDUP_BITS_TO_BYTES(numBits) (*CALC_FULL_BYTES*(numBits) * *CC_BITS_IN_BYTE*)

Macro that round up bits to bytes.

#define ROUNDUP_BYTES_TO_32BIT_WORD(sizeBytes) (*CALC_32BIT_WORDS_FROM_BYTES*(sizeBytes) * *CC_32BIT_WORD_SIZE*)

Macro that round up bytes to 32bits words.

Enumeration type documentation

enum *CCBool*

Boolean definition.

Enumerator:

Enum	Description
CC_FALSE	Boolean false definition.
CC_TRUE	Boolean true definition.

2.4.2 CryptoCell PAL DMA specific definitions

Typedefs

- typedef void * CC_PalDmaBufferHandle

Enumerations

- enum CCPalDmaBufferDirection_t { CC_PAL_DMA_DIR_NONE = 0, CC_PAL_DMA_DIR_TO_DEVICE = 1, CC_PAL_DMA_DIR_FROM_DEVICE = 2, CC_PAL_DMA_DIR_BI_DIRECTION = 3, CC_PAL_DMA_DIR_MAX, CC_PAL_DMA_DIR_RESERVE32 = 0x7FFFFFFF }

Typedef documentation

typedef void* CC_PalDmaBufferHandle

Definition for DMA buffer handle.

Enumeration type documentation

enum CCPalDmaBufferDirection_t

DMA directions configuration

Enumerator:

Enum	Description
`CC_PAL_DMA_DIR_NONE`	No direction.
`CC_PAL_DMA_DIR_TO_DEVICE`	The original buffer is the input to the operation, and should be copied/mapped to a temp buffer, prior to activating the HW on the temp buffer.
`CC_PAL_DMA_DIR_FROM_DEVICE`	The temp buffer holds the output of the HW, and this API should copy/map it to the original output buffer.
`CC_PAL_DMA_DIR_BI_DIRECTION`	Used when the result is written over the original data at the same address. should be treated as CC_PAL_DMA_DIR_TO_DEVICE and CC_PAL_DMA_DIR_FROM_DEVICE.
`CC_PAL_DMA_DIR_MAX`	Maximal DMA directions options.
`CC_PAL_DMA_DIR_RESERVE32`	Reserved.

2.5 Boot services data structure documentation

2.5.1 CCSbCertInfo_t struct reference

```
#include <secureboot_defs.h>
```

Data fields

- uint32_t *otpVersion*
- *CCSbPubKeyIndexType_t keyIndex*
- uint32_t *activeMinSwVersionVal*
- *CCHashResult_t pubKeyHash*
- uint32_t *initDataFlag*

Detailed description

Input/output to the verification API.

Field documentation

uint32_t CCSbCertInfo_t::activeMinSwVersionVal

Holds the SW version value for the certificate-chain.

uint32_t CCSbCertInfo_t::initDataFlag

Initialization indication. Internal flag.

CCSbPubKeyIndexType_t **CCSbCertInfo_t::keyIndex**

Enumeration defining the key hash to retrieve: 128-bit HBK0, 128-bit HBK1, or 256-bit HBK.

uint32_t CCSbCertInfo_t::otpVersion

NV counter saved in OTP.

CCHashResult_t **CCSbCertInfo_t::pubKeyHash**

in/out:

- In: Hash of the public key (N||Np), to compare to the public key stored in the certificate.
- Out: Hash of the public key (N||Np) stored in the certificate, to be used for verification of the public key of the next certificate in the chain.

The documentation for this struct was generated from the following file:

secureboot_defs.h

2.5.2 CCSbCertParserSwCompsInfo_t struct reference

```
#include <cc_crypto_boot_defs.h>
```

Data fields

- uint32_t *numOfSwComps*
- *CCswCodeEncType_t* *swCodeEncType*
- *CCswLoadVerifyScheme_t* *swLoadVerifyScheme*
- *CCswCryptoType_t* *swCryptoType*
- *CCSbNonce_t* *nonce*
- uint32_t * *pSwCompsData*

Detailed description

SW components data.

Field documentation

CCSbNonce_t CCSbCertParserSwCompsInfo_t::nonce

Nonce.

uint32_t CCSbCertParserSwCompsInfo_t::numOfSwComps

Num of SW components.

uint32_t* CCSbCertParserSwCompsInfo_t::pSwCompsData

Pointer to start of sw components data.

CCswCodeEncType_t CCSbCertParserSwCompsInfo_t::swCodeEncType

SW image code encryption type.

CCswCryptoType_t CCSbCertParserSwCompsInfo_t::swCryptoType

SW image crypto type.

CCswLoadVerifyScheme_t CCSbCertParserSwCompsInfo_t::swLoadVerifyScheme

SW image load & verify scheme.

The documentation for this struct was generated from the following file:

cc_crypto_boot_defs.h

2.5.3 CCSbSwVersion_t struct reference

```
#include <secureboot_defs.h>
```

Data fields

- *CCSbPubKeyIndexType_t* *keyIndex*
- uint32_t *swVersion*

Detailed description

SW version

Field documentation

CCSbPubKeyIndexType_t CCSbSwVersion_t::keyIndex

Enumeration defining the key hash to retrieve: 128-bit HBK0, 128-bit HBK1, or 256-bit HBK.

uint32_t CCSbSwVersion_t::swVersion

SW version.

The documentation for this struct was generated from the following file:

- *secureboot_defs.h*

2.6 Boot services file documentation

2.6.1 bootimagesverifier_api.h file reference

`#include "secureboot_defs.h"`

Functions

- *CCError_t CC_SbCertChainVerificationInit* (*CCSbCertInfo_t* *certPkgInfo)
- *CCError_t CC_SbCertVerifySingle* (*CCSbFlashReadFunc* flashReadFunc, void *userContext, unsigned long hwBaseAddress, *CCAddr_t* certStoreAddress, *CCSbCertInfo_t* *certPkgInfo, uint32_t *pHeader, uint32_t headerSize, uint32_t *pWorkspace, uint32_t workspaceSize)

Detailed description

This file contains the set of Secure Boot APIs that are supported by the SBROM library.

Function documentation

CCError_t CC_SbCertChainVerificationInit (CCSbCertInfo_t * `certPkgInfo`)

initializes the Secure Boot certificate chain processing, and must be the first API called when processing Secure Boot certificate chain. It initializes the internal data fields of the certificate package.

Returns:

- CC_OK On success.
- A non-zero value from bsv_error.h on failure.

Parameters:

I/O	Parameter	Description
in,out	`certPkgInfo`	Pointer to the information about the certificate package

CCError_t CC_SbCertVerifySingle (CCSbFlashReadFunc `flashReadFunc`, void * `userContext`, unsigned long `hwBaseAddress`, CCAddr_t `certStoreAddress`, CCSbCertInfo_t * `certPkgInfo`, uint32_t * `pHeader`, uint32_t `headerSize`, uint32_t * `pWorkspace`, uint32_t `workspaceSize`)

This function verifies a single certificate package (containing either a key or content certificate). It verifies the following:

- The public key (as saved in the certificate) against its Hash that is either found in the OTP memory (HBK) or in certPkgInfo.
- The certificate's RSA signature.
- The SW version in the certificate must be higher than or equal to the minimum SW version, as recorded on the device and passed in certPkgInfo.
- Each SW module against its Hash in the certificate (for content certificates).

Returns:

- CC_OK On success.
- A non-zero value from bsv_error.h on failure.

Parameters:

I/O	Parameter	Description
in	`flashReadFunc`	Pointer to the flash read function.

I/O	Parameter	Description
in	`userContext`	An additional pointer for flashRead usage. May be NULL.
in	`hwBaseAddress`	HW registers base address.
in	`certStoreAddress`	Flash address where the certificate is located. This address is provided to flashReadFunc.
in,out	`certPkgInfo`	Pointer to the information about the certificate package.
in,out	`pHeader`	Pointer for extracting the X509 TBS Headers. Should be NULL for proprietary certificates.
in	`headerSize`	The size of the pHeader in bytes. Should be 0 for proprietary certificates.
in	`pWorkspace`	Buffer for the function's internal use.
in	`workspaceSize`	The size of the workspace in bytes. Must be at least CC_SB_MIN_WORKSPACE_SIZE_IN_BYTES.

2.6.2 bootimagesverifier_def.h file reference

`#include "cc_pal_types.h"`

Macros

- #define *CC_SB_MAX_NUM_OF_IMAGES* 16
- #define **CC_SB_MAX_CERT_SIZE_IN_BYTES** (0x700)
- #define **CC_SB_MIN_DBG_WORKSPACE_SIZE_IN_BYTES** (0x350)
- #define *CC_SB_MIN_WORKSPACE_SIZE_IN_BYTES* (CC_DOUBLE_BUFFER_MAX_SIZE_IN_BYTES + CC_SB_MAX_CERT_SIZE_IN_BYTES)

Detailed description

This file contains all definitions that are used for the SBROM service APIs.

Macro definition documentation

#define CC_SB_MAX_NUM_OF_IMAGES 16

Maximum images per content certificate.

#define CC_SB_MIN_WORKSPACE_SIZE_IN_BYTES (CC_DOUBLE_BUFFER_MAX_SIZE_IN_BYTES + CC_SB_MAX_CERT_SIZE_IN_BYTES)

Defines workspace minimum size. The Secure Boot APIs use a temporary workspace for processing the data that is read from the flash, prior to loading the SW modules to their designated memory addresses. This workspace must be large enough to accommodate the size of the certificates, and twice the size of the data that is read from flash in each processing round (for performance reasons, two buffers are processed in parallel). It is assumed that the optimal size of the data to read in each processing round is 4KB, as is the standard page flash memory page size. Therefore, the size of the double buffer (CC_CONFIG_SB_DOUBLE_BUFFER_MAX_SIZE_IN_BYTES) is defined by default as 8KB in the project configuration file. This can be changed to accommodate the optimal value in different environments. The definition of CC_SB_MIN_WORKSPACE_SIZE_IN_BYTES is comprised of CC_DOUBLE_BUFFER_MAX_SIZE_IN_BYTES (defined by the SBROM Makefile as equal to CC_CONFIG_SB_DOUBLE_BUFFER_MAX_SIZE_IN_BYTES) and space for the certificate itself, which resides in the workspace at the same time the SW modules data is processed.

Note

When writing code that uses the Secure Boot APIs and includes the *bootimagesverifier_def.h* file, the value of CC_DOUBLE_BUFFER_MAX_SIZE_IN_BYTES must be defined by the partner's Makefile to be exactly the same value as was used when compiling the SBROM library. CC_SB_X509_CERT_SUPPORTED must also be defined in the Makefile (according to CC_CONFIG_SB_X509_CERT_SUPPORTED definition).

Note

CC_DOUBLE_BUFFER_MAX_SIZE_IN_BYTES size must be a multiple of the hash SHA256 block size (64 bytes).

2.6.3 bootimagesverifier_error.h file reference

```
#include "secureboot_error.h"
```

Macros

- #define *CC_BOOT_IMG_VERIFIER_INV_INPUT_PARAM* (*CC_BOOT_IMG_VERIFIER_BASE_ERROR* + 0x00000001)
- #define *CC_BOOT_IMG_VERIFIER_OTP_VERSION_FAILURE* (*CC_BOOT_IMG_VERIFIER_BASE_ERROR* + 0x00000002)
- #define *CC_BOOT_IMG_VERIFIER_CERT_MAGIC_NUM_INCORRECT* (*CC_BOOT_IMG_VERIFIER_BASE_ERROR* + 0x00000003)
- #define *CC_BOOT_IMG_VERIFIER_CERT_VERSION_NUM_INCORRECT* (*CC_BOOT_IMG_VERIFIER_BASE_ERROR* + 0x00000004)
- #define *CC_BOOT_IMG_VERIFIER_SW_VER_SMALLER_THAN_MIN_VER* (*CC_BOOT_IMG_VERIFIER_BASE_ERROR* + 0x00000005)
- #define *CC_BOOT_IMG_VERIFIER_PUB_KEY_HASH_VALIDATION_FAILURE* (*CC_BOOT_IMG_VERIFIER_BASE_ERROR* + 0x00000006)

- #define CC_BOOT_IMG_VERIFIER_RSA_SIG_VERIFICATION_FAILED (CC_BOOT_IMG_VERIFIER_BASE_ERROR + 0x00000007)
- #define CC_BOOT_IMG_VERIFIER_WORKSPACE_SIZE_TOO_SMALL (CC_BOOT_IMG_VERIFIER_BASE_ERROR + 0x00000008)
- #define CC_BOOT_IMG_VERIFIER_SW_COMP_FAILED_VERIFICATION (CC_BOOT_IMG_VERIFIER_BASE_ERROR + 0x00000009)
- #define CC_BOOT_IMG_VERIFIER_UNSUPPORTED_HASH_ALGORITHM (CC_BOOT_IMG_VERIFIER_BASE_ERROR + 0x0000000B)
- #define CC_BOOT_IMG_VERIFIER_UNSUPPORTED_RSA_ALGORITHM (CC_BOOT_IMG_VERIFIER_BASE_ERROR + 0x0000000C)
- #define CC_BOOT_IMG_VERIFIER_CERT_SW_VER_ILLEGAL (CC_BOOT_IMG_VERIFIER_BASE_ERROR + 0x0000000D)
- #define CC_BOOT_IMG_VERIFIER_SW_COMP_SIZE_IS_NULL (CC_BOOT_IMG_VERIFIER_BASE_ERROR + 0x00000011)
- #define CC_BOOT_IMG_VERIFIER_PUBLIC_KEY_HASH_EMPTY (CC_BOOT_IMG_VERIFIER_BASE_ERROR + 0x00000014)
- #define CC_BOOT_IMG_VERIFIER_ILLEGAL_LCS_FOR_OPERATION_ERR (CC_BOOT_IMG_VERIFIER_BASE_ERROR + 0x00000015)
- #define CC_BOOT_IMG_VERIFIER_PUB_KEY_ALREADY_PROGRAMMED_ERR (CC_BOOT_IMG_VERIFIER_BASE_ERROR + 0x00000016)
- #define CC_BOOT_IMG_VERIFIER_OTP_WRITE_FAIL_ERR (CC_BOOT_IMG_VERIFIER_BASE_ERROR + 0x00000017)
- #define CC_BOOT_IMG_VERIFIER_INCORRECT_CERT_TYPE (CC_BOOT_IMG_VERIFIER_BASE_ERROR + 0x00000018)
- #define CC_BOOT_IMG_VERIFIER_ILLEGAL_HBK_IDX (CC_BOOT_IMG_VERIFIER_BASE_ERROR + 0x00000019)
- #define CC_BOOT_IMG_VERIFIER_PUB_KEY1_NOT_PROGRAMMED_ERR (CC_BOOT_IMG_VERIFIER_BASE_ERROR + 0x0000001A)
- #define CC_BOOT_IMG_VERIFIER_CERT_VER_SIZE_ILLEGAL (CC_BOOT_IMG_VERIFIER_BASE_ERROR + 0x0000001B)
- #define CC_BOOT_IMG_VERIFIER_CERT_VER_VAL_ILLEGAL (CC_BOOT_IMG_VERIFIER_BASE_ERROR + 0x0000001C)
- #define CC_BOOT_IMG_VERIFIER_CERT_DECODING_ILLEGAL (CC_BOOT_IMG_VERIFIER_BASE_ERROR + 0x0000001D)

- #define CC_BOOT_IMG_VERIFIER_ILLEGAL_KCE_IN_RMA_STATE (CC_BOOT_IMG_VERIFIER_BASE_ERROR + 0x0000001E)
- #define CC_BOOT_IMG_VERIFIER_ILLEGAL_SOC_ID_VALUE (CC_BOOT_IMG_VERIFIER_BASE_ERROR + 0x0000001F)
- #define CC_BOOT_IMG_VERIFIER_ILLEGAL_NUM_OF_IMAGES (CC_BOOT_IMG_VERIFIER_BASE_ERROR + 0x00000020)
- #define CC_BOOT_IMG_VERIFIER_SKIP_PUBLIC_KEY_VERIFY (CC_BOOT_IMG_VERIFIER_BASE_ERROR + 0x00000014)

Detailed description

This file contains error code definitions that are used for the SBROM service APIs.

Macro definition documentation

#define CC_BOOT_IMG_VERIFIER_CERT_DECODING_ILLEGAL (CC_BOOT_IMG_VERIFIER_BASE_ERROR + 0x0000001D)

Illegal certificate decoding value.

#define CC_BOOT_IMG_VERIFIER_CERT_MAGIC_NUM_INCORRECT (CC_BOOT_IMG_VERIFIER_BASE_ERROR + 0x00000003)

Illegal certificate's magic number.

#define CC_BOOT_IMG_VERIFIER_CERT_SW_VER_ILLEGAL (CC_BOOT_IMG_VERIFIER_BASE_ERROR + 0x0000000D)

Illegal SW version or ID of SW version.

#define CC_BOOT_IMG_VERIFIER_CERT_VER_SIZE_ILLEGAL (CC_BOOT_IMG_VERIFIER_BASE_ERROR + 0x0000001B)

Illegal certificate version size.

#define CC_BOOT_IMG_VERIFIER_CERT_VER_VAL_ILLEGAL (CC_BOOT_IMG_VERIFIER_BASE_ERROR + 0x0000001C)

Illegal certificate version value.

#define CC_BOOT_IMG_VERIFIER_CERT_VERSION_NUM_INCORRECT (CC_BOOT_IMG_VERIFIER_BASE_ERROR + 0x00000004)

Illegal certificate version.

#define CC_BOOT_IMG_VERIFIER_ILLEGAL_HBK_IDX (CC_BOOT_IMG_VERIFIER_BASE_ERROR + 0x00000019)

Illegal Hash boot key index.

#define CC_BOOT_IMG_VERIFIER_ILLEGAL_KCE_IN_RMA_STATE (*CC_BOOT_IMG_VERIFIER_BASE_ERROR* + 0x0000001E)

Illegal KCE in RMA state.

#define CC_BOOT_IMG_VERIFIER_ILLEGAL_LCS_FOR_OPERATION_ERR (*CC_BOOT_IMG_VERIFIER_BASE_ERROR* + 0x00000015)

Illegal LCS for operation.

#define CC_BOOT_IMG_VERIFIER_ILLEGAL_NUM_OF_IMAGES (*CC_BOOT_IMG_VERIFIER_BASE_ERROR* + 0x00000020)

Illegal number of images per content certificate.

#define CC_BOOT_IMG_VERIFIER_ILLEGAL_SOC_ID_VALUE (*CC_BOOT_IMG_VERIFIER_BASE_ERROR* + 0x0000001F)

Illegal SOC Id value.

#define CC_BOOT_IMG_VERIFIER_INCORRECT_CERT_TYPE (*CC_BOOT_IMG_VERIFIER_BASE_ERROR* + 0x00000018)

Incorrect certificate type.

#define CC_BOOT_IMG_VERIFIER_INV_INPUT_PARAM (*CC_BOOT_IMG_VERIFIER_BASE_ERROR* + 0x00000001)

Invalid input parameters.

#define CC_BOOT_IMG_VERIFIER_OTP_VERSION_FAILURE (*CC_BOOT_IMG_VERIFIER_BASE_ERROR* + 0x00000002)

Invalid OTP version.

#define CC_BOOT_IMG_VERIFIER_OTP_WRITE_FAIL_ERR (*CC_BOOT_IMG_VERIFIER_BASE_ERROR* + 0x00000017)

Write to OTP failed.

#define CC_BOOT_IMG_VERIFIER_PUB_KEY1_NOT_PROGRAMMED_ERR (*CC_BOOT_IMG_VERIFIER_BASE_ERROR* + 0x0000001A)

Hash boot key of ICV is not programmed.

#define CC_BOOT_IMG_VERIFIER_PUB_KEY_ALREADY_PROGRAMMED_ERR (*CC_BOOT_IMG_VERIFIER_BASE_ERROR* + 0x00000016)

Hash of public key is already programmed.

#define CC_BOOT_IMG_VERIFIER_PUB_KEY_HASH_VALIDATION_FAILURE (*CC_BOOT_IMG_VERIFIER_BASE_ERROR* + 0x00000006)

Public key verification versus the OTP failed.

#define CC_BOOT_IMG_VERIFIER_PUBLIC_KEY_HASH_EMPTY (*CC_BOOT_IMG_VERIFIER_BASE_ERROR* + 0x00000014)

Hash of public key is not burned yet.

#define CC_BOOT_IMG_VERIFIER_RSA_SIG_VERIFICATION_FAILED (*CC_BOOT_IMG_VERIFIER_BASE_ERROR* + 0x00000007)

Certificate's RSA signature verification failed.

#define CC_BOOT_IMG_VERIFIER_SKIP_PUBLIC_KEY_VERIFY (*CC_BOOT_IMG_VERIFIER_BASE_ERROR* + 0x00000014)

No need to verify hashed public key.

#define CC_BOOT_IMG_VERIFIER_SW_COMP_FAILED_VERIFICATION (*CC_BOOT_IMG_VERIFIER_BASE_ERROR* + 0x00000009)

SW image hash verification failed.

#define CC_BOOT_IMG_VERIFIER_SW_COMP_SIZE_IS_NULL (*CC_BOOT_IMG_VERIFIER_BASE_ERROR* + 0x00000011)

Illegal number of SW components (zero).

#define CC_BOOT_IMG_VERIFIER_SW_VER_SMALLER_THAN_MIN_VER (*CC_BOOT_IMG_VERIFIER_BASE_ERROR* + 0x00000005)

Illegal certificate's sw version (smaller than version in the otp).

#define CC_BOOT_IMG_VERIFIER_UNSUPPORTED_HASH_ALGORITHM (*CC_BOOT_IMG_VERIFIER_BASE_ERROR* + 0x0000000B)

Unsupported HASH algorithm.

#define CC_BOOT_IMG_VERIFIER_UNSUPPORTED_RSA_ALGORITHM (*CC_BOOT_IMG_VERIFIER_BASE_ERROR* + 0x0000000C)

Unsupported RSA algorithm.

#define CC_BOOT_IMG_VERIFIER_WORKSPACE_SIZE_TOO_SMALL (*CC_BOOT_IMG_VERIFIER_BASE_ERROR* + 0x00000008)

Workspace buffer given to the API is too small.

2.6.4 bsv_api.h file reference

```
#include "cc_pal_types.h"

#include "bsv_defs.h"

#include "cc_sec_defs.h"
```

Macros

- #define *CC_BSV_CHIP_MANUFACTURE_LCS* 0x0
- #define *CC_BSV_DEVICE_MANUFACTURE_LCS* 0x1

- #define *CC_BSV_SECURE_LCS* 0x5
- #define *CC_BSV_RMA_LCS* 0x7

Functions

- *CCError_t CC_BsvInit* (unsigned long hwBaseAddress)
- *CCError_t CC_BsvLcsGet* (unsigned long hwBaseAddress, uint32_t *pLcs)
- *CCError_t CC_BsvLcsGetAndInit* (unsigned long hwBaseAddress, uint32_t *pLcs)
- *CCError_t CC_BsvOTPPrivateKeysErase* (unsigned long hwBaseAddress, *CCBool_t* isHukErase, *CCBool_t* isKpicvErase, *CCBool_t* isKceicvErase, *CCBool_t* isKcpErase, *CCBool_t* isKceErase, uint32_t *pStatus)
- *CCError_t CC_BsvFatalErrorSet* (unsigned long hwBaseAddress)
- *CCError_t CC_BsvRMAModeEnable* (unsigned long hwBaseAddress)
- *CCError_t CC_BsvSocIDCompute* (unsigned long hwBaseAddress, *CCHashResult_t* hashResult)
- *CCError_t CC_BsvICVKeyLock* (unsigned long hwBaseAddress, *CCBool_t* isICVProvisioningKeyLock, *CCBool_t* isICVCodeEncKeyLock)
- *CCError_t CC_BsvICVRMAFlagBitLock* (unsigned long hwBaseAddress)
- *CCError_t CC_BsvCoreClkGatingEnable* (unsigned long hwBaseAddress)
- *CCError_t CC_BsvSecModeSet* (unsigned long hwBaseAddress, *CCBool_t* isSecAccessMode, *CCBool_t* isSecModeLock)
- *CCError_t CC_BsvPrivModeSet* (unsigned long hwBaseAddress, *CCBool_t* isPrivAccessMode, *CCBool_t* isPrivModeLock)
- *CCError_t CC_BsvIcvAssetProvisioningOpen* (unsigned long hwBaseAddress, uint32_t assetId, uint32_t *pAssetPkgBuff, size_t assetPackageLen, uint8_t *pOutAssetData, size_t *pAssetDataLen)

Detailed description

This file contains all the SBROM library APIs and definitions.

Macro definition documentation

#define CC_BSV_CHIP_MANUFACTURE_LCS 0x0

CM lifecycle value.

#define CC_BSV_DEVICE_MANUFACTURE_LCS 0x1

DM lifecycle value.

#define CC_BSV_RMA_LCS 0x7

RMA lifecycle value.

#define CC_BSV_SECURE_LCS 0x5

Secure lifecycle value.

Function documentation

CCError_t CC_BsvCoreClkGatingEnable (unsigned long `hwBaseAddress`)

This API shall enable the core_clk gating mechanism as this feature is disabled during power-up. This API shall be called during the first boot HW initialization so it can save power (due to the gating) from secure debug and Secure Boot.

Returns:

- CC_OK On success.
- A non-zero value from *bsv_error.h* on failure.

Parameters:

I/O	Parameter	Description
In	`hwBaseAddress`	HW registers base address.

CCError_t CC_BsvFatalErrorSet (unsigned long `hwBaseAddress`)

This function sets the "fatal error" flag in the NVM manager to disable any HW Keys usage and security services.

Returns:

- CC_OK On success.
- A non-zero value from *bsv_error.h* on failure.

Parameters:

I/O	Parameter	Description
in	`hwBaseAddress`	HW registers base address.

CCError_t CC_BsvIcvAssetProvisioningOpen (unsigned long `hwBaseAddress`, uint32_t `assetId`, uint32_t * `pAssetPkgBuff`, size_t `assetPackageLen`, uint8_t * `pOutAssetData`, size_t * `pAssetDataLen`)

The function unpacks the ICV asset packet and return the asset data.

Returns:

- CC_OK On success.
- A non-zero value from *bsv_error.h* on failure.

Parameters:

I/O	Parameter	Description
in	`hwBaseAddress`	HW registers base address.
in	`assetId`	An asset identifier.
in	`pAssetPkgBuff`	An asset package word-array formatted to unpack.
in	`assetPackageLen`	An asset package exact length in bytes, must be multiple of 16 bytes.
out	`pOutAssetData`	The decrypted contents of asset data.
in,out	`pAssetDataLen`	As input: the size of the allocated asset data buffer (max size is 512 bytes). As output: the actual size of the decrypted asset data buffer (max size is 512 bytes).

CCError_t CC_BsvICVKeyLock (unsigned long `hwBaseAddress`, *CCBool_t* `isICVProvisioningKeyLock`, *CCBool_t* `isICVCodeEncKeyLock`)

This function should be called in case the user needs to lock one of the ICV keys from further usage.

Returns:

- CC_OK On success.
- A non-zero value from *bsv_error.h* on failure.

Parameters:

I/O	Parameter	Description
in	`hwBaseAddress`	HW registers base address.
in	`isICVProvisioningKeyLock`	The ICV provisioning key mode: • CC_TRUE: Kpicv will be locked for further usage. • CC_FALSE: Kpicv is not locked.
in	`isICVCodeEncKeyLock`	The ICV code encryption key mode: • CC_TRUE: Kceicv will be locked for further usage. • CC_FALSE: Kceicv is not locked.

CCError_t CC_BsvICVRMAFlagBitLock (unsigned long `hwBaseAddress`)

This function should be called by the ICV code to disable the OEM code to change the ICV RMA bit flag.

Returns:

- CC_OK On success.
- A non-zero value from *bsv_error.h* on failure.

Parameters:

I/O	Parameter	Description
In	`hwBaseAddress`	HW registers base address.

CCError_t CC_BsvInit (unsigned long `hwBaseAddress`)

This function should be the first Arm CryptoCell 3xx SBROM library API called. It verifies the HW product and version numbers.

Returns:

- CC_OK On success.
- A non-zero value from *bsv_error.h* on failure.

Parameters:

I/O	Parameter	Description
in	`hwBaseAddress`	HW registers base address.

CCError_t CC_BsvLcsGet (unsigned long `hwBaseAddress`, uint32_t * `pLcs`)

This function retrieves the security lifecycle from the NVM manager.

Returns:

- CC_OK On success.
- A non-zero value from *bsv_error.h* on failure.

Parameters:

I/O	Parameter	Description
in	`hwBaseAddress`	HW registers base address.
out	`pLcs`	Value of the current LCS.

CCError_t CC_BsvLcsGetAndInit (unsigned long `hwBaseAddress`, uint32_t * `pLcs`)

This function retrieves the HW security lifecycle state, performs validity checks, and additional initializations in case the LCS is RMA (sets the OTP secret keys to fixed value).

Note

Invalid LCS results in an error returned. In this case, the partner's code must completely disable the device.

Returns:

- CC_OK On success.
- A non-zero value from *bsv_error.h* on failure.

Parameters:

I/O	Parameter	Description
in	`hwBaseAddress`	HW registers base address.
out	`pLcs`	Returned lifecycle state.

CCError_t CC_BsvOTPPrivateKeysErase (unsigned long `hwBaseAddress`, *CCBool_t* `isHukErase`, *CCBool_t* `isKpicvErase`, *CCBool_t* `isKceicvErase`, *CCBool_t* `isKcpErase`, *CCBool_t* `isKceErase`, uint32_t * `pStatus`)

This function should be called in RMA mode to erase one or more of the private keys.

Returns:

- CC_OK On success.
- A non-zero value from *bsv_error.h* on failure.

Parameters:

I/O	Parameter	Description
in	`hwBaseAddress`	HW registers base address.
in	`isHukErase`	HUK secret key: • CC_TRUE: HUK secret will be erased. • CC_FALSE: HUK secret remains unchanged.
in	`isKpicvErase`	Kpicv secret key: • CC_TRUE: Kpicv secret will be erased. • CC_FALSE: Kpicv secret remains unchanged.
in	`isKceicvErase`	Kceicv secret key: • CC_TRUE: Kceicv secret will be erased. • CC_FALSE: Kceicv secret remains unchanged.
in	`isKcpErase`	Kcp secret key: • CC_TRUE: Kcp secret will be erased. • CC_FALSE: Kcp secret remains unchanged.
in	`isKceErase`	Kce secret key: • CC_TRUE: Kce secret will be erased. • CC_FALSE: Kce secret remains unchanged.
out	`pStatus`	Returned status word.

CCError_t CC_BsvPrivModeSet (unsigned long `hwBaseAddress`, *CCBool_t* `isPrivAccessMode`, *CCBool_t* `isPrivModeLock`)

This function should activate the APB privilege filter. (Allowing only secure transactions to access Arm CryptoCell-312 registers).

Returns:

- CC_OK On success.
- A non-zero value from *bsv_error.h* on failure.

Parameters:

I/O	Parameter	Description
in	`hwBaseAddress`	HW registers base address.
in	`isPrivAccessMode`	The APB privileged mode: • CC_TRUE: only privileged accesses are served. • CC_FALSE: both privileged and non-privileged accesses are served.
in	`isPrivModeLock`	The privileged lock mode: • CC_TRUE: privileged mode is locked for further changes. • CC_FALSE: privileged mode is not locked.

CCError_t CC_BsvRMAModeEnable (unsigned long `hwBaseAddress`)

This function sets the "RMA mode" lifecycle state per OEM/ICV permanently.

Returns:

- CC_OK On success.
- A non-zero value from *bsv_error.h* on failure.

Parameters:

I/O	Parameter	Description
In	`hwBaseAddress`	HW registers base address.

CCError_t CC_BsvSecModeSet (unsigned long `hwBaseAddress`, *CCBool_t* `isSecAccessMode`, *CCBool_t* `isSecModeLock`)

This function control the APB secure filter. (Allowing only secure transactions to access Arm CryptoCell-312 registers).

Returns:

- CC_OK On success.
- A non-zero value from *bsv_error.h* on failure.

Parameters:

I/O	Parameter	Description
in	`hwBaseAddress`	HW registers base address.
in	`isSecAccessMode`	The APB secure filter mode: • CC_TRUE: only secure accesses are served. • CC_FALSE: both secure and non-secure accesses are served.
in	`isSecModeLock`	The APB security lock mode: • CC_TRUE: secure filter mode is locked for further changes. • CC_FALSE: secure filter mode is not locked.

CCError_t CC_BsvSocIDCompute (unsigned long `hwBaseAddress`, CCHashResult_t `hashResult`)

This function derives the device's unique SOC_ID as hashed (HBK || AES_CMAC (HUK)).

Note

SOC_ID is required for the creation of debug certificates. Therefore, the OEM (or CM) should provide a method for a developer to discover the SOC_ID of a target device without having to first enable debugging. One suggested implementation is to have the device ROM code compute the SOC_ID and place it in a specific location in the flash memory, where it can be accessed by the developer.

Returns:

- CC_OK On success.
- A non-zero value from *bsv_error.h* on failure.

Parameters:

I/O	Parameter	Description
in	`hwBaseAddress`	HW registers base address.
out	`hashResult`	The derived SOC ID.

2.6.5 bsv_crypto_api.h file reference

```
#include "cc_pal_types.h"

#include "bsv_crypto_defs.h"

#include "cc_sec_defs.h"
```

Functions

- *CCError_t CC_BsvSHA256* (unsigned long hwBaseAddress, uint8_t *pDataIn, size_t dataSize, *CCHashResult_t* hashBuff)
- *CCError_t CC_BsvKeyDerivation* (unsigned long hwBaseAddress, *CCBsvKeyType_t* keyType, uint32_t *pUserKey, size_t userKeySize, const uint8_t *pLabel, size_t labelSize, const uint8_t *pContextData, size_t contextSize, uint8_t *pDerivedKey, size_t derivedKeySize)
- *CCError_t CC_BsvAesCcm* (unsigned long hwBaseAddress, *CCBsvCcmKey_t* keyBuf, *CCBsvCcmNonce_t* nonceBuf, uint8_t *pAssocData, size_t assocDataSize, uint8_t *pTextDataIn, size_t textDataSize, uint8_t *pTextDataOut, *CCBsvCcmMacRes_t* macBuf)

Detailed description

This file defines crypto ROM APIs : SH256, CMAC KDF, and CCM.

Function documentation

CCError_t CC_BsvAesCcm (unsigned long `hwBaseAddress`, CCBsvCcmKey_t `keyBuf`, CCBsvCcmNonce_t `nonceBuf`, uint8_t * `pAssocData`, size_t `assocDataSize`, uint8_t * `pTextDataIn`, size_t `textDataSize`, uint8_t * `pTextDataOut`, CCBsvCcmMacRes_t `macBuf`)

This API shall allow a limited AESCCM operation (only decrypt and verify) needed for AESCCM verification during boot. AES CCM combines Counter mode encryption with CBC-MAC authentication. Input to CCM includes the following elements:

- Payload - text data that is both decrypted and verified.
- Associated data (Adata) - data that is authenticated but not encrypted, e.g., a header.
- Nonce - A unique value that is assigned to the payload and the associated data.

Returns:

- CC_OK on success.
- A non-zero value on failure as defined *bsv_error.h*.

Parameters:

I/O	Parameter	Description
in	`hwBaseAddress`	HW registers base address.
in	`keyBuf`	Pointer to 128bit AES-CCM key.
in	`nonceBuf`	Pointer to the 12 bytes Nonce.
in	`pAssocData`	Pointer to the associated data. The buffer must be contiguous.
in	`assocDataSize`	Byte size of the associated data limited to (2^16-2^8) bytes.
in	`pTextDataIn`	Pointer to the cipher-text data for decryption. The buffer must be contiguous.
in	`textDataSize`	Byte size of the full text data limited to 64K Bytes.
out	`pTextDataOut`	Pointer to the output (plain text data). The buffer must be contiguous.
in	`macBuf`	Pointer to the MAC result buffer.

CCError_t CC_BsvKeyDerivation (unsigned long `hwBaseAddress`, CCBsvKeyType_t `keyType`, uint32_t * `pUserKey`, size_t `userKeySize`, const uint8_t * `pLabel`, size_t `labelSize`, const uint8_t * `pContextData`, size_t `contextSize`, uint8_t * `pDerivedKey`, size_t `derivedKeySize`)

The key derivation function is as specified in the "KDF in Counter Mode" section of NIST Special Publication 800-108: Recommendation for Key Derivation Using Pseudorandom Functions. The derivation is based on length l, label L, context C and derivation key Ki. AES-CMAC is used as the pseudorandom function (PRF).

Note

The user must well define the label and context for each use-case, when using this API.

User is recommended to derive 256 bit keys only from HUK or 256 bit user keys.

Returns:

- CC_OK on success.
- A non-zero value from *bsv_error.h* on failure.

Parameters:

I/O	Parameter	Description
in	`hwBaseAddress`	HW registers base address.
in	`keyType`	The key type that is used as an input to a key derivation function. Can be one of: HUK / Krtl / KCP / KPICV / 128b User key / 256b User Key.
in	`pUserKey`	A pointer to the user's key buffer.
in	`userKeySize`	The user key size in bytes (limited to 128bits or 256bits).
in	`pLabel`	A string that identifies the purpose for the derived keying material.
in	`labelSize`	The label size should be in range of 1 to 8 bytes length.
in	`pContextData`	A binary string containing the information related to the derived keying material.
in	`contextSize`	The context size should be in range of 1 to 32 bytes length.
out	`pDerivedKey`	Keying material output (MUST be at least the size of derivedKeySize).
in	`derivedKeySize`	Size of the derived keying material in bytes (limited to 128bits or 256bits).

CCError_t CC_BsvSHA256 (unsigned long `hwBaseAddress`, uint8_t * `pDataIn`, size_t `dataSize`, *CCHashResult_t* `hashBuff`)

This function calculates SHA256 digest over contiguous memory in integrated operation.

Returns:

- CC_OK On success.
- A non-zero value from *bsv_error.h* on failure.

Parameters:

I/O	Parameter	Description
in	`hwBaseAddress`	HW registers base address.
in	`pDataIn`	Pointer to the input data to be HASHed. Buffer must be contiguous.
in	`dataSize`	The size of the data to be hashed in bytes. Limited to 64KB.
out	`hashBuff`	Pointer to a word-aligned 32 byte buffer.

2.6.6 bsv_crypto_defs.h file reference

Macros

- #define *CC_BSV_CMAC_RESULT_SIZE_IN_WORDS* 4 /* 128b */
- #define *CC_BSV_CMAC_RESULT_SIZE_IN_BYTES* 16 /* 128b */
- #define *CC_BSV_CCM_KEY_SIZE_BYTES* 16
- #define *CC_BSV_CCM_KEY_SIZE_WORDS* 4
- #define *CC_BSV_CCM_NONCE_SIZE_BYTES* 12

Typedefs

- typedef uint32_t *CCBsvCmacResult_t*[*CC_BSV_CMAC_RESULT_SIZE_IN_WORDS*]
- typedef uint32_t *CCBsvCcmKey_t*[*CC_BSV_CCM_KEY_SIZE_WORDS*]
- typedef uint8_t *CCBsvCcmNonce_t*[*CC_BSV_CCM_NONCE_SIZE_BYTES*]
- typedef uint8_t *CCBsvCcmMacRes_t*[*CC_BSV_CMAC_RESULT_SIZE_IN_BYTES*]

Enumerations

- enum *CCBsvKeyType_t* { *CC_BSV_HUK_KEY* = 0, *CC_BSV_RTL_KEY* = 1, *CC_BSV_PROV_KEY* = 2, *CC_BSV_CE_KEY* = 3, *CC_BSV_ICV_PROV_KEY* = 4, *CC_BSV_ICV_CE_KEY* = 5, *CC_BSV_USER_KEY* = 6, *CC_BSV_END_OF_KEY_TYPE* = 0x7FFFFFFF }

Detailed description

This file contains crypto ROM API definitions.

Macro definition documentation

#define CC_BSV_CCM_KEY_SIZE_BYTES 16

AES CCM 128bit key size in bytes.

#define CC_BSV_CCM_KEY_SIZE_WORDS 4

AES CCM 128bit key size in words.

#define CC_BSV_CCM_NONCE_SIZE_BYTES 12

AES CCM NONCE size in bytes.

#define CC_BSV_CMAC_RESULT_SIZE_IN_BYTES 16 /* 128b */

AES CMAC result size in bytes.

#define CC_BSV_CMAC_RESULT_SIZE_IN_WORDS 4 /* 128b */

AES CMAC result size in words.

Typedef documentation

typedef uint32_t CCBsvCcmKey_t[*CC_BSV_CCM_KEY_SIZE_WORDS*]

AES_CCM key buffer definition.

typedef uint8_t CCBsvCcmMacRes_t[*CC_BSV_CMAC_RESULT_SIZE_IN_BYTES*]

AES_CCM MAC buffer definition.

typedef uint8_t CCBsvCcmNonce_t[*CC_BSV_CCM_NONCE_SIZE_BYTES*]

AES_CCM nonce buffer definition.

typedef uint32_t CCBsvCmacResult_t[CC_BSV_CMAC_RESULT_SIZE_IN_WORDS]

CMAC result buffer.

Enumeration type documentation

enum *CCBsvKeyType_t*

Definitions for AES key types.

Enumerator:

Enum	Description
CC_BSV_HUK_KEY	Root key (Huk).
CC_BSV_RTL_KEY	RTL key (Krtl).
CC_BSV_PROV_KEY	Provision key (Kcp).
CC_BSV_CE_KEY	Code encryption key (Kce).
CC_BSV_ICV_PROV_KEY	Provision key (Kpicv).
CC_BSV_ICV_CE_KEY	Code encryption key (Kceicv).
CC_BSV_USER_KEY	User's key.
CC_BSV_END_OF_KEY_TYPE	Reserved.

2.6.7 bsv_defs.h file reference

Macros

- #define *CC_BSV_MAX_HASH_SIZE_IN_WORDS* 8
- #define *CC_BSV_MAX_HASH_SIZE_IN_BYTES* (*CC_BSV_MAX_HASH_SIZE_IN_WORDS**sizeof(uint32_t))
- #define *CC_BSV_256B_HASH_SIZE_IN_WORDS* *CC_BSV_MAX_HASH_SIZE_IN_WORDS*
- #define *CC_BSV_128B_HASH_SIZE_IN_WORDS* *CC_BSV_MAX_HASH_SIZE_IN_WORDS*/2
- #define *CC_BSV_MAX_HBK0_VERSION_COUNTER* 63
- #define *CC_BSV_MAX_HBK1_VERSION_COUNTER* 95
- #define *CC_BSV_MAX_HBK_VERSION_COUNTER* 159
- #define *DX_BSV_STAUS_HUK_ERR_BIT_SHIFT* 0x0UL
- #define *DX_BSV_STAUS_HUK_ERR_BIT_SIZE* 0x1UL
- #define *DX_BSV_STAUS_KPICV_ERR_BIT_SHIFT* 0x1UL
- #define *DX_BSV_STAUS_KPICV_ERR_BIT_SIZE* 0x1UL
- #define *DX_BSV_STAUS_KCEICV_ERR_BIT_SHIFT* 0x2UL
- #define *DX_BSV_STAUS_KCEICV_ERR_BIT_SIZE* 0x1UL
- #define *DX_BSV_STAUS_KCP_ERR_BIT_SHIFT* 0x3UL
- #define *DX_BSV_STAUS_KCP_ERR_BIT_SIZE* 0x1UL
- #define *DX_BSV_STAUS_KCE_ERR_BIT_SHIFT* 0x4UL
- #define *DX_BSV_STAUS_KCE_ERR_BIT_SIZE* 0x1UL
- #define *CC_BSV_ALL_ONES_VALUE* 0xffffffffUL
- #define *CC_BSV_ALL_ONES_NUM_BITS* 32
- #define *CC_BSV_COUNT_ZEROES*(regVal, regZero)

- #define CONVERT_TO_ADDR(ptr)   (unsigned long)ptr   \

Detailed description

This file contains definitions that are used for the Secure Boot ROM service APIs.

Macro definition documentation

#define CC_BSV_128B_HASH_SIZE_IN_WORDS   CC_BSV_MAX_HASH_SIZE_IN_WORDS/2

Maximal dual hash boot key size in words.

#define CC_BSV_256B_HASH_SIZE_IN_WORDS   CC_BSV_MAX_HASH_SIZE_IN_WORDS

Maximal full hash boot key size in words.

#define CC_BSV_ALL_ONES_NUM_BITS   32

Definition for number of bits in a 32bit word.

#define CC_BSV_ALL_ONES_VALUE   0xffffffffUL

Definition for all ones word.

#define CC_BSV_COUNT_ZEROES(regVal,   regZero)

```
do {                                    \
        uint32_t val = regVal;                                      \
        val = val - ((val >> 1) & 0x55555555);                      \
        val = (val & 0x33333333) + ((val >> 2) & 0x33333333);       \
        val = ((((val + (val >> 4)) & 0xF0F0F0F) * 0x1010101) >> 24);   \
        regZero += (32 - val);                      \
    }while(0)
```

This macro counts the number of zeroes in a 32bits word.

#define CC_BSV_MAX_HASH_SIZE_IN_BYTES   (CC_BSV_MAX_HASH_SIZE_IN_WORDS*sizeof(uint32_t))

Maximal hash boot key size in bytes.

#define CC_BSV_MAX_HASH_SIZE_IN_WORDS   8

Maximal hash boot key size in words.

#define CC_BSV_MAX_HBK0_VERSION_COUNTER   63

ICV Firmware minimal version maximal size.

#define CC_BSV_MAX_HBK1_VERSION_COUNTER   95

OEM Firmware minimal version maximal size.

#define CC_BSV_MAX_HBK_VERSION_COUNTER   159

OEM Firmware minimal version maximal size (no ICV).

#define CONVERT_TO_ADDR(ptr)   (unsigned long)ptr   \

Definition for converting pointer to address.

#define DX_BSV_STAUS_HUK_ERR_BIT_SHIFT   0x0UL

HUK status bit definition.

#define DX_BSV_STAUS_HUK_ERR_BIT_SIZE   0x1UL

HUK status size bit definition.

#define DX_BSV_STAUS_KCE_ERR_BIT_SHIFT   0x4UL

Kce status bit definition.

#define DX_BSV_STAUS_KCE_ERR_BIT_SIZE   0x1UL

Kce status size bit definition.

#define DX_BSV_STAUS_KCEICV_ERR_BIT_SHIFT   0x2UL

Kceicv status bit definition.

#define DX_BSV_STAUS_KCEICV_ERR_BIT_SIZE   0x1UL

Kceicv status size bit definition.

#define DX_BSV_STAUS_KCP_ERR_BIT_SHIFT   0x3UL

Kcp status bit definition.

#define DX_BSV_STAUS_KCP_ERR_BIT_SIZE   0x1UL

Kcp status size bit definition.

#define DX_BSV_STAUS_KPICV_ERR_BIT_SHIFT   0x1UL

Kpicv status bit definition.

#define DX_BSV_STAUS_KPICV_ERR_BIT_SIZE   0x1UL

Kpicv status size bit definition.

2.6.8 bsv_error.h file reference

Macros

- #define CC_BSV_BASE_ERROR 0x0B000000
- #define CC_BSV_CRYPTO_ERROR 0x0C000000
- #define CC_BSV_ILLEGAL_INPUT_PARAM_ERR (CC_BSV_BASE_ERROR + 0x00000001)
- #define CC_BSV_ILLEGAL_HUK_VALUE_ERR (CC_BSV_BASE_ERROR + 0x00000002)
- #define CC_BSV_ILLEGAL_KCP_VALUE_ERR (CC_BSV_BASE_ERROR + 0x00000003)
- #define CC_BSV_ILLEGAL_KCE_VALUE_ERR (CC_BSV_BASE_ERROR + 0x00000004)
- #define CC_BSV_ILLEGAL_KPICV_VALUE_ERR (CC_BSV_BASE_ERROR + 0x00000005)
- #define CC_BSV_ILLEGAL_KCEICV_VALUE_ERR (CC_BSV_BASE_ERROR + 0x00000006)
- #define CC_BSV_HASH_NOT_PROGRAMMED_ERR (CC_BSV_BASE_ERROR + 0x00000007)
- #define CC_BSV_HBK_ZERO_COUNT_ERR (CC_BSV_BASE_ERROR + 0x00000008)
- #define CC_BSV_ILLEGAL_LCS_ERR (CC_BSV_BASE_ERROR + 0x00000009)
- #define CC_BSV_OTP_WRITE_CMP_FAIL_ERR (CC_BSV_BASE_ERROR + 0x0000000A)
- #define CC_BSV_ERASE_KEY_FAILED_ERR (CC_BSV_BASE_ERROR + 0x0000000B)
- #define CC_BSV_ILLEGAL_PIDR_ERR (CC_BSV_BASE_ERROR + 0x0000000C)
- #define CC_BSV_ILLEGAL_CIDR_ERR (CC_BSV_BASE_ERROR + 0x0000000D)
- #define CC_BSV_FAILED_TO_SET_FATAL_ERR (CC_BSV_BASE_ERROR + 0x0000000E)
- #define CC_BSV_FAILED_TO_SET_RMA_ERR (CC_BSV_BASE_ERROR + 0x0000000F)
- #define CC_BSV_ILLEGAL_RMA_INDICATION_ERR (CC_BSV_BASE_ERROR + 0x00000010)
- #define CC_BSV_VER_IS_NOT_INITIALIZED_ERR (CC_BSV_BASE_ERROR + 0x00000011)
- #define CC_BSV_APB_SECURE_IS_LOCKED_ERR (CC_BSV_BASE_ERROR + 0x00000012)
- #define CC_BSV_APB_PRIVILEG_IS_LOCKED_ERR (CC_BSV_BASE_ERROR + 0x00000013)
- #define CC_BSV_ILLEGAL_OPERATION_ERR (CC_BSV_BASE_ERROR + 0x00000014)
- #define CC_BSV_ILLEGAL_ASSET_SIZE_ERR (CC_BSV_BASE_ERROR + 0x00000015)
- #define CC_BSV_ILLEGAL_ASSET_VAL_ERR (CC_BSV_BASE_ERROR + 0x00000016)
- #define CC_BSV_KPICV_IS_LOCKED_ERR (CC_BSV_BASE_ERROR + 0x00000017)
- #define CC_BSV_INVALID_DATA_IN_POINTER_ERROR (CC_BSV_CRYPTO_ERROR + 0x00000001)
- #define CC_BSV_INVALID_DATA_OUT_POINTER_ERROR (CC_BSV_CRYPTO_ERROR + 0x00000002)
- #define CC_BSV_INVALID_DATA_SIZE_ERROR (CC_BSV_CRYPTO_ERROR + 0x00000003)
- #define CC_BSV_INVALID_KEY_TYPE_ERROR (CC_BSV_CRYPTO_ERROR + 0x00000004)
- #define CC_BSV_INVALID_KEY_SIZE_ERROR (CC_BSV_CRYPTO_ERROR + 0x00000005)
- #define CC_BSV_ILLEGAL_KDF_LABEL_ERROR (CC_BSV_CRYPTO_ERROR + 0x00000006)
- #define CC_BSV_ILLEGAL_KDF_CONTEXT_ERROR (CC_BSV_CRYPTO_ERROR + 0x00000007)

- #define CC_BSV_CCM_INVALID_KEY_ERROR (CC_BSV_CRYPTO_ERROR + 0x00000008)
- #define CC_BSV_CCM_INVALID_NONCE_ERROR (CC_BSV_CRYPTO_ERROR + 0x00000009)
- #define CC_BSV_CCM_INVALID_ASSOC_DATA_ERROR (CC_BSV_CRYPTO_ERROR + 0x0000000A)
- #define CC_BSV_CCM_INVALID_TEXT_DATA_ERROR (CC_BSV_CRYPTO_ERROR + 0x0000000B)
- #define CC_BSV_CCM_INVALID_MAC_BUF_ERROR (CC_BSV_CRYPTO_ERROR + 0x0000000C)
- #define CC_BSV_CCM_DATA_OUT_DATA_IN_OVERLAP_ERROR (CC_BSV_CRYPTO_ERROR + 0x0000000D)
- #define CC_BSV_CCM_MAC_INVALID_ERROR (CC_BSV_CRYPTO_ERROR + 0x0000000E)
- #define CC_BSV_CCM_INVALID_MODE_ERROR (CC_BSV_CRYPTO_ERROR + 0x0000000F)
- #define CC_BSV_INVALID_OUT_POINTER_ERROR (CC_BSV_CRYPTO_ERROR + 0x00000010)
- #define CC_BSV_INVALID_CRYPTO_MODE_ERROR (CC_BSV_CRYPTO_ERROR + 0x00000011)
- #define CC_BSV_INVALID_IV_POINTER_ERROR (CC_BSV_CRYPTO_ERROR + 0x00000012)
- #define CC_BSV_INVALID_RESULT_BUFFER_POINTER_ERROR (CC_BSV_CRYPTO_ERROR + 0x00000013)
- #define CC_BSV_AO_WRITE_FAILED_ERR (CC_BSV_BASE_ERROR + 0x00000019)

Detailed description

This file defines the error return code types of the error codes from Secure Boot API's.

Macro definition documentation

#define CC_BSV_AO_WRITE_FAILED_ERR (CC_BSV_BASE_ERROR + 0x00000019)

AO write operation error.

#define CC_BSV_APB_PRIVILEG_IS_LOCKED_ERR (CC_BSV_BASE_ERROR + 0x00000013)

APB privilege mode is locked.

#define CC_BSV_APB_SECURE_IS_LOCKED_ERR (CC_BSV_BASE_ERROR + 0x00000012)

APB secure mode is locked.

#define CC_BSV_BASE_ERROR 0x0B000000

Definition for BSV error base.

#define CC_BSV_CCM_DATA_OUT_DATA_IN_OVERLAP_ERROR (CC_BSV_CRYPTO_ERROR + 0x0000000D)

Output and input data are overlapping.

#define CC_BSV_CCM_INVALID_ASSOC_DATA_ERROR (CC_BSV_CRYPTO_ERROR + 0x0000000A)

Invalid CCM associated data.

#define CC_BSV_CCM_INVALID_KEY_ERROR (CC_BSV_CRYPTO_ERROR + 0x00000008)

Invalid CCM key.

#define CC_BSV_CCM_INVALID_MAC_BUF_ERROR (CC_BSV_CRYPTO_ERROR + 0x0000000C)

Invalid CCM MAC buffer.

#define CC_BSV_CCM_INVALID_MODE_ERROR (CC_BSV_CRYPTO_ERROR + 0x0000000F)

Invalid CCM mode.

#define CC_BSV_CCM_INVALID_NONCE_ERROR (CC_BSV_CRYPTO_ERROR + 0x00000009)

Invalid CCM Nonce.

#define CC_BSV_CCM_INVALID_TEXT_DATA_ERROR (CC_BSV_CRYPTO_ERROR + 0x0000000B)

Invalid CCM text data.

#define CC_BSV_CCM_MAC_INVALID_ERROR (CC_BSV_CRYPTO_ERROR + 0x0000000E)

CCM MAC comparison failed.

#define CC_BSV_CRYPTO_ERROR 0x0C000000

Definition for BSV Crypto error base.

#define CC_BSV_ERASE_KEY_FAILED_ERR (CC_BSV_BASE_ERROR + 0x0000000B)

Erase key in OTP failed.

#define CC_BSV_FAILED_TO_SET_FATAL_ERR (CC_BSV_BASE_ERROR + 0x0000000E)

Device failed to move to fatal error state.

#define CC_BSV_FAILED_TO_SET_RMA_ERR (CC_BSV_BASE_ERROR + 0x0000000F)

Failed to set RMA LCS.

#define CC_BSV_HASH_NOT_PROGRAMMED_ERR (CC_BSV_BASE_ERROR + 0x00000007)

Hash boot key not programmed in the OTP.

#define CC_BSV_HBK_ZERO_COUNT_ERR (CC_BSV_BASE_ERROR + 0x00000008)

Illegal Hash boot key zero count in the OTP.

#define CC_BSV_ILLEGAL_ASSET_SIZE_ERR (CC_BSV_BASE_ERROR + 0x00000015)

Illegal asset size.

#define CC_BSV_ILLEGAL_ASSET_VAL_ERR (*CC_BSV_BASE_ERROR* + 0x00000016)

Illegal asset value.

#define CC_BSV_ILLEGAL_CIDR_ERR (*CC_BSV_BASE_ERROR* + 0x0000000D)

Illegal CIDR.

#define CC_BSV_ILLEGAL_HUK_VALUE_ERR (*CC_BSV_BASE_ERROR* + 0x00000002)

Illegal HUK value.

#define CC_BSV_ILLEGAL_INPUT_PARAM_ERR (*CC_BSV_BASE_ERROR* + 0x00000001)

Illegal input parameter.

#define CC_BSV_ILLEGAL_KCE_VALUE_ERR (*CC_BSV_BASE_ERROR* + 0x00000004)

Illegal Kce value.

#define CC_BSV_ILLEGAL_KCEICV_VALUE_ERR (*CC_BSV_BASE_ERROR* + 0x00000006)

Illegal Kceicv value.

#define CC_BSV_ILLEGAL_KCP_VALUE_ERR (*CC_BSV_BASE_ERROR* + 0x00000003)

Illegal Kcp value.

#define CC_BSV_ILLEGAL_KDF_CONTEXT_ERROR (*CC_BSV_CRYPTO_ERROR* + 0x00000007)

Illegal KDF context.

#define CC_BSV_ILLEGAL_KDF_LABEL_ERROR (*CC_BSV_CRYPTO_ERROR* + 0x00000006)

Illegal KDF label.

#define CC_BSV_ILLEGAL_KPICV_VALUE_ERR (*CC_BSV_BASE_ERROR* + 0x00000005)

Illegal Kpicv value.

#define CC_BSV_ILLEGAL_LCS_ERR (*CC_BSV_BASE_ERROR* + 0x00000009)

Illegal LCS.

#define CC_BSV_ILLEGAL_OPERATION_ERR (*CC_BSV_BASE_ERROR* + 0x00000014)

Illegal operation.

#define CC_BSV_ILLEGAL_PIDR_ERR (*CC_BSV_BASE_ERROR* + 0x0000000C)

Illegal PIDR.

#define CC_BSV_ILLEGAL_RMA_INDICATION_ERR (*CC_BSV_BASE_ERROR* + 0x00000010)

Illegal RMA indication.

#define CC_BSV_INVALID_CRYPTO_MODE_ERROR (CC_BSV_CRYPTO_ERROR + 0x00000011)

Illegal crypto mode.

#define CC_BSV_INVALID_DATA_IN_POINTER_ERROR (CC_BSV_CRYPTO_ERROR + 0x00000001)

Illegal data in pointer.

#define CC_BSV_INVALID_DATA_OUT_POINTER_ERROR (CC_BSV_CRYPTO_ERROR + 0x00000002)

Illegal data out pointer.

#define CC_BSV_INVALID_DATA_SIZE_ERROR (CC_BSV_CRYPTO_ERROR + 0x00000003)

Illegal data size.

#define CC_BSV_INVALID_IV_POINTER_ERROR (CC_BSV_CRYPTO_ERROR + 0x00000012)

Illegal IV pointer.

#define CC_BSV_INVALID_KEY_SIZE_ERROR (CC_BSV_CRYPTO_ERROR + 0x00000005)

Illegal key size.

#define CC_BSV_INVALID_KEY_TYPE_ERROR (CC_BSV_CRYPTO_ERROR + 0x00000004)

Illegal key type.

#define CC_BSV_INVALID_OUT_POINTER_ERROR (CC_BSV_CRYPTO_ERROR + 0x00000010)

Invalid out pointer.

#define CC_BSV_INVALID_RESULT_BUFFER_POINTER_ERROR (CC_BSV_CRYPTO_ERROR + 0x00000013)

Illegal result buffer pointer.

#define CC_BSV_KPICV_IS_LOCKED_ERR (CC_BSV_BASE_ERROR + 0x00000017)

Kpicv is locked.

#define CC_BSV_OTP_WRITE_CMP_FAIL_ERR (CC_BSV_BASE_ERROR + 0x0000000A)

OTP write compare failure.

#define CC_BSV_VER_IS_NOT_INITIALIZED_ERR (CC_BSV_BASE_ERROR + 0x00000011)

BSV version is not initialized.

2.6.9 bsv_otp_api.h file reference

Functions

- *CCError_t CC_BsvOTPWordRead* (unsigned long hwBaseAddress, uint32_t otpAddress, uint32_t *pOtpWord)

- *CCError_t CC_BsvOTPWordWrite* (unsigned long hwBaseAddress, uint32_t otpAddress, uint32_t otpWord)

Detailed description

This file contains functions that access the OTP memory for read and write operations.

Note

This implementation can be replaced by the partner depending on memory requirements.

Function documentation

CCError_t CC_BsvOTPWordRead (unsigned long `hwBaseAddress`, uint32_t `otpAddress`, uint32_t * `pOtpWord`)

This function retrieves a 32-bit OTP memory word from a given address.

Returns:

- CC_OK On success.
- A non-zero value from bsv_error.h on failure.

Parameters:

I/O	Parameter	Description
in	`hwBaseAddress`	HW registers base address.
in	`otpAddress`	Word address in the OTP memory to read from.
out	`pOtpWord`	The OTP memory word contents.

CCError_t CC_BsvOTPWordWrite (unsigned long `hwBaseAddress`, uint32_t `otpAddress`, uint32_t `otpWord`)

This function writes a 32-bit OTP memory word to a given address. Prior to writing, the function reads the current value in the OTP memory word, and performs bit-wise OR to generate the expected value. After writing, the word is read and compared to the expected value.

Note

This API is only a reference implementation. It should be replaced with an implementation that performs bit-wise programming of the bits of otpWord that should be changed from 0 to 1, in order to avoid over-programming.

Returns:

- CC_OK On success.
- A non-zero value from bsv_error.h on failure.

Parameters:

I/O	Parameter	Description
in	`hwBaseAddress`	HW registers base address.
in	`otpAddress`	Word address in the OTP memory to write to.
in	`otpWord`	The OTP memory word contents.

2.6.10 cc_bitops.h file reference

Macros

- #define CC_1K_SIZE_IN_BYTES   1024
- #define CC_BITS_IN_BYTE   8
- #define CC_BITS_IN_32BIT_WORD   32
- #define CC_32BIT_WORD_SIZE   (sizeof(uint32_t))
- #define BITMASK(mask_size)
- #define BITMASK_AT(mask_size,   mask_offset)   (BITMASK(mask_size) << (mask_offset))
- #define BITFIELD_GET(word,   bit_offset,   bit_size)   (((word) >> (bit_offset)) & BITMASK(bit_size))
- #define BITFIELD_SET(word,   bit_offset,   bit_size,   new_val)
- #define IS_ALIGNED(val,   align)   (((uintptr_t)(val) & ((align) - 1)) == 0)
- #define SWAP_ENDIAN(word)
- #define SWAP_TO_LE(word)   word
- #define SWAP_TO_BE(word)   SWAP_ENDIAN(word)
- #define ALIGN_TO_4BYTES(x)   (((unsigned long)(x) + (CC_32BIT_WORD_SIZE-1)) & ~(CC_32BIT_WORD_SIZE-1))
- #define IS_MULT(val,   mult)   (((val) & ((mult) - 1)) == 0)
- #define IS_NULL_ADDR(adr)   (!(adr))

Detailed description

This file defines bit fields operations macros.

Macro definition documentation

#define ALIGN_TO_4BYTES(x)   (((unsigned long)(x) + (CC_32BIT_WORD_SIZE-1)) & ~(CC_32BIT_WORD_SIZE-1))

Align X to uint32_t size.

#define BITFIELD_GET(word,   bit_offset,   bit_size)   (((word) >> (bit_offset)) & BITMASK(bit_size))

Definition for getting bits value from a word.

#define BITFIELD_SET(word,   bit_offset,   bit_size,   new_val)

```
do {    \
    word = ((word) & ~BITMASK_AT(bit_size, bit_offset)) |       \
        (((new_val) & BITMASK(bit_size)) << (bit_offset));  \
} while (0)
```

Definition for setting bits value from a word.

#define BITMASK(mask_size)

```
(((mask_size) < 32) ?   \
    ((1UL << (mask_size)) - 1) : 0xFFFFFFFFUL)
```

Definition for bitmask

#define BITMASK_AT(mask_size,   mask_offset)   (BITMASK(mask_size) << (mask_offset))

Definition for bitmask in a given offset.

#define CC_1K_SIZE_IN_BYTES   1024

Definition of 1KB in bytes.

#define CC_32BIT_WORD_SIZE   (sizeof(uint32_t))

Definition of number of bytes in a 32bits word.

#define CC_BITS_IN_32BIT_WORD   32

Definition of number of bits in a 32bits word.

#define CC_BITS_IN_BYTE   8

Definition of number of bits in a byte.

#define IS_ALIGNED(val,   align)   (((uintptr_t)(val) & ((align) - 1)) == 0)

Definition for is val aligned to "align" ("align" must be power of 2).

#define IS_MULT(val,   mult)   (((val) & ((mult) - 1)) == 0)

Definition for is val a multiple of "mult" ("mult" must be power of 2).

#define IS_NULL_ADDR(adr)   (!(adr))

Definition for is NULL address.

#define SWAP_ENDIAN(word)

```
(((word) >> 24) | (((word) & 0x00FF0000) >> 8) | \
   (((word) & 0x0000FF00) << 8) | (((word) & 0x000000FF) << 24))
```

Definition swap endianity for 32 bits word.

#define SWAP_TO_BE(word)   *SWAP_ENDIAN*(word)

Definition for swapping to BE.

#define SWAP_TO_LE(word)   word

Definition for swapping to LE.

2.6.11 cc_crypto_boot_defs.h file reference

Data structures

- struct *CCSbCertParserSwCompsInfo_t*

Macros

- #define *CC_SB_MAX_SIZE_NONCE_BYTES*   (2*sizeof(uint32_t))

Typedefs

- typedef uint8_t *CCSbNonce_t*[*CC_SB_MAX_SIZE_NONCE_BYTES*]

Enumerations

- enum *CCSbPubKeyIndexType_t* { *CC_SB_HASH_BOOT_KEY_0_128B* = 0, *CC_SB_HASH_BOOT_KEY_1_128B* = 1, *CC_SB_HASH_BOOT_KEY_256B* = 2, **CC_SB_HASH_BOOT_NOT_USED** = 0xF, **CC_SB_HASH_MAX_NUM** = 0x7FFFFFFF }
- enum *CCswCodeEncType_t* { *CC_SB_NO_IMAGE_ENCRYPTION* = 0, *CC_SB_ICV_CODE_ENCRYPTION* = 1, *CC_SB_OEM_CODE_ENCRYPTION* = 2, **CC_SB_CODE_ENCRYPTION_MAX_NUM** = 0x7FFFFFF }

- enum *CCswLoadVerifyScheme_t* { *CC_SB_LOAD_AND_VERIFY* = 0, *CC_SB_VERIFY_ONLY_IN_FLASH* = 1, *CC_SB_VERIFY_ONLY_IN_MEM* = 2, *CC_SB_LOAD_ONLY* = 3, **CC_SB_LOAD_VERIFY_MAX_NUM** = 0x7FFFFFFF }
- enum *CCswCryptoType_t* { *CC_SB_HASH_ON_DECRYPTED_IMAGE* = 0, *CC_SB_HASH_ON_ENCRYPTED_IMAGE* = 1, **CC_SB_CRYPTO_TYPE_MAX_NUM** = 0x7FFFFFFF }

Detailed description

This file contains SBROM definitions.

Macro definition documentation

#define CC_SB_MAX_SIZE_NONCE_BYTES (2*sizeof(uint32_t))

Maximal size of Secure Boot's nonce.

Typedef documentation

typedef uint8_t CCSbNonce_t[*CC_SB_MAX_SIZE_NONCE_BYTES*]

Table nonce used in composing iv for SW-component decryption.

Enumeration type documentation

enum *CCSbPubKeyIndexType_t*

HASH boot key definition.

Enumerator:

Enum	Description
`CC_SB_HASH_BOOT_KEY_0_128B`	128-bit truncated SHA256 digest of public key 0.
`CC_SB_HASH_BOOT_KEY_1_128B`	128-bit truncated SHA256 digest of public key 1.
`CC_SB_HASH_BOOT_KEY_256B`	256-bit SHA256 digest of public key.

enum *CCswCodeEncType_t*

SW image code encryption type definition.

Enumerator:

Enum	Description
`CC_SB_NO_IMAGE_ENCRYPTION`	Plain SW image.
`CC_SB_ICV_CODE_ENCRYPTION`	Use Kceicv for cipher SW image.
`CC_SB_OEM_CODE_ENCRYPTION`	Use Kce for cipher SW image.

enum *CCswCryptoType_t*

SW image crypto type.

Enumerator:

Enum	Description
`CC_SB_HASH_ON_DECRYPTED_IMAGE`	AES to HASH.
`CC_SB_HASH_ON_ENCRYPTED_IMAGE`	AES and HASH.

enum *CCswLoadVerifyScheme_t*

SW image load & verify scheme.

Enumerator:

Enum	Description
`CC_SB_LOAD_AND_VERIFY`	Load and Verify from flash to memory.
`CC_SB_VERIFY_ONLY_IN_FLASH`	Verify only in flash.
`CC_SB_VERIFY_ONLY_IN_MEM`	Verify only in memory.
`CC_SB_LOAD_ONLY`	Load only from flash to memory.

2.6.12 cc_hal_sb.h file reference

`#include "cc_hal_sb_plat.h"`

Functions

- *CCError_t* *SB_HalWaitInterrupt* (unsigned long hwBaseAddress, uint32_t data)
- void *SB_HalMaskInterrupt* (unsigned long hwBaseAddress, uint32_t data)
- void *SB_HalClearInterruptBit* (unsigned long hwBaseAddress, uint32_t data)

Detailed description

This file contains the functions that are used for the SBROM HAL layer.

Function documentation

void SB_HalClearInterruptBit (unsigned long `hwBaseAddress`, uint32_t `data`)

This function is used to clear bits in the Interrupt Clear Register (ICR).

Returns:

void

Parameters:

I/O	Parameter	Description
in	`hwBaseAddress`	CryptoCell base address.
in	`data`	The bits to clear.

void SB_HalMaskInterrupt (unsigned long `hwBaseAddress`, uint32_t `data`)

This function is used to mask a specific interrupt bit.

Returns:

void

Parameters:

I/O	Parameter	Description
in	`hwBaseAddress`	CryptoCell base address.
in	`data`	The bits to mask.

CCError_t SB_HalWaitInterrupt (unsigned long `hwBaseAddress`, uint32_t `data`)

This function is used to wait for the IRR interrupt signal (according to given bits). The existing implementation performs a "busy wait" on the IRR, and it should be adapted to the partner's system.

Returns:

A non-zero value from *secureboot_error.h* on failure.

Parameters:

I/O	Parameter	Description
in	`hwBaseAddress`	CryptoCell base address.
in	`data`	The interrupt bits to wait for.

2.6.13 cc_hal_sb_plat.h file reference

Macros

- #define *SB_HAL_READ_REGISTER*(addr, val) ((val) = (*((volatile uint32_t*)(addr))))
- #define *SB_HAL_WRITE_REGISTER*(addr, val) ((*((volatile uint32_t*)(addr))) = (unsigned long)(val))

Detailed description

This file contains definitions that are used for the SBROM HAL layer.

Macro definition documentation

#define SB_HAL_READ_REGISTER(addr, val) ((val) = (*((volatile uint32_t*)(addr))))

Read a 32-bit value from an Arm TrustZone CryptoCell memory-mapped register.

#define SB_HAL_WRITE_REGISTER(addr, val) ((*((volatile uint32_t*)(addr))) = (unsigned long)(val))

Write a 32-bit value to an Arm TrustZone CryptoCell memory-mapped register.

Note

This macro must be modified to make the operation synchronous, i.e. the write operation must complete and the new value must be written to the register before the macro returns. The mechanisms required to achieve this are architecture-dependent (e.g., the memory barrier in Arm architecture).

2.6.14 cc_pal_dma_defs.h file reference

Typedefs

- typedef void * *CC_PalDmaBufferHandle*

Enumerations

- enum *CCPalDmaBufferDirection_t* { *CC_PAL_DMA_DIR_NONE* = 0, *CC_PAL_DMA_DIR_TO_DEVICE* = 1, *CC_PAL_DMA_DIR_FROM_DEVICE* = 2, *CC_PAL_DMA_DIR_BI_DIRECTION* = 3, *CC_PAL_DMA_DIR_MAX*, *CC_PAL_DMA_DIR_RESERVE32* = 0x7FFFFFFF }

Detailed description

This file contains the platform-dependent DMA definitions.

2.6.15 cc_pal_sb_plat.h file reference

```
#include "cc_pal_types.h"
```

Typedefs

- typedef uint32_t *CCDmaAddr_t*
- typedef uint32_t *CCAddr_t*

Detailed description

This file contains the platform dependent definitions that are used in the SBROM code.

Typedef documentation

typedef uint32_t *CCAddr_t*

Definition of CryptoCell address type, can be 32 bits or 64 bits according to platform.

typedef uint32_t *CCDmaAddr_t*

Definition of DMA address type, can be 32 bits or 64 bits according to platform.

2.6.16 cc_pal_types.h file reference

```
#include "cc_pal_types_plat.h"
```

Macros

- #define *CC_SUCCESS* 0UL
- #define *CC_FAIL* 1UL
- #define *CC_OK* 0
- #define *CC_UNUSED_PARAM*(prm) ((void)prm)
- #define *CC_MAX_UINT32_VAL* (0xFFFFFFFF)
- #define *CC_MIN*(a, b) (((a) < (b)) ? (a) : (b))
- #define *CC_MAX*(a, b) (((a) > (b)) ? (a) : (b))
- #define *CALC_FULL_BYTES*(numBits) ((numBits)/*CC_BITS_IN_BYTE* + (((numBits) & (*CC_BITS_IN_BYTE*-1)) > 0))
- #define *CALC_FULL_32BIT_WORDS*(numBits) ((numBits)/*CC_BITS_IN_32BIT_WORD* + (((numBits) & (*CC_BITS_IN_32BIT_WORD*-1)) > 0))
- #define *CALC_32BIT_WORDS_FROM_BYTES*(sizeBytes) ((sizeBytes)/*CC_32BIT_WORD_SIZE* + (((sizeBytes) & (*CC_32BIT_WORD_SIZE*-1)) > 0))
- #define *ROUNDUP_BITS_TO_32BIT_WORD*(numBits) (*CALC_FULL_32BIT_WORDS*(numBits) * *CC_BITS_IN_32BIT_WORD*)
- #define *ROUNDUP_BITS_TO_BYTES*(numBits) (*CALC_FULL_BYTES*(numBits) * *CC_BITS_IN_BYTE*)
- #define *ROUNDUP_BYTES_TO_32BIT_WORD*(sizeBytes) (*CALC_32BIT_WORDS_FROM_BYTES*(sizeBytes) * *CC_32BIT_WORD_SIZE*)

Enumerations

- enum *CCBool* { *CC_FALSE* = 0, *CC_TRUE* = 1 }

Detailed description

This file contains platform-dependent definitions and types.

2.6.17 cc_pal_types_plat.h file reference

```
#include <stdint.h>
```

```
#include <stddef.h>
```

```
#include <stdbool.h>
```

Macros

- #define *CCError_t* *CCStatus*
- #define *CC_INFINITE* 0xFFFFFFFF
- #define *CEXPORT_C*
- #define *CIMPORT_C*

Typedefs

- typedef uintptr_t *CCVirtAddr_t*
- typedef uint32_t *CCBool_t*
- typedef uint32_t *CCStatus*

Detailed description

This file contains basic type definitions that are platform-dependent.

Macro definition documentation

#define CC_INFINITE 0xFFFFFFFF

Definition for infinite value.

#define CCError_t *CCStatus*

Definition for error.

#define CEXPORT_C

Definition for export.

#define CIMPORT_C

Definition for import.

Typedef documentation

typedef uint32_t *CCBool_t*

Definition for boolean type.

typedef uint32_t *CCStatus*

Definition for status.

typedef uintptr_t *CCVirtAddr_t*

Definition for virtual address.

2.6.18 cc_sec_defs.h file reference

Macros

- #define *HASH_BLOCK_SIZE_IN_WORDS*   16
- #define *HASH_RESULT_SIZE_IN_WORDS*   8
- #define *HASH_RESULT_SIZE_IN_BYTES*   32

Typedefs

- typedef uint32_t *CCHashResult_t*[*HASH_RESULT_SIZE_IN_WORDS*]

Detailed description

This file contains general hash definitions and types.

Macro definition documentation

#define HASH_BLOCK_SIZE_IN_WORDS   16

The hash block size in words.

#define HASH_RESULT_SIZE_IN_BYTES   32

SHA256 result size in bytes.

#define HASH_RESULT_SIZE_IN_WORDS   8

SHA256 result size in words.

Typedef documentation

typedef uint32_t CCHashResult_t[*HASH_RESULT_SIZE_IN_WORDS*]

Definition for hash result array.

2.6.19 sbrom_sec_debug_api.h file reference

```
#include "cc_pal_types_plat.h"
```

Macros

- #define *CC_BSV_SEC_DEBUG_SOC_ID_SIZE*   0x20

Functions

- *CCError_t* *CC_BsvSecureDebugSet* (unsigned long hwBaseAddress, uint32_t *pDebugCertPkg, uint32_t certPkgSize, uint32_t *pEnableRmaMode, uint32_t *pWorkspace, uint32_t workspaceSize)

Detailed description

This file contains a secure debug function that defines the allowed debug domains.

Macro definition documentation

#define CC_BSV_SEC_DEBUG_SOC_ID_SIZE   0x20

SOC ID size.

Function documentation

CCError_t CC_BsvSecureDebugSet (unsigned long `hwBaseAddress`, uint32_t * `pDebugCertPkg`, uint32_t `certPkgSize`, uint32_t * `pEnableRmaMode`, uint32_t * `pWorkspace`, uint32_t `workspaceSize`)

This API must always be called as part of the boot sequence, to define the allowed debug domains. It behaves as follows:

1. If the debug certificate is not NULL, it:
   a. verifies the OEM's public key against the public key hash in OTP memory (skipped in DM LCS).
   b. verifies the certificate chain included in the certificate package.
   c. compares the SOC_ID in the certificate against the SOC_ID retrieved by CC_BsvSocIDCompute (skipped in DM LCS).
2. If the debug certificate is successfully verified in step (1):
   a. If the certificate is used for Secure Debug - programs the DCU bits according to the domain mask in the certificate. The domain mask written to the DCU register is the result of an "AND" operation between the masks in the enabler and the developer debug certificates.
   b. If the certificate is used for RMA LCS entry - returns "Enable RMA mode" flag.
   c. If the debug certificate is NULL, or not successfully verified, or it is an RMA certificate it sets a predefined bit mask to the DCU register according to the LCS, and locks it for any further changes.

Returns:

- CC_OK On success.
- A non-zero value from bsv_error.h on failure.

Parameters:

I/O	Parameter	Description
in	`hwBaseAddress`	HW registers base address.
in	`pDebugCertPkg`	Pointer to the Secure Debug certificate package. NULL is a valid value.
in	`certPkgSize`	Byte size of the certificate package.
out	`pEnableRmaMode`	RMA entry flag. Non-zero indicates RMA LCS entry is required.
in	`pWorkspace`	Pointer buffer used internally
in	`workspaceSize`	Size of the buffer used internally, minimum size should be CC_SB_MIN_DBG_WORKSPACE_SIZE_IN_BYTES

2.6.20 secureboot_basetypes.h file reference

```
#include "cc_pal_types.h"
```

```
#include "cc_pal_types_plat.h"
```

Detailed description

This file contains basic type definitions for the Secure Boot.

2.6.21 secureboot_defs.h file reference

```
#include "cc_crypto_boot_defs.h"
#include "cc_sec_defs.h"
#include "cc_bitops.h"
#include "secureboot_basetypes.h"
#include "secureboot_gen_defs.h"
```

Data structures

- struct *CCSbCertInfo_t*
- struct *CCSbSwVersion_t*

Macros

- #define *SW_REC_SIGNED_DATA_SIZE_IN_BYTES* 44
- #define *SW_REC_NONE_SIGNED_DATA_SIZE_IN_BYTES* 8
- #define *CC_SW_COMP_NO_MEM_LOAD_INDICATION* 0xFFFFFFFFUL

Detailed description

This file contains basic type definitions for the Secure Boot in TrustZone platform.

Macro definition documentation

#define CC_SW_COMP_NO_MEM_LOAD_INDICATION 0xFFFFFFFFUL

Indication to load or not the component to memory.

#define SW_REC_NONE_SIGNED_DATA_SIZE_IN_BYTES 8

Definition for SW component additional data size.

#define SW_REC_SIGNED_DATA_SIZE_IN_BYTES 44

Definition for SW component.

2.6.22 secureboot_error.h file reference

Macros

- #define CC_SECUREBOOT_BASE_ERROR 0xF0000000
- #define CC_SECUREBOOT_LAYER_BASE_ERROR 0x01000000
- #define CC_SB_VERIFIER_LAYER_PREFIX 1
- #define CC_SB_DRV_LAYER_PREFIX 2
- #define CC_SB_SW_REVOCATION_LAYER_PREFIX 3
- #define CC_SB_NVM_LAYER_PREFIX 4
- #define CC_SB_HAL_LAYER_PREFIX 6
- #define CC_SB_RSA_LAYER_PREFIX 7
- #define CC_SB_VERIFIER_CERT_LAYER_PREFIX 8
- #define CC_SB_X509_CERT_LAYER_PREFIX 9
- #define CC_BOOT_IMG_VERIFIER_BASE_ERROR (CC_SECUREBOOT_BASE_ERROR + CC_SB_VERIFIER_LAYER_PREFIX*CC_SECUREBOOT_LAYER_BASE_ERROR)
- #define CC_NVM_BASE_ERROR (CC_SECUREBOOT_BASE_ERROR + CC_SB_NVM_LAYER_PREFIX*CC_SECUREBOOT_LAYER_BASE_ERROR)
- #define CC_SB_HAL_BASE_ERROR (CC_SECUREBOOT_BASE_ERROR + CC_SB_HAL_LAYER_PREFIX*CC_SECUREBOOT_LAYER_BASE_ERROR)
- #define CC_SB_RSA_BASE_ERROR (CC_SECUREBOOT_BASE_ERROR + CC_SB_RSA_LAYER_PREFIX*CC_SECUREBOOT_LAYER_BASE_ERROR)
- #define CC_BOOT_IMG_VERIFIER_CERT_BASE_ERROR (CC_SECUREBOOT_BASE_ERROR + CC_SB_VERIFIER_CERT_LAYER_PREFIX*CC_SECUREBOOT_LAYER_BASE_ERROR)
- #define CC_SB_X509_CERT_BASE_ERROR (CC_SECUREBOOT_BASE_ERROR + CC_SB_X509_CERT_LAYER_PREFIX*CC_SECUREBOOT_LAYER_BASE_ERROR)
- #define CC_SB_DRV_BASE_ERROR (CC_SECUREBOOT_BASE_ERROR + CC_SB_DRV_LAYER_PREFIX*CC_SECUREBOOT_LAYER_BASE_ERROR)
- #define CC_SB_HAL_FATAL_ERROR_ERR (CC_SB_HAL_BASE_ERROR + 0x00000001)
- #define CC_SB_DRV_ILLEGAL_INPUT_ERR (CC_SB_DRV_BASE_ERROR + 0x00000001)
- #define CC_SB_DRV_ILLEGAL_KEY_ERR (CC_SB_DRV_BASE_ERROR + 0x00000002)
- #define CC_SB_DRV_ILLEGAL_SIZE_ERR (CC_SB_DRV_BASE_ERROR + 0x00000003)

Detailed description

This file defines the error code types returned from the Secure Boot code.

Macro definition documentation

#define CC_BOOT_IMG_VERIFIER_BASE_ERROR (CC_SECUREBOOT_BASE_ERROR + CC_SB_VERIFIER_LAYER_PREFIX*CC_SECUREBOOT_LAYER_BASE_ERROR)

Boot Images Manager Base error = 0xF1000000.

#define CC_BOOT_IMG_VERIFIER_CERT_BASE_ERROR (CC_SECUREBOOT_BASE_ERROR + CC_SB_VERIFIER_CERT_LAYER_PREFIX*CC_SECUREBOOT_LAYER_BASE_ERROR)

Boot Images verifier cert Base error = 0xF8000000.

#define CC_NVM_BASE_ERROR (CC_SECUREBOOT_BASE_ERROR + CC_SB_NVM_LAYER_PREFIX*CC_SECUREBOOT_LAYER_BASE_ERROR)

NVM Base error = 0xF4000000.

#define CC_SB_DRV_BASE_ERROR (CC_SECUREBOOT_BASE_ERROR + CC_SB_DRV_LAYER_PREFIX*CC_SECUREBOOT_LAYER_BASE_ERROR)

Cryptographic driver error base = 0xF2000000.

#define CC_SB_DRV_ILLEGAL_INPUT_ERR (CC_SB_DRV_BASE_ERROR + 0x00000001)

Illegal input.

#define CC_SB_DRV_ILLEGAL_KEY_ERR (CC_SB_DRV_BASE_ERROR + 0x00000002)

Illegal key.

#define CC_SB_DRV_ILLEGAL_SIZE_ERR (CC_SB_DRV_BASE_ERROR + 0x00000003)

Illegal size.

#define CC_SB_DRV_LAYER_PREFIX 2

Secure Boot driver layer prefix number.

#define CC_SB_HAL_BASE_ERROR (CC_SECUREBOOT_BASE_ERROR + CC_SB_HAL_LAYER_PREFIX*CC_SECUREBOOT_LAYER_BASE_ERROR)

HAL Base error = 0xF6000000.

#define CC_SB_HAL_FATAL_ERROR_ERR (CC_SB_HAL_BASE_ERROR + 0x00000001)

HAL fatal error.

#define CC_SB_HAL_LAYER_PREFIX 6

Secure Boot HAL layer prefix number.

#define CC_SB_NVM_LAYER_PREFIX 4

Secure Boot NVM layer prefix number.

#define CC_SB_RSA_BASE_ERROR (CC_SECUREBOOT_BASE_ERROR + CC_SB_RSA_LAYER_PREFIX*CC_SECUREBOOT_LAYER_BASE_ERROR)

RSA Base error = 0xF7000000.

#define CC_SB_RSA_LAYER_PREFIX 7

Secure Boot RSA layer prefix number.

#define CC_SB_SW_REVOCATION_LAYER_PREFIX 3

Secure Boot revocation layer prefix number.

#define CC_SB_VERIFIER_CERT_LAYER_PREFIX 8

Secure Boot certificate verifier layer prefix number.

#define CC_SB_VERIFIER_LAYER_PREFIX 1

Secure Boot verifier layer prefix number.

#define CC_SB_X509_CERT_BASE_ERROR (CC_SECUREBOOT_BASE_ERROR + CC_SB_X509_CERT_LAYER_PREFIX*CC_SECUREBOOT_LAYER_BASE_ERROR)

x509 Base error = 0xF9000000.

#define CC_SB_X509_CERT_LAYER_PREFIX 9

Secure Boot X509 certificate layer prefix number.

#define CC_SECUREBOOT_BASE_ERROR 0xF0000000

The definitions of the error number space used for the different modules.

Secure Boot base error number.

#define CC_SECUREBOOT_LAYER_BASE_ERROR 0x01000000

Secure Boot layer error number.

2.6.23 secureboot_gen_defs.h file reference

```
#include "cc_pal_sb_plat.h"

#include "secureboot_basetypes.h"

#include "cc_sec_defs.h"
```

Typedefs

- typedef uint32_t *CCSbCertPubKeyHash_t*[*HASH_RESULT_SIZE_IN_WORDS*]
- typedef uint32_t *CCSbCertSocId_t*[*HASH_RESULT_SIZE_IN_WORDS*]
- typedef uint32_t(* *CCSbFlashReadFunc*) (*CCAddr_t* flashAddress, uint8_t *memDst, uint32_t sizeToRead, void *context)
- typedef uint32_t(* *CCBsvFlashWriteFunc*) (*CCAddr_t* flashAddress, uint8_t *memSrc, uint32_t sizeToWrite, void *context)

Detailed description

This file contains all of the definitions and structures that are used for the Secure Boot.

Typedef documentation

typedef uint32_t(* CCBsvFlashWriteFunc) (CCAddr_t flashAddress, uint8_t *memSrc, uint32_t sizeToWrite, void *context)

Typedef of Flash write function pointer, to be implemented by the partner. Used for writing back authenticated and decrypted SW modules to Flash memory.

Parameters:

I/O	Parameter	Description
in	`flashAddress`	Address for writing to Flash memory.
out	`memSrc`	Pointer to the RAM source to write the data from.
in	`sizeToWrite`	Size to write in bytes.
in	`context`	For partner use.

typedef uint32_t CCSbCertPubKeyHash_t[HASH_RESULT_SIZE_IN_WORDS]

Definition of public key hash array.

typedef uint32_t CCSbCertSocId_t[HASH_RESULT_SIZE_IN_WORDS]

Definition of SOC id array.

typedef uint32_t(* CCSbFlashReadFunc) (CCAddr_t flashAddress, uint8_t *memDst, uint32_t sizeToRead, void *context)

Typedef of Flash read function pointer, to be implemented by the partner. Used for reading the certificates and SW modules from Flash memory.

Parameters:

I/O	Parameter	Description
in	`flashAddress`	Address for reading from flash memory.
out	`memDst`	Pointer to the RAM destination address to write the data to.
in	`sizeToRead`	The size to read in bytes.
in	`context`	For partner use.

Appendix A: Revisions

Table A-1: Revision 0000-00

Change	Location	Affects
First release	-	r0p0

Table A-2: Differences between revision 0000-00 and revision 0000-01

Change	Location	Affects
Updated note.	*CC_SB_MIN_WORKSPACE_SIZE_IN_BYTES*	r0p0
Added `CC_BSV_AO_WRITE_FAILED_ERR`.	*bsv_error.h file reference*	r0p0

Table A-3: Differences between revision 0000-01 and revision 0000-02

Change	Location	Affects
Added several standards to the list.	*Referenced standards*	r0p0-00eac0
Added the *Runtime APIs* chapter.	Entire document.	r0p0-00eac0