IROT 硬件抽象层接口

Rev 1.0

Release Date: 2018-04-08

1.概述

1.1 目的

本文档主要提供给安全芯片厂商,厂商按下文描述封装硬件抽象层接口,提供给 SDK 调用，为上层提供基础安全服务。

2.硬件抽象层接口描述

2.1 读取 ID^2 的 ID

函数原型：irot_result_t irot_hal_get_id2(uint8_t* id2, uint32_t* len);
功能描述：读取 ID^2 的 ID
参数描述：[OUT] id2，ID 的起始地址
[IN/OUT] len，输入为缓冲区长度，输出为 ID 的实际长度。（**HEX 数据格式，当前长度为 12 字节**）
返 回 值：见 irot_result_t

2.2 对称算法

函数原型：
```
irot_result_t irot_hal_sym_crypto(key_object* key_obj, uint8_t key_id,
                                  const uint8_t* iv, uint32_t iv_len,
                                  const uint8_t* in, uint32_t in_len,
                                  uint8_t* out, uint32_t* out_len,
                                  sym_crypto_param_t* crypto_param
                                 );
```
功能描述：用对称算法实现数据的加解密
参数描述：[IN] key_obj， 密钥对象，见 key_object 类型
[IN] key_id， 密钥标识，索引内部密钥
[IN] iv，初始化向量地址
[IN] iv_len， 初始化向量长度
[IN] in，输入数据起始地址
[IN] in_len，输入数据长度
[OUT] out,，输出数据地址
[IN/OUT] out_len，输入为缓冲区大小，输出为真实数据长度
[IN] crypto_param，加解密参数结构，见 sym_crypto_param_t

返 回 值：见 irot_result_t

备注：key_obj 和 key_id 两个参数为二选一使用，当 key_obj 参数不为 NULL，则使用此参数作为密钥进行运算。当 key_obj 参数为 NULL，则使用 key_id 标识内部密钥进行运算。其它类似 API 同上。

2.3 非对称算法

2.3.1 私钥签名

函数原型：irot_result_t irot_hal_asym_priv_sign(key_object* key_obj, uint8_t key_id,
const uint8_t* in, uint32_t in_len,
uint8_t* out, uint32_t* out_len,
asym_sign_verify_t type);

功能描述：用私钥对数据进行签名

参数描述：[IN] key_obj，　密钥对象，见 key_object 类型
[IN] key_id，　密钥标识，索引内部密钥
[IN] in，待签名数据的起始地址
[IN] in_len，待签名数据的长度
[OUT] sign，签名数据的起始地址
[IN/OUT] sign_len，输入为缓冲区长度，输出为签名数据的实际长度
[IN] type，签名模式，见 asym_sign_verify_t

返 回 值：见 irot_result_t

2.3.2 私钥解密

函数原型：irot_result_t irot_hal_asym_priv_decrypt(key_object* key_obj, uint8_t key_id,
const uint8_t* in, uint32_t in_len,
uint8_t* out, uint32_t* out_len,
asym_padding_t padding);

功能描述：用私钥对数据进行解密

参数描述：[IN] key_obj，　密钥对象，见 key_object 类型
[IN] key_id，　密钥标识，索引内部密钥
[IN] in，密文数据的起始地址
[IN] in_len，密文数据的长度
[OUT] out，明文数据的起始地址
[IN/OUT] out_len，输入为缓冲区长度，输出为明文数据的实际长度
[IN] padding，填充方式，见 asym_padding_t

返 回 值：见 irot_result_t

2.4 哈希算法

函数原型：irot_result_t irot_hal_hash_sum(const uint8_t* in, uint32_t in_len,
uint8_t* out, uint32_t* out_len,

hash_t type);

功能描述：用指定算法对数据做摘要

参数描述：[IN] in，原始数据的起始地址

[IN] in_len，原始数据的长度

[OUT] out，摘要数据的起始地址

[IN/OUT] out_len，输入为缓冲区长度，输出为摘要数据的实际长度

[IN] type，摘要算法，见 hash_t

返 回 值：见 irot_result_t

3.附录

3.1 错误码定义

```
typedef enum
{
    IROT_SUCCESS                    = 0,   ///< The operation was successful.
    IROT_ERROR_GENERIC              = -1,  ///< Non-specific cause.
    IROT_ERROR_BAD_PARAMETERS       = -2,  ///< Input parameters were invalid.
    IROT_ERROR_SHORT_BUFFER         = -3,  ///< The supplied buffer is too short for the output.
    IROT_ERROR_EXCESS_DATA          = -4,  ///< Too much data for the requested operation was passed.
    IROT_ERROR_OUT_OF_MEMORY        = -5,  ///< System out of memory resources.
    IROT_ERROR_COMMUNICATION        = -7,  ///< Communication error
    IROT_ERROR_NOT_SUPPORTED        = -8,  ///< The request operation is valid but is not supported in this implementation.
    IROT_ERROR_NOT_IMPLEMENTED      = -9,  ///< The requested operation should exist but is not yet implementation.
    IROT_ERROR_TIMEOUT              = -10,///< Communication Timeout
    IROT_ERROR_ITEM_NOT_FOUND       = -11,///< Id2 is not exist
} irot_result_t;
```

3.2 类型定义

3.2.1 对称算法-算法类型

```
typedef enum
{
    CIPHER_TYPE_INVALID     = 0x00,
    CIPHER_TYPE_AES         = 0x01,
    CIPHER_TYPE_3DES        = 0x03,
    CIPHER_TYPE_SM4         = 0x04,
} cipher_t;
```

3.2.2 对称算法-块模式

```
typedef enum
{
    BLOCK_MODE_ECB              = 0x00,
    BLOCK_MODE_CBC              = 0x01,
    BLOCK_MODE_CTR              = 0x02,
} block_mode_t;
```

3.2.3 对称算法-填充类型

```
typedef enum
{
    SYM_PADDING_NOPADDING       = 0x00,
    SYM_PADDING_PKCS5           = 0x02,
    SYM_PADDING_PKCS7           = 0x03,
} irot_sym_padding_t;
```

3.2.4 非对称算法-填充类型

```
typedef enum
{
    ASYM_PADDING_NOPADDING      = 0x00,
    ASYM_PADDING_PKCS1          = 0x01,
} irot_asym_padding_t;
```

3.2.5 加密解密类型

```
typedef enum
{
    MODE_DECRYPT                = 0x00,
    MODE_ENCRYPT                = 0x01,
} crypto_mode_t;
```

3.2.6 非对称算法-签名类型

```
typedef enum
{
    ASYM_TYPE_RSA_MD5_PKCS1        = 0x00,
    ASYM_TYPE_RSA_SHA1_PKCS1       = 0x01,
    ASYM_TYPE_RSA_SHA256_PKCS1     = 0x02,
    ASYM_TYPE_RSA_SHA384_PKCS1     = 0x03,
    ASYM_TYPE_RSA_SHA512_PKCS1     = 0x04,
    ASYM_TYPE_SM3_SM2              = 0x05,
    ASYM_TYPE_ECDSA                = 0x06,
} asym_sign_verify_t;
```

3.2.7 哈希算法类型

```
typedef enum
{
    HASH_TYPE_SHA1                 = 0x00,
    HASH_TYPE_SHA224               = 0x01,
    HASH_TYPE_SHA256               = 0x02,
    HASH_TYPE_SHA384               = 0x03,
    HASH_TYPE_SHA512               = 0x04,
    HASH_TYPE_SM3                  = 0x05,
} hash_t;
```

3.2.8 对称算法加解密参数

```
typedef struct _sym_crypto_param_t
{
    cipher_t cipher_type;              ///< cipher type
    block_mode_t block_mode;           ///< block mode
    sym_padding_t padding_type;        ///< padding type
    mode_t mode;                       ///< mode(encrypt or decrypt)
} sym_crypto_param_t;
```

3.2.9 密钥对象-类型

```
enum
{
```

```
    KEY_TYPE_3DES              = 0x01,
    KEY_TYPE_AES               = 0x02,
    KEY_TYPE_SM4               = 0x03,

    KEY_TYPE_RSA_PUBLIC        = 0x04,
    KEY_TYPE_RSA_PRIVATE       = 0x05,
    KEY_TYPE_RSA_CRT_PRIVATE = 0x06,
};
```

3.2.10 通用-密钥对象

```
typedef struct
{
    struct
    {
        uint8_t key_object_type; ///< the key object type
    } head;
    struct
    {
        uint8_t buf[0x04];         ///< placeholder for key
    } body;
} key_object;
```

3.2.11 对称算法-密钥对象

```
typedef struct
{
    struct
    {
        uint8_t key_object_type;
    } head;
    struct
    {
        uint8_t* key_value; ///< the key value
        uint32_t key_len;     ///< the key length(bytes)
    } body;
} key_object_sym;
```

3.2.12 RSA 公钥-密钥对象

```
typedef struct
{
    struct
    {
        uint8_t key_object_type;
    } head;
    struct
    {
        uint8_t* e;           ///< public exponent
        uint32_t e_len;       ///< public exponent length(bytes)
        uint8_t* n;           ///< public modulus
        uint32_t n_len;       ///< public modulus length(bytes)
    } body;
} key_object_rsa_public;
```

3.2.13 RSA 私钥-密钥对象

```
typedef struct
{
    struct
    {
        uint8_t key_object_type;
    } head;
    struct
    {
        uint8_t* d;           ///< private exponent
        uint32_t d_len;       ///< private exponent length(bytes)
        uint8_t* n;           ///< private modulus
        uint32_t n_len;       ///< private modulus length(bytes)
    } body;
} key_object_rsa_private;
```

3.2.14 RSA CRT 格式私钥-密钥对象

```
typedef struct
{
    struct
    {
```

```
        uint8_t key_object_type;
    } head;
    struct
    {
        uint8_t* p;          ///< 1st prime factor
        uint8_t* q;          ///< 2st prime factor
        uint8_t* dp;         ///< d % (p - 1)
        uint8_t* dq;         ///< d % (q - 1)
        uint8_t* qinv;       ///< (1/q) % p
        uint32_t len;        ///< the length for the 5 parameters must with the same length(bytes)
    } body;
} key_object_rsa_crt_private;
```