PSA Crypto Driver Model Specification

Draft 0.3

Contents

Chapter 1

Introduction

Arm's Platform Security Architecture (PSA) is a holistic set of threat models, security analyses, hardware and firmware architecture specifications, and an open source firmware reference implementation. PSA provides a recipe, based on industry best practice, that allows security to be consistently designed in, at both a hardware and firmware level.

The PSA Cryptographic Driver Model described in this document is an important component of the PSA that provides common function names and behaviors that aide in developing PSA Cryptographic API implementations that are easily extended with hardware support, as well as providing common interfaces that hardware vendors can support to enable use by those various implementations.

Background

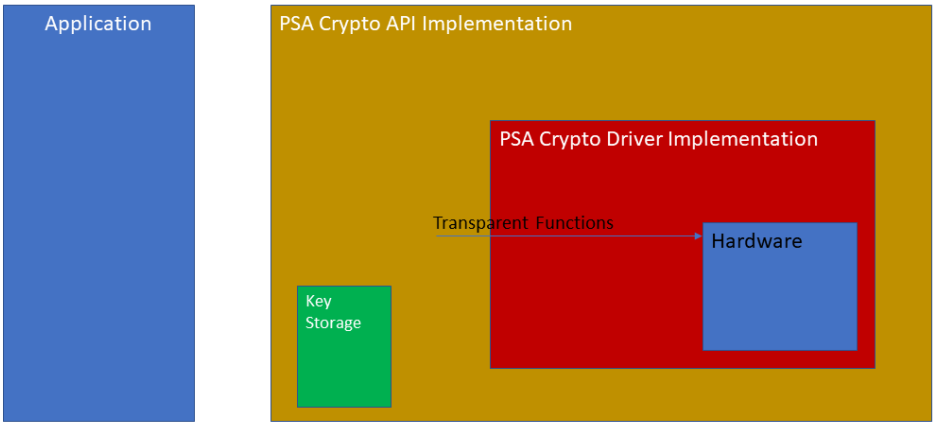
The PSA Cryptographic API provides the full suite of modern cryptographic primitives for resource-constrained devices. It does not require the caller of the API to have access to the key material, instead using opaque key handles.

The PSA Cryptographic Driver Model described in this document is a companion API to the PSA Cryptographic API. It describes functions, structures, and naming conventions that hardware vendors should implement to allow PSA Crypto API implementations to easily use the hardware.

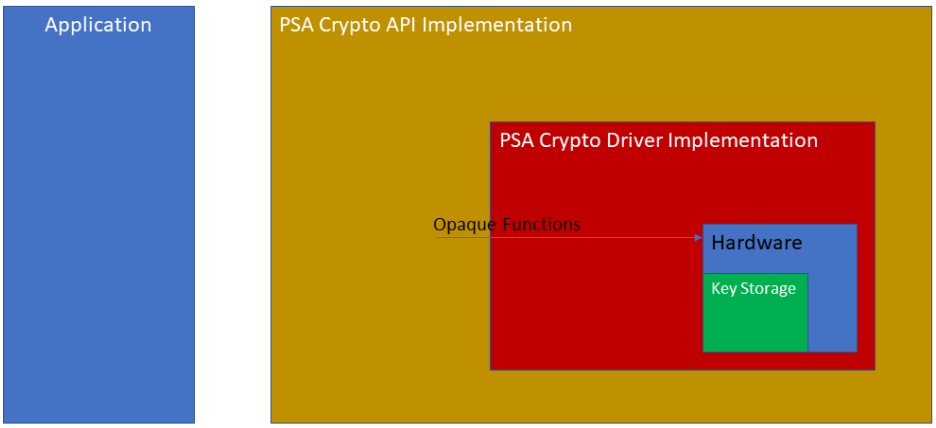
There are two broad types of functions and structures in this Model:

- The first provide access to cryptographic primitives while keeping the key material within the boundary of the hardware. These are referred to as opaque functions, and are intended to be used with secure elements.
- The second require that the caller have access to the key material and provide it as a parameter in the function call. These are referred to as transparent functions, and are intended to be used with cryptographic accelerators without secure element features.

In the transparent functions, the key material is kept inside of the PSA Crypto implementation, and passed in as a parameter in the function calls:



For the opaque functions, the key material is intended to be kept inside of the hardware secure element:



Since it is anticipated that PSA Crypto API implementations will often want to support multiple secure elements on a given platform, opaque key functions are grouped according to function into structures. PSA Crypto API implementations are then free to have as many instances of these structures as is necessary for the number and types of hardware implementations available.

On the other hand, it is not anticipated that PSA Crypto API implementations will want to support multiple accelerators with duplicate functionality in the same build. Therefore, implementers of transparent functions should obey the naming conventions described in this document. This will allow PSA Crypto API implementations to simply include a header file, and link against a binary of the driver implementation. This has the advantage of adding no runtime overhead to the function calls.

Theory of Operation

A PSA Cryptographic API implementation should have a number of "substitution points". At a substitution point, the code can do one of three things, based on static configuration:

1. Hardware only: call the driver function
2. Runtime fallback: call the driver function, if available. Else, continue to a software implementation of the algorithm.
3. Software only: Do not call the driver function, instead implement the whole thing in software
4. Neither software or hardware - algorithm not supported

For an opaque key, there is a different set of driver functions, one set per external secure element type.

Substitution points can be "nested". For example there is a substitution point for AES_CBC. If there is no suitable driver then there is another substitution point for AES_ECB, with the driver code only doing the block permutation and the software doing the padding and chaining.

Driver functions take “simple” inputs. E.g. for a block cipher driver the high-level code guarantees that the buffer sizes are multiple of the block size. The high-level code guarantees that buffers are big enough.

Sample Function Flows

Message Authentication Codes

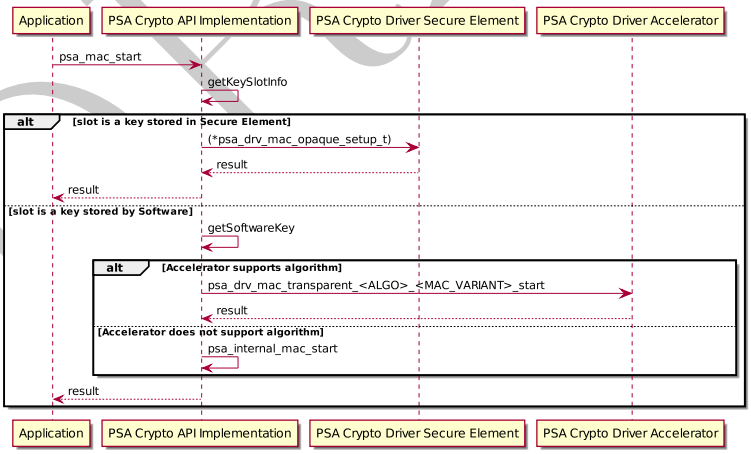


Figure 1.1 Example psa_mac_start Implementation Sequence

Sample PSA Crypto API function implementation:

```
psa_status_t psa_mac_start(psa_mac_operation_t *operation,
                           psa_key_slot_t key,
                           psa_algorithm_t alg) {
    psa_status_t return_value = PSA_SUCCESS;
    // Find the key slot information
    key_slot_t *p_slot = ...;
    if ( p_slot->external != NULL ) {
        // a Secure Element has the key data
        return_value = p_slot->external->p_mac->p_setup( &(operation->external_context),
```

```
                                                    p_slot->hardware_key_slot,
                                                    alg );
    } else {
        // Software has the key data
        // see if we can use an accelerator
        return_value = PSA_ERROR_NOT_SUPPORTED;
#if PSA_DRV_HAS_HMAC_SHA1
        if ( pslot->type == PSA_KEY_TYPE_HMAC_SHA1 ) {

            return_value = psa_drv_mac_transparent_SHA1_HMAC_start(...);
        }
#endif
#if PSA_DRV_HAS_HMAC_SHA256
    ...
#endif
if PSA_DRV_HAS_HMAC_XXX
    ...
#endif
        if ( return_value == PSA_ERROR_NOT_SUPPORTED) {
            // we apparently don't have a HW implementation, use software
            return_value = psa_crysoft_hmac_setup(...);
        }
    }
    return return_value;
}
```

Symmetric Block Ciphers

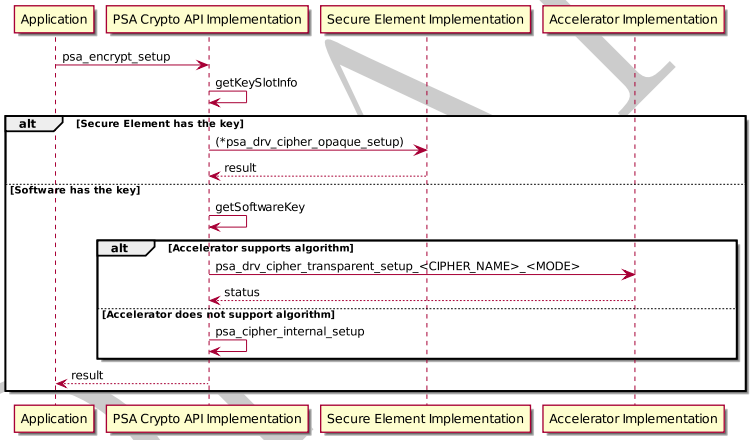


Figure 1.2 Proposed psa_encrypt_setup Implementation Sequence

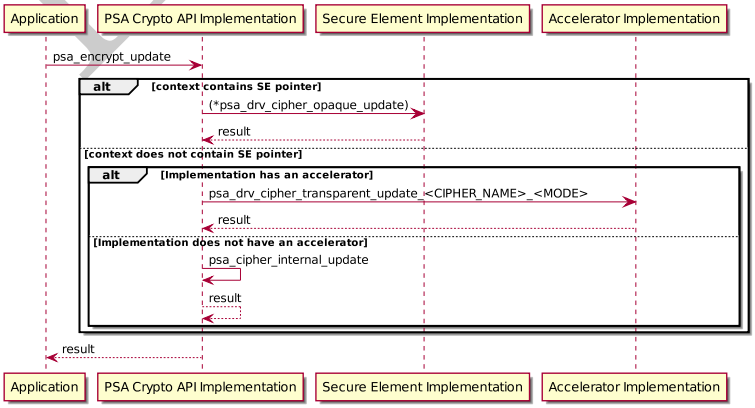


Figure 1.3 Proposed psa_cipher_update Implementation Sequence

Sample `psa_encrypt_setup` implementation:

```
psa_status_t psa_encrypt_setup(psa_cipher_operation_t *operation,
                               psa_key_slot_t psa,
                               psa_algorithm_t alg) {
    psa_status_t return_value = PSA_SUCCESS;
    // Find the key slot information
    key_slot_t *p_slot = ...;
    if ( p_slot->external != NULL ) {
        // A Secure Element has the key data
        return_value = p_slot->external->p_cipher->p_setup( &(operation->external_context),
                                                            p_slot->hardware_key_id,
                                                            alg,
                                                            PSA_CRYPTO_DRIVER_ENCRYPT );
    } else {
        // We've got the key data
        // See if we can use an accelerator
        return_value = PSA_ERROR_NOT_SUPPORTED;
        if ( p_slot->type == PSA_KEY_TYPE_AES ) {
            switch (alg) {
#if PSA_DRV_HAS_AES_CBC
                case PSA_ALG_CBC:
                    return_value = psa_drv_cipher_transparent_setup_aes_cbc(operation->external_context,
                                                                            PSA_CRYPTO_DRIVER_ENCRYPT,
                                                                            p_slot->key_data,
                                                                            p_slot->key_data_size);

                    break;
#endif // PSA_DRV_HAS_AES_CBC
#if PSA_DRV_HAS_AES_CTR
                case PSA_ALG_CTR:
                    return_value = psa_drv_cipher_transparent_setup_aes_ctr(operation->external_context,
                                                                            PSA_CRYPTO_DRIVER_ENCRYPT,
                                                                            p_slot->key_data,
                                                                            p_slot->key_data_size);

                    break;
#endif // PSA_DRV_HAS_AES_CTR
#if PSA_DRV_HAS_AES_XXX
                case PSA_ALG_XXX:
                    ...
                    break;
#endif // PSA_DRV_HAS_AES_XXX
                default:
                    break;
            }
        } else if ( p_slot->type == PSA_KEY_TYPE_DES ) {
            switch (alg) {
#if PSA_DRV_HAS_DES_CBC
                case PSA_ALG_CBC:
                    return_value = psa_drv_cipher_transparent_setup_des_cbc(operation->external_context,
                                                                            PSA_CRYPTO_DRIVER_ENCRYPT,
                                                                            p_slot->key_data,
                                                                            p_slot->key_data_size);

                    break;
#endif // PSA_DRV_HAS_DES_CBC
if PSA_DRV_HAS_DES_CTR
                case PSA_ALG_CTR:
                    return_value = psa_drv_cipher_transparent_setup_des_ctr(operation->external_context,
                                                                            PSA_CRYPTO_DRIVER_ENCRYPT,
                                                                            p_slot->key_data,
                                                                            p_slot->key_data_size);

                    break;
#endif // PSA_DRV_HAS_DES_CTR
#if PSA_DRV_HAS_DES_XXX
                ...
#endif // PSA_DRV_HAS_DES_XXX
                default:
                    break;
            }
        } else if (...) {
            ...
        }
        if (return_value == PSA_ERROR_NOT_SUPPORTED) {
            // now we try to use the software implementation
            return_value = psa_crysoft_aes_setup(operation->external_context,
                                                 PSA_CRYPTO_DRIVER_ENCRYPT,
                                                 p_slot->key_data,
                                                 p_slot->key_data_size);
        }
    }
    return return_value;
}
```

Sample `psa_cipher_update` implementation:

```
psa_status_t psa_cipher_update(psa_cipher_operation_t *operation,
```

```
                                const uint8_t *input,
                                size_t input_length,
                                uint8_t *output,
                                size_t output_size,
                                size_t *output_length) {
    key_slot_t *p_slot = ...;
    if ( p_slot->external != NULL ) {
        return_value = p_slot->opaque.cipher.p_update( operation->external_context,
                                                       input,
                                                       input_size,
                                                       output,
                                                       output_size,
                                                       output_length );
    } else {
        return_vlaue = PSA_ERROR_NOT_SUPPORTED;
        if (p_slot->type == PSA_KEY_TYPE_AES) {
            switch (alg) {
#if PSA_DRV_HAS_AES_CBC
                case PSA_ALG_CBC:
                    return_value = psa_drv_cipher_transparent_update_aes_cbc(...);
                    break;
#endif // PSA_DRV_HAS_AES_CBC
#if PSA_DRV_HAS_AES_CTR
                case PSA_ALG_CTR:
                    return_value = psa_drv_cipher_transparent_update_aes_ctr(...);
                    break;
#endif // PSA_DRV_HAS_AES_CTR
#if PSA_DRV_HAS_AES_XXX
                case PSA_ALG_XXX:
                    ...
                    break;
#endif // PSA_DRV_HAS_AES_XXX
                default:
                    break;
            }
        } else if ( p_slot->type == PSA_KEY_TYPE_DES ) {
            switch (alg) {
#if PSA_DRV_HAS_DES_CBC
                case PSA_ALG_CBC:
                    return_value = psa_drv_cipher_transparent_update_des_cbc(...);
                    break;
#endif // PSA_DRV_HAS_DES_CBC
if PSA_DRV_HAS_DES_CTR
                case PSA_ALG_CTR:
                    return_value = psa_drv_cipher_transparent_update_des_ctr(...);
                    break;
#endif // PSA_DRV_HAS_DES_CTR
#if PSA_DRV_HAS_DES_XXX
                ...
#endif // PSA_DRV_HAS_DES_XXX
                default:
                    break;
            }
        } else if (...) {
            ...
        }
        if (return_value == PSA_ERROR_NOT_SUPPORTED) {
            // now we try to use the software implementation
            return_value = psa_crysoft_cipher_update(...);
        }
    }
    return return_value;
}
```


Entropy Generation

The PSA Crypto API implementation includes a random generator to generate material such as keys and initialization vectors. It may also use the random generator internally to generate data such as nonces and blinding values. Arm recommends that PSA platforms include one or more entropy sources, preferably a hardware random number generator (HRNG, also called TRNG "true random number generator").

The PSA driver model includes an interface for entropy sources in the form of a function type `psa_drv_entropy_get_bits_t`. A driver can provide one or more functions with this type. The generic part of a PSA Crypto API implementation uses these entropy sources to seed a cryptographically secure pseudorandom number generator (CSPRNG, also called DRBG for deterministic random bit generator). The PSA Crypto API will call the available entropy source functions during startup, and may call the functions again later if it needs to reseed its internal random generator.

A get-entropy function of type `psa_drv_entropy_get_bits_t`) has input parameters for a buffer and the buffer size, and an output paramter for the estimated amount of received entropy. On each call, the get-entropy function fills part or all of the buffer with random data, and reports an estimate of the number of bits of entropy. If the data is perfectly random then this estimate is the size of the data written by the function is bits, but most entropy sources are not perfectly uniform so will have a lower estimate. The entropy does not need to be distributed evenly in the returned buffer, so the generic part of the PSA Crypto API must not make any assuption about the location of the randomness in the output buffer.

A simplified sample entropy retrieval implementation that only uses one source is shown below:

```
static int psa_crypto_implementation_get_entropy() {
    #define NUM_NEEDED_ENTROPY_BITS 512
    struct psa_drv_entropy_t *p_entropy_table = SetupEntropyFunctions(...);

    uint8_t entropy_buffer[(NUM_NEEDED_ENTROPY_BITS + 7) / 8];

    struct psa_drv_entropy_context_t *p_entropy_context = p_entropy_table->
      p_init(...);
    if ( p_entropy_context == NULL ) {
        return ERROR;
    }
    uint32_t total_received_entropy_bits = 0;
    while (total_received_entropy_bits < NUM_NEEDED_ENTROPY_BITS) {
        uint32_t received_entropy_bits = 0;
        psa_status_t status = p_entropy_table->p_get_bits(p_entropy_context, entropy_buffer,
      sizeof(entropy_buffer), &received_entropy_bits);
        if (status != PSA_SUCCESS) {
            return ERROR;
        }
        SeedDRBG(entropy_buffer, sizeof(entropy_buffer));
        total_received_entropy_bits += received_entropy_bits;
    }
    return SUCCESS;
}
```

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

Module Documentation

4.1 Opaque Message Authentication Code

4.1.1 Detailed Description

Generation and authentication of Message Authentication Codes (MACs) using opaque keys can be done either as a single function call (via the `psa_drv_mac_opaque_generate_t` or `psa_drv_mac_opaque_verify_t` functions), or in parts using the following sequence:

- `psa_drv_mac_opaque_setup_t`
- `psa_drv_mac_opaque_update_t`
- `psa_drv_mac_opaque_update_t`
- ...
- `psa_drv_mac_opaque_finish_t` or `psa_drv_mac_opaque_finish_verify_t`

If a previously started Opaque MAC operation needs to be terminated, it should be done so by the `psa_drv_mac_opaque_abort_t`. Failure to do so may result in allocated resources not being freed or in other undefined behavior.

4.1.2 Types

4.1.2.1 typedef psa_status_t(* psa_drv_mac_opaque_setup_t) (void *p_context, psa_key_slot_t key_slot, psa_algorithm_t algorithm)

A function that starts a MAC operation for a PSA Crypto Driver implementation using an opaque key.

Parameters

`in,out`	p_context	A structure that will contain the hardware-specific MAC context
`in`	key_slot	The slot of the key to be used for the operation
`in`	algorithm	The algorithm to be used to underly the MAC operation

Return values

PSA_SUCCESS	Success.

4.1.2.2 typedef psa_status_t(* psa_drv_mac_opaque_update_t) (void *p_context, const uint8_t *p_input, size_t input_length)

A function that continues a previously started MAC operation using an opaque key.

Parameters

in,out	p_context	A hardware-specific structure for the previously-established MAC operation to be continued
in	p_input	A buffer containing the message to be appended to the MAC operation
in	input_length	The size in bytes of the input message buffer

4.1.2.3 typedef psa_status_t(* psa_drv_mac_opaque_finish_t) (void *p_context, uint8_t *p_mac, size_t mac_size, size_t *p_mac_length)

a function that completes a previously started MAC operation by returning the resulting MAC using an opaque key

Parameters

in,out	p_context	A hardware-specific structure for the previously started MAC operation to be finished
out	p_mac	A buffer where the generated MAC will be placed
in	mac_size	The size in bytes of the buffer that has been allocated for the output buffer
out	p_mac_length	After completion, will contain the number of bytes placed in the p_mac buffer

Return values

PSA_SUCCESS	Success.

4.1.2.4 typedef psa_status_t(* psa_drv_mac_opaque_finish_verify_t) (void *p_context, const uint8_t *p_mac, size_t mac_length)

A function that completes a previously started MAC operation by comparing the resulting MAC against a known value using an opaque key.

Parameters

in,out	p_context	A hardware-specific structure for the previously started MAC operation to be fiinished
in	p_mac	The MAC value against which the resulting MAC will be compared against
in	mac_length	The size in bytes of the value stored in p_mac

Return values

PSA_SUCCESS	The operation completed successfully and the MACs matched each other
PSA_ERROR_INVALID_SIGNATURE	The operation completed successfully, but the calculated MAC did not match the provided MAC

4.1.2.5 typedef psa_status_t(* psa_drv_mac_opaque_abort_t) (void *p_context)

A function that aborts a previous started opaque-key MAC operation.

Parameters

`in,out`	p_context	A hardware-specific structure for the previously started MAC operation to be aborted

4.1.2.6 typedef psa_status_t(* psa_drv_mac_opaque_generate_t) (const uint8_t *p_input, size_t input_length, psa_key_slot_t key_slot, psa_algorithm_t alg, uint8_t *p_mac, size_t mac_size, size_t *p_mac_length)

A function that performs a MAC operation in one command and returns the calculated MAC using an opaque key.

Parameters

`in`	p_input	A buffer containing the message to be MACed
`in`	input_length	The size in bytes of `p_input`
`in`	key_slot	The slot of the key to be used
`in`	alg	The algorithm to be used to underlie the MAC operation
`out`	p_mac	A buffer where the generated MAC will be placed
`in`	mac_size	The size in bytes of the `p_mac` buffer
`out`	p_mac_length	After completion, will contain the number of bytes placed in the `output` buffer

Return values

PSA_SUCCESS	Success.

4.1.2.7 typedef psa_status_t(* psa_drv_mac_opaque_verify_t) (const uint8_t *p_input, size_t input_length, psa_key_slot_t key_slot, psa_algorithm_t alg, const uint8_t *p_mac, size_t mac_length)

A function that performs an MAC operation in one command and compare the resulting MAC against a known value using an opaque key.

Parameters

in	p_input	A buffer containing the message to be MACed
in	input_length	The size in bytes of input
in	key_slot	The slot of the key to be used
in	alg	The algorithm to be used to underlie the MAC operation
in	p_mac	The MAC value against which the resulting MAC will be compared against
in	mac_length	The size in bytes of mac

Return values

PSA_SUCCESS	The operation completed successfully and the MACs matched each other
PSA_ERROR_INVALID_SIGNATURE	The operation completed successfully, but the calculated MAC did not match the provided MAC

4.2 Transparent Message Authentication Code

4.2.1 Detailed Description

Generation and authentication of Message Authentication Codes (MACs) using transparent keys can be done either as a single function call (via the `psa_drv_mac_transparent_generate_t` or `psa_drv_mac_transparent_verify_t` functions), or in parts using the following sequence:

- `psa_drv_mac_transparent_setup_t`
- `psa_drv_mac_transparent_update_t`
- `psa_drv_mac_transparent_update_t`
- ...
- `psa_drv_mac_transparent_finish_t` or `psa_drv_mac_transparent_finish_verify_t`

If a previously started Transparent MAC operation needs to be terminated, it should be done so by the `psa_drv_mac_transparent_abort_t`. Failure to do so may result in allocated resources not being freed or in other undefined behavior.

4.2.2 Types

4.2.2.1 typedef struct psa_drv_mac_transparent_context_s psa_drv_mac_transparent_context_t

The hardware-specific transparent-key MAC context structure.

The contents of this structure are implementation dependent and are therefore not described here.

4.2.2.2 typedef psa_status_t(* psa_drv_mac_transparent_setup_t) (psa_drv_mac_transparent_context_t *p_context, const uint8_t *p_key, size_t key_length)

The function prototype for the setup operation of a transparent-key MAC operation.

Functions that implement the prototype should be named in the following convention:

```
psa_drv_mac_transparent_<ALGO>_<MAC_VARIANT>_setup
```

Where `ALGO` is the name of the underlying primitive, and `MAC_VARIANT` is the specific variant of a MAC operation (such as HMAC or CMAC)

Parameters

`in,out`	p_context	A structure that will contain the hardware-specific MAC context
`in`	p_key	A buffer containing the cleartext key material to be used in the operation
`in`	key_length	The size in bytes of the key material

Return values

PSA_SUCCESS	Success.

4.2.2.3 typedef psa_status_t(* psa_drv_mac_transparent_update_t) (psa_drv_mac_transparent_context_t *p_context, const uint8_t *p_input, size_t input_length)

The function prototype for the update operation of a transparent-key MAC operation.

Functions that implement the prototype should be named in the following convention:

```
psa_drv_mac_transparent_<ALGO>_<MAC_VARIANT>_update
```

Where `ALGO` is the name of the underlying algorithm, and `MAC_VARIANT` is the specific variant of a MAC operation (such as HMAC or CMAC)

Parameters

`in,out`	p_context	A hardware-specific structure for the previously-established MAC operation to be continued
`in`	p_input	A buffer containing the message to be appended to the MAC operation
`in`	input_length	The size in bytes of the input message buffer

4.2.2.4 typedef psa_status_t(* psa_drv_mac_transparent_finish_t) (psa_drv_mac_transparent_context_t *p_context, uint8_t *p_mac, size_t mac_length)

The function prototype for the finish operation of a transparent-key MAC operation.

Functions that implement the prototype should be named in the following convention:

```
psa_drv_mac_transparent_<ALGO>_<MAC_VARIANT>_finish
```

Where `ALGO` is the name of the underlying algorithm, and `MAC_VARIANT` is the specific variant of a MAC operation (such as HMAC or CMAC)

Parameters

`in,out`	p_context	A hardware-specific structure for the previously started MAC operation to be finished
`out`	p_mac	A buffer where the generated MAC will be placed
`in`	mac_length	The size in bytes of the buffer that has been allocated for the `p_mac` buffer

Return values

PSA_SUCCESS	Success.

4.2.2.5 typedef psa_status_t(* psa_drv_mac_transparent_finish_verify_t) (psa_drv_mac_transparent_context_t *p_context, const uint8_t *p_mac, size_t mac_length)

The function prototype for the finish and verify operation of a transparent-key MAC operation.

Functions that implement the prototype should be named in the following convention:

```
psa_drv_mac_transparent_<ALGO>_<MAC_VARIANT>_finish_verify
```

Where `ALGO` is the name of the underlying algorithm, and `MAC_VARIANT` is the specific variant of a MAC operation (such as HMAC or CMAC)

Parameters

`in,out`	p_context	A hardware-specific structure for the previously started MAC operation to be verified and finished
`in`	p_mac	A buffer containing the MAC that will be used for verification
`in`	mac_length	The size in bytes of the data in the `p_mac` buffer

Return values

PSA_SUCCESS	The operation completed successfully and the comparison matched

4.2.2.6 typedef psa_status_t(* psa_drv_mac_transparent_abort_t) (psa_drv_mac_transparent↩ _context_t *p_context)

The function prototype for the abort operation for a previously started transparent-key MAC operation.

Functions that implement the prototype should be named in the following convention:

```
psa_drv_mac_transparent_<ALGO>_<MAC_VARIANT>_abort
```

Where `ALGO` is the name of the underlying algorithm, and `MAC_VARIANT` is the specific variant of a MAC operation (such as HMAC or CMAC)

Parameters

`in,out`	p_context	A hardware-specific structure for the previously started MAC operation to be aborted

4.2.2.7 typedef psa_status_t(* psa_drv_mac_transparent_t) (const uint8_t *p_input, size_t input_length, const uint8_t *p_key, size_t key_length, psa_algorithm_t alg, uint8_t *p_mac, size_t mac_length)

The function prototype for a one-shot operation of a transparent-key MAC operation.

Functions that implement the prototype should be named in the following convention:

```
psa_drv_mac_transparent_<ALGO>_<MAC_VARIANT>
```

Where ALGO is the name of the underlying algorithm, and MAC_VARIANT is the specific variant of a MAC operation (such as HMAC or CMAC)

Parameters

in	p_input	A buffer containing the data to be MACed
in	input_length	The length in bytes of the p_input data
in	p_key	A buffer containing the key material to be used for the MAC operation
in	key_length	The length in bytes of the p_key data
in	alg	The algorithm to be performed
out	p_mac	The buffer where the resulting MAC will be placed upon success
in	mac_length	The length in bytes of the p_mac buffer

4.2.2.8 typedef psa_status_t(* psa_drv_mac_transparent_verify_t) (const uint8_t *p_input, size_t input_length, const uint8_t *p_key, size_t key_length, psa_algorithm_t alg, const uint8_t *p_mac, size_t mac_length)

The function prototype for a one-shot operation of a transparent-key MAC Verify operation.

Functions that implement the prototype should be named in the following convention:

```
psa_drv_mac_transparent_<ALGO>_<MAC_VARIANT>_verify
```

Where ALGO is the name of the underlying algorithm, and MAC_VARIANT is the specific variant of a MAC operation (such as HMAC or CMAC)

Parameters

in	p_input	A buffer containing the data to be MACed
in	input_length	The length in bytes of the p_input data
in	p_key	A buffer containing the key material to be used for the MAC operation
in	key_length	The length in bytes of the p_key data
in	alg	The algorithm to be performed
in	p_mac	The MAC data to be compared
in	mac_length	The length in bytes of the p_mac buffer

Return values

PSA_SUCCESS	The operation completed successfully and the comparison matched

4.3 Opaque Symmetric Ciphers

4.3.1 Detailed Description

Encryption and Decryption using opaque keys in block modes other than ECB must be done in multiple parts, using the following flow:

- `psa_drv_cipher_opaque_setup_t`
- `psa_drv_cipher_opaque_set_iv_t` (optional depending upon block mode)
- `psa_drv_cipher_opaque_update_t`
- ...
- `psa_drv_cipher_opaque_finish_t`

If a previously started Opaque Cipher operation needs to be terminated, it should be done so by the `psa_drv_cipher_opaque_abort_t`. Failure to do so may result in allocated resources not being freed or in other undefined behavior.

In situations where a PSA Cryptographic API implementation is using a block mode not-supported by the underlying hardware or driver, it can construct the block mode itself, while calling the `psa_drv_↩ cipher_opaque_ecb_t` function pointer for the cipher operations.

4.3.2 Types

4.3.2.1 typedef psa_status_t(* psa_drv_cipher_opaque_setup_t) (void *p_context, psa_key_slot_t key_slot, psa_algorithm_t algorithm, psa_encrypt_or_decrypt_t direction)

A function pointer that provides the cipher setup function for opaque-key operations.

Parameters

`in,out`	p_context	A structure that will contain the hardware-specific cipher context.
`in`	key_slot	The slot of the key to be used for the operation
`in`	algorithm	The algorithm to be used in the cipher operation
`in`	direction	Indicates whether the operation is an encrypt or decrypt

Return values

PSA_SUCCESS	
PSA_ERROR_NOT_SUPPORTED	

4.3.2.2 typedef psa_status_t(* psa_drv_cipher_opaque_set_iv_t) (void *p_context, const uint8_t *p_iv, size_t iv_length)

A function pointer that sets the initialization vector (if necessary) for an opaque cipher operation.

Rationale: The psa_cipher_* function in the PSA Cryptographic API has two IV functions: one to set the IV, and one to generate it internally. The generate function is not necessary for the drivers to implement as the PSA Crypto implementation can do the generation using its RNG features.

Parameters

in,out	p_context	A structure that contains the previously set up hardware-specific cipher context
in	p_iv	A buffer containing the initialization vector
in	iv_length	The size (in bytes) of the p_iv buffer

Return values

PSA_SUCCESS	

4.3.2.3 typedef psa_status_t(* psa_drv_cipher_opaque_update_t) (void *p_context, const uint8_t *p_input, size_t input_size, uint8_t *p_output, size_t output_size, size_t *p_output_length)

A function that continues a previously started opaque-key cipher operation.

Parameters

in,out	p_context	A hardware-specific structure for the previously started cipher operation
in	p_input	A buffer containing the data to be encrypted/decrypted
in	input_size	The size in bytes of the buffer pointed to by p_input
out	p_output	The caller-allocated buffer where the output will be placed
in	output_size	The allocated size in bytes of the p_output buffer
out	p_output_length	After completion, will contain the number of bytes placed in the p_output buffer

Return values

PSA_SUCCESS	

4.3.2.4 typedef psa_status_t(* psa_drv_cipher_opaque_finish_t) (void *p_context, uint8_t *p_output, size_t output_size, size_t *p_output_length)

A function that completes a previously started opaque-key cipher operation.

Parameters

in,out	p_context	A hardware-specific structure for the previously started cipher operation
out	p_output	The caller-allocated buffer where the output will be placed
in	output_size	The allocated size in bytes of the p_output buffer
out	p_output_length	After completion, will contain the number of bytes placed in the p_output buffer

Return values

PSA_SUCCESS	

4.3.2.5 typedef psa_status_t(* psa_drv_cipher_opaque_abort_t) (void *p_context)

A function that aborts a previously started opaque-key cipher operation.

Parameters

`in,out`	p_context	A hardware-specific structure for the previously started cipher operation

4.3.2.6 typedef psa_status_t(* psa_drv_cipher_opaque_ecb_t) (psa_key_slot_t key_slot, psa_algorithm_t algorithm, psa_encrypt_or_decrypt_t direction, const uint8_t *p_input, size_t input_size, uint8_t *p_output, size_t output_size)

A function that performs the ECB block mode for opaque-key cipher operations.

Note: this function should only be used with implementations that do not provide a needed higher-level operation.

Parameters

`in`	key_slot	The slot of the key to be used for the operation
`in`	algorithm	The algorithm to be used in the cipher operation
`in`	direction	Indicates whether the operation is an encrypt or decrypt
`in`	p_input	A buffer containing the data to be encrypted/decrypted
`in`	input_size	The size in bytes of the buffer pointed to by `p_input`
`out`	p_output	The caller-allocated buffer where the output will be placed
`in`	output_size	The allocated size in bytes of the `p_output` buffer

Return values

PSA_SUCCESS	
PSA_ERROR_NOT_SUPPORTED	

4.4 Transparent Block Cipher

4.4.1 Detailed Description

Encryption and Decryption using transparent keys in block modes other than ECB must be done in multiple parts, using the following flow:

- `psa_drv_cipher_transparent_setup_t`
- `psa_drv_cipher_transparent_set_iv_t` (optional depending upon block mode)
- `psa_drv_cipher_transparent_update_t`
- ...
- `psa_drv_cipher_transparent_finish_t`

If a previously started Transparent Cipher operation needs to be terminated, it should be done so by the `psa_drv_cipher_transparent_abort_t`. Failure to do so may result in allocated resources not being freed or in other undefined behavior.

4.4.2 Types

4.4.2.1 typedef struct psa_drv_cipher_transparent_context_s psa_drv_cipher_transparent_context_t

The hardware-specific transparent-key Cipher context structure.

The contents of this structure are implementation dependent and are therefore not described here.

4.4.2.2 typedef psa_status_t(* psa_drv_cipher_transparent_setup_t) (psa_drv_cipher_transparent_context_t *p_context, psa_encrypt_or_decrypt_t direction, const uint8_t *p_key_data, size_t key_data_size)

The function prototype for the setup operation of transparent-key block cipher operations. Functions that implement the prototype should be named in the following conventions:

```
psa_drv_cipher_transparent_setup_<CIPHER_NAME>_<MODE>
```

Where.

- `CIPHER_NAME` is the name of the underlying block cipher (i.e. AES or DES)
- `MODE` is the block mode of the cipher operation (i.e. CBC or CTR) or for stream ciphers:

  ```
  psa_drv_cipher_transparent_setup_<CIPHER_NAME>
  ```

  Where `CIPHER_NAME` is the name of a stream cipher (i.e. RC4)

Parameters

in, out	p_context	A structure that will contain the hardware-specific cipher context
in	direction	Indicates if the operation is an encrypt or a decrypt
in	p_key_data	A buffer containing the cleartext key material to be used in the operation
in	key_data_size	The size in bytes of the key material

Return values

PSA_SUCCESS	

4.4.2.3 typedef psa_status_t(* psa_drv_cipher_transparent_set_iv_t) (psa_drv_cipher_transparent_context_t *p_context, const uint8_t *p_iv, size_t iv_length)

The function prototype for the set initialization vector operation of transparent-key block cipher operations Functions that implement the prototype should be named in the following convention:

```
psa_drv_cipher_transparent_set_iv_<CIPHER_NAME>_<MODE>
```

Where.

- CIPHER_NAME is the name of the underlying block cipher (i.e. AES or DES)
- MODE is the block mode of the cipher operation (i.e. CBC or CTR)

Parameters

in, out	p_context	A structure that contains the previously setup hardware-specific cipher context
in	p_iv	A buffer containing the initialization vecotr
in	iv_length	The size in bytes of the contents of p_iv

Return values

PSA_SUCCESS	

4.4.2.4 typedef psa_status_t(* psa_drv_cipher_transparent_update_t) (psa_drv_cipher_transparent_context_t *p_context, const uint8_t *p_input, size_t input_size, uint8_t *p_output, size_t output_size, size_t *p_output_length)

The function prototype for the update operation of transparent-key block cipher operations.

Functions that implement the prototype should be named in the following convention:

```
psa_drv_cipher_transparent_update_<CIPHER_NAME>_<MODE>
```

Where

- CIPHER_NAME is the name of the underlying block cipher (i.e. AES or DES)
- MODE is the block mode of the cipher operation (i.e. CBC or CTR)

Parameters

in,out	p_context	A hardware-specific structure for the previously started cipher operation
in	p_input	A buffer containing the data to be encrypted or decrypted
in	input_size	The size in bytes of the p_input buffer
out	p_output	A caller-allocated buffer where the generated output will be placed
in	output_size	The size in bytes of the p_output buffer
out	p_output_length	After completion, will contain the number of bytes placed in the p_output buffer

Return values

PSA_SUCCESS	

4.4.2.5 typedef psa_status_t(* psa_drv_cipher_transparent_finish_t) (psa_drv_cipher_↩ transparent_context_t *p_context, uint8_t *p_output, size_t output_size, size_t *p_output_length)

The function prototype for the finish operation of transparent-key block cipher operations.

Functions that implement the prototype should be named in the following convention:

```
psa_drv_cipher_transparent_finish_<CIPHER_NAME>_<MODE>
```

Where

- CIPHER_NAME is the name of the underlying block cipher (i.e. AES or DES)
- MODE is the block mode of the cipher operation (i.e. CBC or CTR)

Parameters

in,out	p_context	A hardware-specific structure for the previously started cipher operation
out	p_output	A caller-allocated buffer where the generated output will be placed
in	output_size	The size in bytes of the p_output buffer
out	p_output_length	After completion, will contain the number of bytes placed in the p_output buffer

Return values

PSA_SUCCESS	

4.4.2.6 typedef psa_status_t(* psa_drv_cipher_transparent_abort_t) (psa_drv_cipher_transparent_context_t *p_context)

The function prototype for the abort operation of transparent-key block cipher operations.

Functions that implement the following prototype should be named in the following convention:

```
psa_drv_cipher_transparent_abort_<CIPHER_NAME>_<MODE>
```

Where

- `CIPHER_NAME` is the name of the underlying block cipher (i.e. AES or DES)
- `MODE` is the block mode of the cipher operation (i.e. CBC or CTR)

Parameters

`in,out`	p_context	A hardware-specific structure for the previously started cipher operation

Return values

PSA_SUCCESS	

4.5 Message Digests

4.5.1 Detailed Description

Generation and authentication of Message Digests (aka hashes) must be done in parts using the following sequence:

- `psa_drv_hash_setup_t`
- `psa_drv_hash_update_t`
- ...
- `psa_drv_hash_finish_t`

If a previously started Message Digest operation needs to be terminated before the `psa_drv_hash_finish_t` operation is complete, it should be aborted by the `psa_drv_hash_abort_t`. Failure to do so may result in allocated resources not being freed or in other undefined behavior.

4.5.2 Types

4.5.2.1 typedef struct psa_drv_hash_context_s psa_drv_hash_context_t

The hardware-specific hash context structure.

The contents of this structure are implementation dependent and are therefore not described here

4.5.2.2 typedef psa_status_t(* psa_drv_hash_setup_t) (psa_drv_hash_context_t *p_context)

The function prototype for the start operation of a hash (message digest) operation.

Functions that implement the prototype should be named in the following convention:

```
psa_drv_hash_<ALGO>_setup
```

Where `ALGO` is the name of the underlying hash function

Parameters

`in,out`	p_context	A structure that will contain the hardware-specific hash context

Return values

PSA_SUCCESS	Success.

4.5.2.3 typedef psa_status_t(* psa_drv_hash_update_t) (psa_drv_hash_context_t *p_context, const uint8_t *p_input, size_t input_length)

The function prototype for the update operation of a hash (message digest) operation.

Functions that implement the prototype should be named in the following convention:

```
psa_drv_hash_<ALGO>_update
```

Where `ALGO` is the name of the underlying algorithm

Parameters

`in,out`	p_context	A hardware-specific structure for the previously-established hash operation to be continued
`in`	p_input	A buffer containing the message to be appended to the hash operation
`in`	input_length	The size in bytes of the input message buffer

4.5.2.4 typedef psa_status_t(* psa_drv_hash_finish_t) (psa_drv_hash_context_t *p_context, uint8_t *p_output, size_t output_size, size_t *p_output_length)

The prototype for the finish operation of a hash (message digest) operation.

Functions that implement the prototype should be named in the following convention:

```
psa_drv_hash_<ALGO>_finish
```

Where `ALGO` is the name of the underlying algorithm

Parameters

`in,out`	p_context	A hardware-specific structure for the previously started hash operation to be fiinished
`out`	p_output	A buffer where the generated digest will be placed
`in`	output_size	The size in bytes of the buffer that has been allocated for the `p_output` buffer
`out`	p_output_length	The number of bytes placed in `p_output` after success

Return values

PSA_SUCCESS	Success.

4.5.2.5 typedef void(* psa_drv_hash_abort_t) (psa_drv_hash_context_t *p_context)

The function prototype for the abort operation of a hash (message digest) operation.

Functions that implement the prototype should be named in the following convention:

`psa_drv_hash_<ALGO>_abort`

Where `ALGO` is the name of the underlying algorithm

Parameters

`in,out`	p_context	A hardware-specific structure for the previously started hash operation to be aborted

4.6 Opaque Asymmetric Cryptography

4.6.1 Detailed Description

Since the amount of data that can (or should) be encrypted or signed using asymmetric keys is limited by the key size, asymmetric key operations using opaque keys must be done in single function calls.

4.6.2 Types

4.6.2.1 typedef psa_status_t(* psa_drv_asymmetric_opaque_sign_t) (psa_key_slot_t key_slot, psa_algorithm_t alg, const uint8_t *p_hash, size_t hash_length, uint8_t *p_signature, size_t signature_size, size_t *p_signature_length)

A function that signs a hash or short message with a private key.

Parameters

in	key_slot	Key slot of an asymmetric key pair
in	alg	A signature algorithm that is compatible with the type of key
in	p_hash	The hash to sign
in	hash_length	Size of the p_hash buffer in bytes
out	p_signature	Buffer where the signature is to be written
in	signature_size	Size of the p_signature buffer in bytes
out	p_signature_length	On success, the number of bytes that make up the returned signature value

Return values

PSA_SUCCESS	

4.6.2.2 typedef psa_status_t(* psa_drv_asymmetric_opaque_verify_t) (psa_key_slot_t key_slot, psa_algorithm_t alg, const uint8_t *p_hash, size_t hash_length, const uint8_t *p_signature, size_t signature_length)

A function that verifies the signature a hash or short message using an asymmetric public key.

Parameters

in	key_slot	Key slot of a public key or an asymmetric key pair
in	alg	A signature algorithm that is compatible with the type of key
in	p_hash	The hash whose signature is to be verified
in	hash_length	Size of the p_hash buffer in bytes
in	p_signature	Buffer containing the signature to verify
in	signature_length	Size of the p_signature buffer in bytes

Return values

PSA_SUCCESS	The signature is valid.

4.6.2.3 typedef psa_status_t(* psa_drv_asymmetric_opaque_encrypt_t) (psa_key_slot_t key_slot, psa_algorithm_t alg, const uint8_t *p_input, size_t input_length, const uint8_t *p_salt, size_t salt_length, uint8_t *p_output, size_t output_size, size_t *p_output_length)

A function that encrypts a short message with an asymmetric public key.

Parameters

in	key_slot	Key slot of a public key or an asymmetric key pair
in	alg	An asymmetric encryption algorithm that is compatible with the type of `key`
in	p_input	The message to encrypt
in	input_length	Size of the `p_input` buffer in bytes
in	p_salt	A salt or label, if supported by the encryption algorithm If the algorithm does not support a salt, pass `NULL`. If the algorithm supports an optional salt and you do not want to pass a salt, pass `NULL`. For #PSA_ALG_RSA_PKCS1V15_CRYPT, no salt is supported.
in	salt_length	Size of the `p_salt` buffer in bytes If `p_salt` is `NULL`, pass 0.
out	p_output	Buffer where the encrypted message is to be written
in	output_size	Size of the `p_output` buffer in bytes
out	p_output_length	On success, the number of bytes that make up the returned output

Return values

PSA_SUCCESS	

4.6.2.4 typedef psa_status_t(* psa_drv_asymmetric_opaque_decrypt_t) (psa_key_slot_t key_slot, psa_algorithm_t alg, const uint8_t *p_input, size_t input_length, const uint8_t *p_salt, size_t salt_length, uint8_t *p_output, size_t output_size, size_t *p_output_length)

Decrypt a short message with an asymmetric private key.

Parameters

in	key_slot	Key slot of an asymmetric key pair
in	alg	An asymmetric encryption algorithm that is compatible with the type of `key`
in	p_input	The message to decrypt
in	input_length	Size of the `p_input` buffer in bytes
in	p_salt	A salt or label, if supported by the encryption algorithm If the algorithm does not support a salt, pass `NULL`. If the algorithm supports an optional salt and you do not want to pass a salt, pass `NULL`. For #PSA_ALG_RSA_PKCS1V15_CRYPT, no salt is supported.

Parameters

in	salt_length	Size of the p_salt buffer in bytes If p_salt is NULL, pass 0.
out	p_output	Buffer where the decrypted message is to be written
in	output_size	Size of the p_output buffer in bytes
out	p_output_length	On success, the number of bytes that make up the returned output

Return values

PSA_SUCCESS	

4.7 Transparent Asymmetric Cryptography

4.7.1 Detailed Description

Since the amount of data that can (or should) be encrypted or signed using asymmetric keys is limited by the key size, asymmetric key operations using transparent keys must be done in single function calls.

4.7.2 Types

4.7.2.1 typedef psa_status_t(* psa_drv_asymmetric_transparent_sign_t) (const uint8_t *p_key, size_t key_size, psa_algorithm_t alg, const uint8_t *p_hash, size_t hash_length, uint8_t *p_signature, size_t signature_size, size_t *p_signature_length)

A function that signs a hash or short message with a transparent asymmetric private key.

Functions that implement the prototype should be named in the following convention:

```
psa_drv_asymmetric_<ALGO>_sign
```

Where `ALGO` is the name of the signing algorithm

Parameters

`in`	p_key	A buffer containing the private key material
`in`	key_size	The size in bytes of the `p_key` data
`in`	alg	A signature algorithm that is compatible with the type of `p_key`
`in`	p_hash	The hash or message to sign
`in`	hash_length	Size of the `p_hash` buffer in bytes
`out`	p_signature	Buffer where the signature is to be written
`in`	signature_size	Size of the `p_signature` buffer in bytes
`out`	p_signature_length	On success, the number of bytes that make up the returned signature value

Return values

PSA_SUCCESS	

4.7.2.2 typedef psa_status_t(* psa_drv_asymmetric_transparent_verify_t) (const uint8_t *p_key, size_t key_size, psa_algorithm_t alg, const uint8_t *p_hash, size_t hash_length, const uint8_t *p_signature, size_t signature_length)

A function that verifies the signature a hash or short message using a transparent asymmetric public key.

Functions that implement the prototype should be named in the following convention:

```
psa_drv_asymmetric_<ALGO>_verify
```

Where `ALGO` is the name of the signing algorithm

Parameters

in	p_key	A buffer containing the public key material
in	key_size	The size in bytes of the p_key data
in	alg	A signature algorithm that is compatible with the type of key
in	p_hash	The hash or message whose signature is to be verified
in	hash_length	Size of the p_hash buffer in bytes
in	p_signature	Buffer containing the signature to verify
in	signature_length	Size of the p_signature buffer in bytes

Return values

PSA_SUCCESS	The signature is valid.

4.7.2.3 typedef psa_status_t(* psa_drv_asymmetric_transparent_encrypt_t) (const uint8_t *p_key, size_t key_size, psa_algorithm_t alg, const uint8_t *p_input, size_t input_length, const uint8_t *p_salt, size_t salt_length, uint8_t *p_output, size_t output_size, size_t *p_output_length)

A function that encrypts a short message with a transparent asymmetric public key.

Functions that implement the prototype should be named in the following convention:

```
psa_drv_asymmetric_<ALGO>_encrypt
```

Where ALGO is the name of the encryption algorithm

Parameters

in	p_key	A buffer containing the public key material
in	key_size	The size in bytes of the p_key data
in	alg	An asymmetric encryption algorithm that is compatible with the type of key
in	p_input	The message to encrypt
in	input_length	Size of the p_input buffer in bytes
in	p_salt	A salt or label, if supported by the encryption algorithm If the algorithm does not support a salt, pass NULL If the algorithm supports an optional salt and you do not want to pass a salt, pass NULL. For #PSA_ALG_RSA_PKCS1V15_CRYPT, no salt is supported.
in	salt_length	Size of the p_salt buffer in bytes If p_salt is NULL, pass 0.
out	p_output	Buffer where the encrypted message is to be written
in	output_size	Size of the p_output buffer in bytes
out	p_output_length	On success, the number of bytes that make up the returned output

Return values

PSA_SUCCESS	

4.7.2.4 typedef psa_status_t(* psa_drv_asymmetric_transparent_decrypt_t) (const uint8_t *p_key, size_t key_size, psa_algorithm_t alg, const uint8_t *p_input, size_t input_length, const uint8_t *p_salt, size_t salt_length, uint8_t *p_output, size_t output_size, size_t *p_output_length)

Decrypt a short message with a transparent asymmetric private key.

Functions that implement the prototype should be named in the following convention:

```
psa_drv_asymmetric_<ALGO>_decrypt
```

Where `ALGO` is the name of the encryption algorithm

Parameters

`in`	p_key	A buffer containing the private key material
`in`	key_size	The size in bytes of the `p_key` data
`in`	alg	An asymmetric encryption algorithm that is compatible with the type of `key`
`in`	p_input	The message to decrypt
`in`	input_length	Size of the `p_input` buffer in bytes
`in`	p_salt	A salt or label, if supported by the encryption algorithm If the algorithm does not support a salt, pass `NULL`. If the algorithm supports an optional salt and you do not want to pass a salt, pass `NULL`. For #PSA_ALG_RSA_PKCS1V15_CRYPT, no salt is supported
`in`	salt_length	Size of the `p_salt` buffer in bytes If `p_salt` is `NULL`, pass 0
`out`	p_output	Buffer where the decrypted message is to be written
`in`	output_size	Size of the `p_output` buffer in bytes
`out`	p_output_length	On success, the number of bytes that make up the returned output

Return values

PSA_SUCCESS	

4.8 AEAD Opaque

4.8.1 Detailed Description

Authenticated Encryption with Additional Data (AEAD) operations with opaque keys must be done in one function call. While this creates a burden for implementers as there must be sufficient space in memory for the entire message, it prevents decrypted data from being made available before the authentication operation is complete and the data is known to be authentic.

4.8.2 Types

4.8.2.1 typedef psa_status_t(* psa_drv_aead_opaque_encrypt_t) (psa_key_slot_t key_slot, psa_algorithm_t algorithm, const uint8_t *p_nonce, size_t nonce_length, const uint8_t *p_additional_data, size_t additional_data_length, const uint8_t *p_plaintext, size_t plaintext_length, uint8_t *p_ciphertext, size_t ciphertext_size, size_t *p_ciphertext_length)

Process an authenticated encryption operation using an opaque key.

Parameters

in	key_slot	Slot containing the key to use.
in	algorithm	The AEAD algorithm to compute (PSA_ALG_XXX value such that #PSA_ALG_IS_AEAD(alg) is true)
in	p_nonce	Nonce or IV to use
in	nonce_length	Size of the p_nonce buffer in bytes
in	p_additional_data	Additional data that will be authenticated but not encrypted
in	additional_data_length	Size of p_additional_data in bytes
in	p_plaintext	Data that will be authenticated and encrypted
in	plaintext_length	Size of p_plaintext in bytes
out	p_ciphertext	Output buffer for the authenticated and encrypted data. The additional data is not part of this output. For algorithms where the encrypted data and the authentication tag are defined as separate outputs, the authentication tag is appended to the encrypted data.
in	ciphertext_size	Size of the p_ciphertext buffer in bytes
out	p_ciphertext_length	On success, the size of the output in the p_ciphertext buffer

Return values

#PSA_SUCCESS	Success.

4.8.2.2 typedef psa_status_t(* psa_drv_aead_opaque_decrypt_t) (psa_key_slot_t key_slot, psa_algorithm_t algorithm, const uint8_t *p_nonce, size_t nonce_length, const uint8_t *p_additional_data, size_t additional_data_length, const uint8_t *p_ciphertext, size_t ciphertext_length, uint8_t *p_plaintext, size_t plaintext_size, size_t *p_plaintext_length)

Process an authenticated decryption operation using an opaque key

Parameters

in	key_slot	Slot containing the key to use
in	algorithm	The AEAD algorithm to compute (PSA_ALG_XXX value such that #PSA_ALG_IS_AEAD(alg) is true)
in	p_nonce	Nonce or IV to use
in	nonce_length	Size of the p_nonce buffer in bytes
in	p_additional_data	Additional data that has been authenticated but not encrypted
in	additional_data_length	Size of p_additional_data in bytes
in	p_ciphertext	Data that has been authenticated and encrypted. For algorithms where the encrypted data and the authentication tag are defined as separate inputs, the buffer must contain the encrypted data followed by the authentication tag.
in	ciphertext_length	Size of p_ciphertext in bytes
out	p_plaintext	Output buffer for the decrypted data
in	plaintext_size	Size of the p_plaintext buffer in bytes
out	p_plaintext_length	On success, the size of the output in the p_plaintext buffer

Return values

#PSA_SUCCESS	Success.

4.9 AEAD Transparent

4.9.1 Detailed Description

Authenticated Encryption with Additional Data (AEAD) operations with transparent keys must be done in one function call. While this creates a burden for implementers as there must be sufficient space in memory for the entire message, it prevents decrypted data from being made available before the authentication operation is complete and the data is known to be authentic.

4.9.2 Types

4.9.2.1 typedef psa_status_t(* psa_drv_aead_transparent_encrypt_t) (const uint8_t *p_key, size_t key_length, psa_algorithm_t alg, const uint8_t *nonce, size_t nonce_length, const uint8_t *additional_data, size_t additional_data_length, const uint8_t *plaintext, size_t plaintext_length, uint8_t *ciphertext, size_t ciphertext_size, size_t *ciphertext_length)

Process an authenticated encryption operation using an opaque key.

Functions that implement the prototype should be named in the following convention:

```
psa_drv_aead_<ALGO>_encrypt
```

Where `ALGO` is the name of the AEAD algorithm

Parameters

in	p_key	A pointer to the key material
in	key_length	The size in bytes of the key material
in	alg	The AEAD algorithm to compute (`PSA_ALG_XXX` value such that #PSA_ALG_IS_AEAD(`alg`) is true)
in	nonce	Nonce or IV to use
in	nonce_length	Size of the `nonce` buffer in bytes
in	additional_data	Additional data that will be MACed but not encrypted.
in	additional_data_length	Size of `additional_data` in bytes
in	plaintext	Data that will be MACed and encrypted.
in	plaintext_length	Size of `plaintext` in bytes
out	ciphertext	Output buffer for the authenticated and encrypted data. The additional data is not part of this output. For algorithms where the encrypted data and the authentication tag are defined as separate outputs, the authentication tag is appended to the encrypted data.
in	ciphertext_size	Size of the `ciphertext` buffer in bytes This must be at least #PSA_AEAD_ENCRYPT_OUTPUT_SIZE(`alg`, `plaintext_length`).
out	ciphertext_length	On success, the size of the output in the `ciphertext` buffer

Return values

#PSA_SUCCESS	

4.9.2.2 typedef psa_status_t(* psa_drv_aead_transparent_decrypt_t) (const uint8_t *p_key, size_t key_length, psa_algorithm_t alg, const uint8_t *nonce, size_t nonce_length, const uint8_t *additional_data, size_t additional_data_length, const uint8_t *ciphertext, size_t ciphertext_length, uint8_t *plaintext, size_t plaintext_size, size_t *plaintext_length)

Process an authenticated decryption operation using an opaque key.

Functions that implement the prototype should be named in the following convention:

```
psa_drv_aead_<ALGO>_decrypt
```

Where `ALGO` is the name of the AEAD algorithm

Parameters

`in`	p_key	A pointer to the key material
`in`	key_length	The size in bytes of the key material
`in`	alg	The AEAD algorithm to compute (`PSA_ALG_XXX` value such that #PSA_ALG_IS_AEAD(`alg`) is true)
`in`	nonce	Nonce or IV to use
`in`	nonce_length	Size of the `nonce` buffer in bytes
`in`	additional_data	Additional data that has been MACed but not encrypted
`in`	additional_data_length	Size of `additional_data` in bytes
`in`	ciphertext	Data that has been MACed and encrypted For algorithms where the encrypted data and the authentication tag are defined as separate inputs, the buffer must contain the encrypted data followed by the authentication tag.
`in`	ciphertext_length	Size of `ciphertext` in bytes
`out`	plaintext	Output buffer for the decrypted data
`in`	plaintext_size	Size of the `plaintext` buffer in bytes This must be at least #PSA_AEAD_DECRYPT_OUTPUT_SIZE(`alg`, `ciphertext_length`).
`out`	plaintext_length	On success, the size of the output in the plaintext buffer

Return values

#PSA_SUCCESS	Success.

4.10 Entropy Generation

4.10.1 Detailed Description

4.10.2 Types

4.10.2.1 typedef psa_status_t(* psa_drv_entropy_init_t) (psa_drv_entropy_context_t *p_context)

Initialize an entropy driver.

Parameters

in,out	p_context	A hardware-specific structure containing any context information for the implementation

Return values

PSA_SUCCESS	

4.10.2.2 typedef psa_status_t(* psa_drv_entropy_get_bits_t) (psa_drv_entropy_context_t *p_context, uint8_t *p_buffer, uint32_t buffer_size, uint32_t *p_received_entropy_bits)

Get a specified number of bits from the entropy source.

It retrives `buffer_size` bytes of data from the entropy source. The entropy source will always fill the provided buffer to its full size, however, most entropy sources have biases, and the actual amount of entropy contained in the buffer will be less than the number of bytes. The driver will return the actual number of bytes of entropy placed in the buffer in `p_received_entropy_bytes`. A PSA Crypto API implementation will likely feed the output of this function into a Digital Random Bit Generator (DRBG), and typically has a minimum amount of entropy that it needs. To accomplish this, the PSA Crypto implementation should be designed to call this function multiple times until it has received the required amount of entropy from the entropy source.

Parameters

in,out	p_context	A hardware-specific structure containing any context information for the implementation
out	p_buffer	A caller-allocated buffer for the retrieved entropy to be placed in
in	buffer_size	The allocated size of `p_buffer`
out	p_received_entropy_bits	The amount of entropy (in bits) actually provided in `p_buffer`

Return values

PSA_SUCCESS	

4.11 Key Management

4.11.1 Detailed Description

Currently, key management is limited to importing keys in the clear, destroying keys, and exporting keys in the clear. Whether a key may be exported is determined by the key policies in place on the key slot.

4.11.2 Types

4.11.2.1 typedef psa_status_t(* psa_drv_opaque_import_key_t) (psa_key_slot_t key_slot, psa_key_type_t type, psa_algorithm_t algorithm, psa_key_usage_t usage, const uint8_t *p_data, size_t data_length)

Import a key in binary format.

This function can support any output from psa_export_key(). Refer to the documentation of psa_↩ export_key() for the format for each key type.

Parameters

in	key_slot	Slot where the key will be stored This must be a valid slot for a key of the chosen type. It must be unoccupied.
in	type	Key type (a PSA_KEY_TYPE_XXX value)
in	algorithm	Key algorithm (a PSA_ALG_XXX value)
in	usage	The allowed uses of the key
in	p_data	Buffer containing the key data
in	data_length	Size of the data buffer in bytes

Return values

#PSA_SUCCESS	Success.

4.11.2.2 typedef psa_status_t(* psa_drv_destroy_key_t) (psa_key_slot_t key)

Destroy a key and restore the slot to its default state.

This function destroys the content of the key slot from both volatile memory and, if applicable, non-volatile storage. Implementations shall make a best effort to ensure that any previous content of the slot is unrecoverable.

This function also erases any metadata such as policies. It returns the specified slot to its default state.

Parameters

in	key_slot	The key slot to erase.

Return values

#PSA_SUCCESS	The slot's content, if any, has been erased.

4.11.2.3 typedef psa_status_t(* psa_drv_export_key_t) (psa_key_slot_t key, uint8_t *p_data, size_t data_size, size_t *p_data_length)

Export a key in binary format.

The output of this function can be passed to psa_import_key() to create an equivalent object.

If a key is created with `psa_import_key()` and then exported with this function, it is not guaranteed that the resulting data is identical: the implementation may choose a different representation of the same key if the format permits it.

For standard key types, the output format is as follows:

- For symmetric keys (including MAC keys), the format is the raw bytes of the key.
- For DES, the key data consists of 8 bytes. The parity bits must be correct.
- For Triple-DES, the format is the concatenation of the two or three DES keys.
- For RSA key pairs (#PSA_KEY_TYPE_RSA_KEYPAIR), the format is the non-encrypted DER representation defined by PKCS#1 (RFC 8017) as RSAPrivateKey.
- For RSA public keys (#PSA_KEY_TYPE_RSA_PUBLIC_KEY), the format is the DER representation defined by RFC 5280 as SubjectPublicKeyInfo.

Parameters

`in`	key	Slot whose content is to be exported. This must be an occupied key slot.
`out`	p_data	Buffer where the key data is to be written.
`in`	data_size	Size of the `p_data` buffer in bytes.
`out`	p_data_length	On success, the number of bytes that make up the key data.

Return values

#PSA_SUCCESS	
#PSA_ERROR_EMPTY_SLOT	
#PSA_ERROR_NOT_PERMITTED	
#PSA_ERROR_NOT_SUPPORTED	
#PSA_ERROR_COMMUNICATION_FAILURE	
#PSA_ERROR_HARDWARE_FAILURE	
#PSA_ERROR_TAMPERING_DETECTED	

4.11.2.4 typedef psa_status_t(* psa_drv_export_public_key_t) (psa_key_slot_t key, uint8_t *p_data, size_t data_size, size_t *p_data_length)

Export a public key or the public part of a key pair in binary format.

The output of this function can be passed to psa_import_key() to create an object that is equivalent to the public key.

For standard key types, the output format is as follows:

- For RSA keys (#PSA_KEY_TYPE_RSA_KEYPAIR or #PSA_KEY_TYPE_RSA_PUBLI↩ C_KEY), the format is the DER representation of the public key defined by RFC 5280 as Subject↩ PublicKeyInfo.

Parameters

`in`	key_slot	Slot whose content is to be exported. This must be an occupied key slot.
`out`	p_data	Buffer where the key data is to be written.
`in`	data_size	Size of the `data` buffer in bytes.
`out`	p_data_length	On success, the number of bytes that make up the key data.

Return values

#PSA_SUCCESS	

4.12 Key Derivation and Agreement

4.12.1 Detailed Description

Key derivation is the process of generating new key material using an existing key and additional parameters, iterating through a basic cryptographic function, such as a hash. Key agreement is a part of cryptographic protocols that allows two parties to agree on the same key value, but starting from different original key material. The flows are similar, and the PSA Crypto Driver Model uses the same functions for both of the flows.

There are two different final functions for the flows, `psa_drv_key_derivation_derive` and `psa_drv_key_derivation_export`. `psa_drv_key_derivation_derive` is used when the key material should be placed in a slot on the hardware and not exposed to the caller. `psa_drv_key_derivation_export` is used when the key material should be returned to the PSA Cryptographic API implementation.

Different key derivation algorithms require a different number of inputs. Instead of having an API that takes as input variable length arrays, which can be problemmatic to manage on embedded platforms, the inputs are passed to the driver via a function, `psa_drv_key_derivation_collateral`, that is called multiple times with different `collateral_id`s. Thus, for a key derivation algorithm that required 3 paramter inputs, the flow would look something like:

```
psa_drv_key_derivation_setup(kdf_algorithm, source_key, dest_key_size_bytes);
psa_drv_key_derivation_collateral(kdf_algorithm_collateral_id_0,
                                  p_collateral_0,
                                  collateral_0_size);
psa_drv_key_derivation_collateral(kdf_algorithm_collateral_id_1,
                                  p_collateral_1,
                                  collateral_1_size);
psa_drv_key_derivation_collateral(kdf_algorithm_collateral_id_2,
                                  p_collateral_2,
                                  collateral_2_size);
psa_drv_key_derivation_derive();
```

key agreement example:

```
psa_drv_key_derivation_setup(alg, source_key, dest_key_size_bytes);
psa_drv_key_derivation_collateral(DHE_PUBKEY, p_pubkey, pubkey_size);
psa_drv_key_derivation_export(p_session_key,
                              session_key_size,
                              &session_key_length);
```

4.12.2 Types

4.12.2.1 typedef struct psa_drv_key_derivation_context_s psa_drv_key_derivation_context_t

The hardware-specific key derivation context structure.

The contents of this structure are implementation dependent and are therefore not described here

4.12.2.2 typedef psa_status_t(* psa_drv_key_derivation_setup_t) (psa_drv_key_derivation_context_t *p_context, psa_algorithm_t kdf_alg, psa_key_slot_t source_key)

Set up a key derivation operation by specifying the algorithm and the source key sot.

Parameters

in,out	p_context	A hardware-specific structure containing any context information for the implementation
in	kdf_alg	The algorithm to be used for the key derivation
in	souce_key	The key to be used as the source material for the key derivation

Return values

PSA_SUCCESS	

4.12.2.3 typedef psa_status_t(* psa_drv_key_derivation_collateral_t) (psa_drv_key_derivation↩ _context_t *p_context, uint32_t collateral_id, const uint8_t *p_collateral, size_t collateral_size)

Provide collateral (parameters) needed for a key derivation or key agreement operation.

Since many key derivation algorithms require multiple parameters, it is expeced that this function may be called multiple times for the same operation, each with a different algorithm-specific `collateral_id`

Parameters

in,out	p_context	A hardware-specific structure containing any context information for the implementation
in	collateral_id	An ID for the collateral being provided
in	p_collateral	A buffer containing the collateral data
in	collateral_size	The size in bytes of the collateral

Return values

PSA_SUCCESS	

4.12.2.4 typedef psa_status_t(* psa_drv_key_derivation_derive_t) (psa_drv_key_derivation_↩ context_t *p_context, psa_key_slot_t dest_key)

Perform the final key derivation step and place the generated key material in a slot.

Parameters

in,out	p_context	A hardware-specific structure containing any context information for the implementation
in	dest_key	The slot where the generated key material should be placed

Return values

PSA_SUCCESS	

4.12.2.5 typedef psa_status_t(* psa_drv_key_derivation_export_t) (uint8_t *p_output, size_t output_size, size_t *p_output_length)

Perform the final step of a key agreement and place the generated key material in a buffer.

Parameters

`out`	p_output	Buffer in which to place the generated key material
`in`	output_size	The size in bytes of `p_output`
`out`	p_output_length	Upon success, contains the number of bytes of key material placed in `p_output`

Return values

PSA_SUCCESS	

Chapter 5

Data Structure Documentation

5.1 psa_drv_aead_opaque_t Struct Reference

A struct containing all of the function pointers needed to implement Authenticated Encryption with Additional Data operations using opaque keys.

`#include <crypto_driver.h>`

Data Fields

- psa_drv_aead_opaque_encrypt_t * p_encrypt
- psa_drv_aead_opaque_decrypt_t * p_decrypt

5.1.1 Detailed Description

A struct containing all of the function pointers needed to implement Authenticated Encryption with Additional Data operations using opaque keys.

PSA Crypto API implementations should populate instances of the table as appropriate upon startup.

If one of the functions is not implemented, it should be set to NULL.

5.1.2 Field Documentation

5.1.2.1 psa_drv_aead_opaque_encrypt_t* psa_drv_aead_opaque_t::p_encrypt

Function that performs the AEAD encrypt operation

5.1.2.2 psa_drv_aead_opaque_decrypt_t* psa_drv_aead_opaque_t::p_decrypt

Function that performs the AEAD decrypt operation

The documentation for this struct was generated from the following file:

- /mnt/c/Users/dermil01/psa-crypto/mbedtls/include/psa/crypto_driver.h

5.2 psa_drv_asymmetric_opaque_t Struct Reference

A struct containing all of the function pointers needed to implement asymmetric cryptographic operations using opaque keys.

`#include <crypto_driver.h>`

Data Fields

- psa_drv_asymmetric_opaque_sign_t * p_sign
- psa_drv_asymmetric_opaque_verify_t * p_verify
- psa_drv_asymmetric_opaque_encrypt_t * p_encrypt
- psa_drv_asymmetric_opaque_decrypt_t * p_decrypt

5.2.1 Detailed Description

A struct containing all of the function pointers needed to implement asymmetric cryptographic operations using opaque keys.

PSA Crypto API implementations should populate instances of the table as appropriate upon startup.

If one of the functions is not implemented, it should be set to NULL.

5.2.2 Field Documentation

5.2.2.1 psa_drv_asymmetric_opaque_sign_t* psa_drv_asymmetric_opaque_t::p_sign

Function that performs the asymmetric sign operation

5.2.2.2 psa_drv_asymmetric_opaque_verify_t* psa_drv_asymmetric_opaque_t::p_verify

Function that performs the asymmetric verify operation

5.2.2.3 psa_drv_asymmetric_opaque_encrypt_t* psa_drv_asymmetric_opaque_t::p_encrypt

Function that performs the asymmetric encrypt operation

5.2.2.4 psa_drv_asymmetric_opaque_decrypt_t* psa_drv_asymmetric_opaque_t::p_decrypt

Function that performs the asymmetric decrypt operation

The documentation for this struct was generated from the following file:

- /mnt/c/Users/dermil01/psa-crypto/mbedtls/include/psa/crypto_driver.h

5.3 psa_drv_cipher_opaque_t Struct Reference

A struct containing all of the function pointers needed to implement cipher operations using opaque keys.

`#include <crypto_driver.h>`

Data Fields

- size_t size
- psa_drv_cipher_opaque_setup_t * p_setup
- psa_drv_cipher_opaque_set_iv_t * p_set_iv
- psa_drv_cipher_opaque_update_t * p_update
- psa_drv_cipher_opaque_finish_t * p_finish
- psa_drv_cipher_opaque_abort_t * p_abort
- psa_drv_cipher_opaque_ecb_t * p_ecb

5.3.1 Detailed Description

A struct containing all of the function pointers needed to implement cipher operations using opaque keys.

PSA Crypto API implementations should populate instances of the table as appropriate upon startup.

If one of the functions is not implemented (such as `psa_drv_cipher_opaque_ecb_t`), it should be set to NULL.

5.3.2 Field Documentation

5.3.2.1 size_t psa_drv_cipher_opaque_t::size

The size in bytes of the hardware-specific Opaque Cipher context structure

5.3.2.2 psa_drv_cipher_opaque_setup_t* psa_drv_cipher_opaque_t::p_setup

Function that performs the setup operation

5.3.2.3 psa_drv_cipher_opaque_set_iv_t* psa_drv_cipher_opaque_t::p_set_iv

Function that sets the IV (if necessary)

5.3.2.4 psa_drv_cipher_opaque_update_t* psa_drv_cipher_opaque_t::p_update

Function that performs the update operation

5.3.2.5 psa_drv_cipher_opaque_finish_t* psa_drv_cipher_opaque_t::p_finish

Function that completes the operation

5.3.2.6 psa_drv_cipher_opaque_abort_t* psa_drv_cipher_opaque_t::p_abort

Function that aborts the operation

5.3.2.7 psa_drv_cipher_opaque_ecb_t* psa_drv_cipher_opaque_t::p_ecb

Function that performs ECB mode for the cipher (Danger: ECB mode should not be used directly by clients of the PSA Crypto Client API)

The documentation for this struct was generated from the following file:

- /mnt/c/Users/dermil01/psa-crypto/mbedtls/include/psa/crypto_driver.h

5.4 psa_drv_entropy_t Struct Reference

A struct containing all of the function pointers needed to interface to an entropy source.

```
#include <crypto_driver.h>
```

Data Fields

- psa_drv_entropy_init_t * p_init
- psa_drv_entropy_get_bits_t * p_get_bits

5.4.1 Detailed Description

A struct containing all of the function pointers needed to interface to an entropy source.

PSA Crypto API implementations should populate instances of the table as appropriate upon startup.

If one of the functions is not implemented, it should be set to NULL.

5.4.2 Field Documentation

5.4.2.1 psa_drv_entropy_init_t* psa_drv_entropy_t::p_init

Function that performs initialization for the entropy source

5.4.2.2 psa_drv_entropy_get_bits_t* psa_drv_entropy_t::p_get_bits

Function that performs the get_bits operation for the entropy source

The documentation for this struct was generated from the following file:

- /mnt/c/Users/dermil01/psa-crypto/mbedtls/include/psa/crypto_driver.h

5.5 psa_drv_key_derivation_t Struct Reference

A struct containing all of the function pointers needed to for key derivation and agreement.

`#include <crypto_driver.h>`

Data Fields

- psa_drv_key_derivation_setup_t * p_setup
- psa_drv_key_derivation_collateral_t * p_collateral
- psa_drv_key_derivation_derive_t * p_derive
- psa_drv_key_derivation_export_t * p_export

5.5.1 Detailed Description

A struct containing all of the function pointers needed to for key derivation and agreement.

PSA Crypto API implementations should populate instances of the table as appropriate upon startup.

If one of the functions is not implemented, it should be set to NULL.

5.5.2 Field Documentation

5.5.2.1 psa_drv_key_derivation_setup_t* psa_drv_key_derivation_t::p_setup

Function that performs the key derivation setup

5.5.2.2 psa_drv_key_derivation_collateral_t* psa_drv_key_derivation_t::p_collateral

Function that sets the key derivation collateral

5.5.2.3 psa_drv_key_derivation_derive_t* psa_drv_key_derivation_t::p_derive

Function that performs the final key derivation step

5.5.2.4 psa_drv_key_derivation_export_t* psa_drv_key_derivation_t::p_export

Function that performsm the final key derivation or agreement and exports the key

The documentation for this struct was generated from the following file:

- /mnt/c/Users/dermil01/psa-crypto/mbedtls/include/psa/crypto_driver.h

5.6 psa_drv_key_management_t Struct Reference

A struct containing all of the function pointers needed to for key management using opaque keys.

`#include <crypto_driver.h>`

Data Fields

- psa_drv_opaque_import_key_t * p_import
- psa_drv_destroy_key_t * p_destroy
- psa_drv_export_key_t * p_export
- psa_drv_export_public_key_t * p_export_public

5.6.1 Detailed Description

A struct containing all of the function pointers needed to for key management using opaque keys.

PSA Crypto API implementations should populate instances of the table as appropriate upon startup.

If one of the functions is not implemented, it should be set to NULL.

5.6.2 Field Documentation

5.6.2.1 psa_drv_opaque_import_key_t* psa_drv_key_management_t::p_import

Function that performs the key import operation

5.6.2.2 psa_drv_destroy_key_t* psa_drv_key_management_t::p_destroy

Function that performs the key destroy operation

5.6.2.3 psa_drv_export_key_t* psa_drv_key_management_t::p_export

Function that performs the key export operation

5.6.2.4 psa_drv_export_public_key_t* psa_drv_key_management_t::p_export_public

Function that performsm the public key export operation

The documentation for this struct was generated from the following file:

- /mnt/c/Users/dermil01/psa-crypto/mbedtls/include/psa/crypto_driver.h

5.7 psa_drv_mac_opaque_t Struct Reference

A struct containing all of the function pointers needed to implement MAC operations using opaque keys.

`#include <crypto_driver.h>`

Data Fields

- size_t context_size
- psa_drv_mac_opaque_setup_t * p_setup
- psa_drv_mac_opaque_update_t * p_update
- psa_drv_mac_opaque_finish_t * p_finish
- psa_drv_mac_opaque_finish_verify_t * p_finish_verify
- psa_drv_mac_opaque_abort_t * p_abort
- psa_drv_mac_opaque_generate_t * p_mac
- psa_drv_mac_opaque_verify_t * p_mac_verify

5.7.1 Detailed Description

A struct containing all of the function pointers needed to implement MAC operations using opaque keys.

PSA Crypto API implementations should populate the table as appropriate upon startup.

If one of the functions is not implemented (such as `psa_drv_mac_opaque_generate_t`), it should be set to NULL.

Driver implementers should ensure that they implement all of the functions that make sense for their hardware, and that they provide a full solution (for example, if they support `p_setup`, they should also support `p_update` and at least one of `p_finish` or `p_finish_verify`).

5.7.2 Field Documentation

5.7.2.1 size_t psa_drv_mac_opaque_t::context_size

The size in bytes of the hardware-specific Opaque-MAC Context structure

5.7.2.2 psa_drv_mac_opaque_setup_t* psa_drv_mac_opaque_t::p_setup

Function that performs the setup operation

5.7.2.3 psa_drv_mac_opaque_update_t* psa_drv_mac_opaque_t::p_update

Function that performs the update operation

5.7.2.4 psa_drv_mac_opaque_finish_t* psa_drv_mac_opaque_t::p_finish

Function that completes the operation

5.7.2.5 psa_drv_mac_opaque_finish_verify_t* psa_drv_mac_opaque_t::p_finish_verify

Function that completed a MAC operation with a verify check

5.7.2.6 psa_drv_mac_opaque_abort_t* psa_drv_mac_opaque_t::p_abort

Function that aborts a previoustly started operation

5.7.2.7 psa_drv_mac_opaque_generate_t* psa_drv_mac_opaque_t::p_mac

Function that performs the MAC operation in one call

5.7.2.8 psa_drv_mac_opaque_verify_t* psa_drv_mac_opaque_t::p_mac_verify

Function that performs the MAC and verify operation in one call

The documentation for this struct was generated from the following file:

- /mnt/c/Users/dermil01/psa-crypto/mbedtls/include/psa/crypto_driver.h

Index