iscrypt_sec 系列算法函数库

使用手册

（版本：1.0.0）

北京华大信安科技有限公司
Beijing Huada Infosec Technology Co.,Ltd

2014 年 11 月

声　明

北京华大信安科技有限公司保留未经通知用户对本文档内容进行修改的权利，我们会对手册的内容进行定期的审查，欢迎提出改进意见，建议您在最终设计前从北京华大信安科技有限公司获取本文档的最新版本。

目　录

第 1 章 概 述

1.1 iscrypt_sec 系列算法函数库概述

iscrypt_sec 为针对华大信安 IS8U256 系列芯片定制，已通过安全检测机构核定的安全算法库，包含 iscrypt_random_sec.lib、iscrypt_des_tdes_sec.lib、iscrypt_rsa2048_sec.lib 、 iscrypt_sm1_sec.lib 、 iscrypt_sm4_sec.lib 和 iscrypt_sm2_sec.lib 等几个库文件，分别提供了随机数生成、DES 和 Triple DES 的 ECB 和 CBC 模式的加解密、RSA1024 和 RSA2048 的 CRT 模式的解密和签名、SM1、SM4 的 ECB 和 CBC 模式的加解密、SM2 生成密钥对、加解密、签名验证、密钥协商等功能。同时，该算法函数库还包括 SHA、SM3 哈希算法。

1.2 iscrypt_sec 系列算法函数库及头文件说明

iscrypt_sec 系列算法函数库中配套提供了以下头文件：

头文件	描述	其他头文件关联
iscrypt_random_sec.h	提供随机数的自检函数和安全生成随机数函数的声明。	iscrypt.h
iscrypt_des_selftest.h	提供 DES 的自检函数声明。	无
iscrypt_rsa2048_selftest.h	提供 RSA 的自检函数声明。	无
iscrypt_des_tdes_sec.h	提供安全的 DES 和 Triple DES 的函数声明。	iscrypt.h iscrypt_symmetric.h
iscrypt_rsa2048_sec.h	提供安全的 RSA 解密和签名的函数声明。	iscrypt.h iscrypt_rsa.h
iscrypt_sm1_sec.h	提供安全的 SM1 函数声明。	iscrypt.h iscrypt_symmetric.h
iscrypt_sm2_sec.h	提供安全的 SM2 加解密、签名、密钥生成等函数声明，提供验证、密钥协商等函数声明。	iscrypt.h iscrypt_sm2.h
iscrypt_sm4_sec.h	提供安全的 SM4 函数声明。	iscrypt.h iscrypt_symmetric.h

iscrypt_sha.h	提供 SHA1、SHA256 函数声明。	无
iscrypt_sm3.h	提供 SM3 函数声明。	无
iscrypt_selftest.h	提供国标算法、SHA 算法的自检函数声明。	无

由于一些安全措施的采用，部分函数库的使用涉及随机数生成的操作，因此 iscrypt_des_tdes_sec.lib 、 iscrypt_rsa2048_sec.lib 、 iscrypt_sm1_sec.lib 、iscrypt_sm2_sec.lib 和 iscrypt_sm4_sec.lib 使用过程中，必须添加 iscrypt_random.lib 同时包含头文件 iscrypt_random.h（非 iscrypt_random_sec.h，该头文件中所声明的函数为外部使用）。

第 2 章 iscrypt_sec 系列算法函数库使用介绍

2.1 iscrypt_xxxx_sec.lib 的使用

使用者建立自己的 KEIL 工程后，将提供的需要使用库文件和头文件添加进工程中。以下以 iscrypt_des_tdes_sec.lib 为示例，将 iscrypt_des_tdes_sec.lib、iscrypt_random.lib 、 iscrypt.h 、 iscrypt_symmetric.h 、 iscrypt_random.h 和 iscrypt_des_tdes_sec.h 添加进工程中即可。

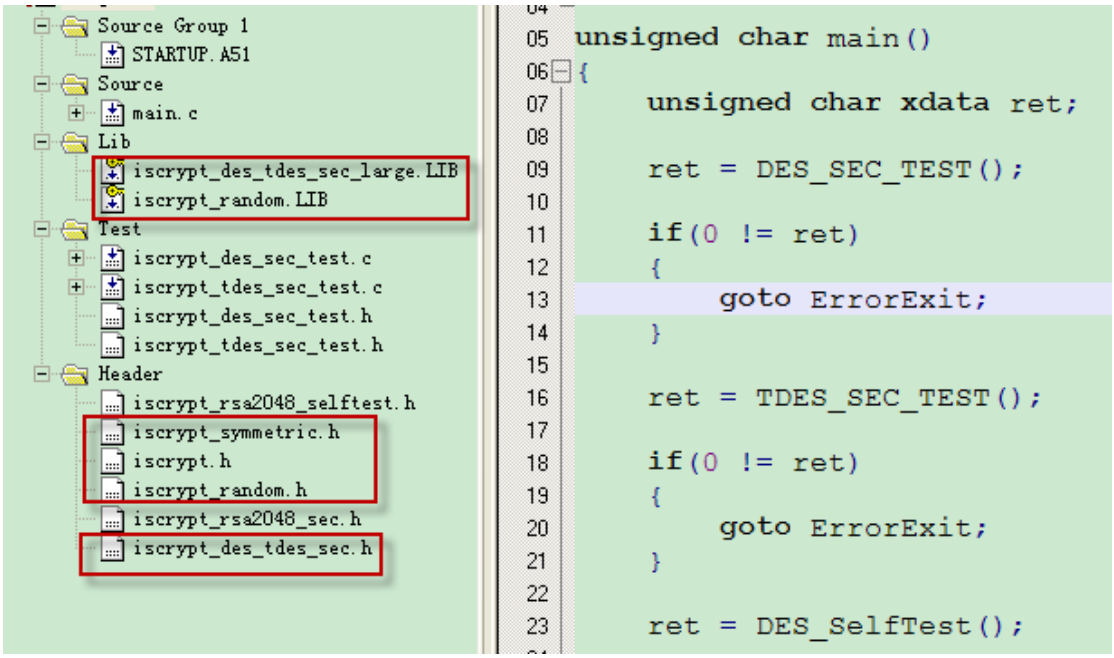


图 2.1 添加文件至工程

2.2 iscrypt_xxxx_sec.lib 的各模块说明

鼠标右键点击“iscrypt_xxxx_sec.lib”，选择“Options for Files ‘iscrypt_xxxx_sec.lib’”，弹出该对话框后，在“Select Modules to Always Include”中，可看到相应库文件中所包含的模块，以下对各个库文件中的模块进行简要说明，以供外部灵活使。

2.2.1 iscrypt_random_sec.lib 的模块说明

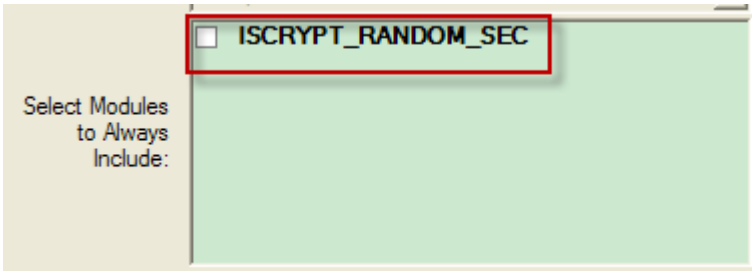


图 2.2 iscrypt_random_sec.lib 的模块说明

从图可知，iscrypt_random_sec.lib 中只包含有一个模块。

ISCRYPT_RANDOM_SEC：提供随机数自检（上电检测、在线检测等）功能，提供外部使用的随机数生成功能。

2.2.2 iscrypt_des_tdes_sec.lib 的模块说明

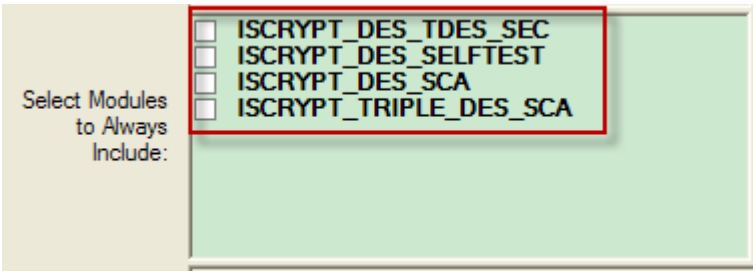


图 2.3 iscrypt_des_tdes_sec.lib 的模块说明

从图可知，iscrypt_des_tdes_sec.lib 中包含有四个模块。

ISCRYPT _DES_SCA：提供可抵御 SCA 攻击的 DES 加解密的功能。

ISCRYPT_TRIPLE_DES_SCA：提供可抵御 SCA 攻击的 Triple DES 加解密的功能。

ISCRYPT_DES_SELFTEST：提供 DES 自检功能，实质为内部进行一次加解密的测试，所用数据与外部无关。

ISCRYPT_DES_TDES_SEC：提供可抵御 SCA 攻击和 FA 攻击的 DES 和 Triple DES 的加解密功能。

外部使用者可根据自身应用环境需要选择合适的模块使用或全部使用。

2.2.3 iscrypt_rsa2048_sec.lib 的模块说明

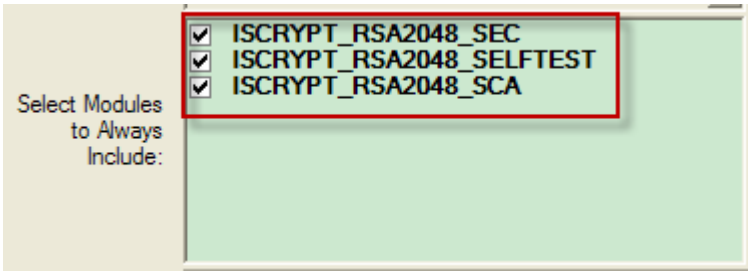


图 2.4 iscrypt_rsa2048_sec.lib 的模块说明

从图可知，iscrypt_rsa2048_sec.lib 中包含有三个模块。

ISCRYPT_ RSA2048_SCA：提供可抵御 SCA 攻击的 RSA 解密和签名的功能。

ISCRYPT_RSA2048_SELFTEST：提供 RSA 自检功能，实质为内部进行一次加解密的测试，所用数据与外部无关。

ISCRYPT_RSA2048_SEC：提供可抵御 SCA 攻击和 FA 攻击的 RSA 解密和签名功能。

外部使用者可根据自身应用环境需要选择合适的模块使用或全部使用。

2.2.4 iscrypt_sm1_sec.lib 的模块说明

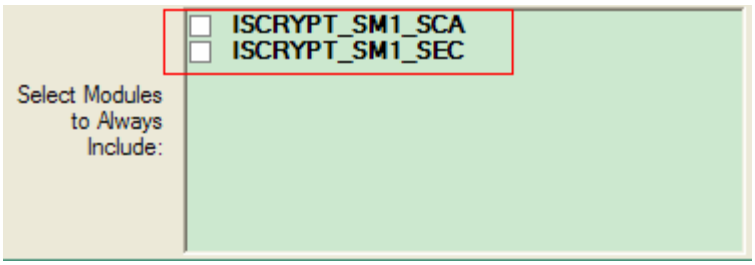


图 2.5 iscrypt_sm1_sec.lib 的模块说明

从图可知，iscrypt_sm1_sec.lib 中包含有两个模块。

ISCRYPT _SM1_SCA：提供可抵御 SCA 攻击的 SM1 加解密的功能。

ISCRYPT_SM1_SEC：提供可抵御 SCA 攻击和 FA 攻击的 SM1 的加解密功能。

外部使用者可根据自身应用环境需要选择合适的模块使用或全部使用。

2.2.5 iscrypt_sm2_sec.lib 的模块说明

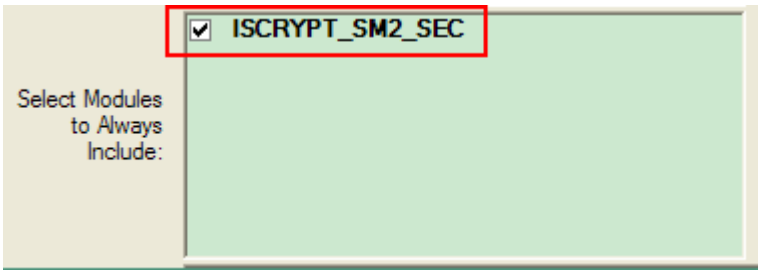


图 2.6 iscrypt_sm2_sec.lib 的模块说明

从图可知，iscrypt_sm2_sec.lib 中包含一个模块。

ISCRYPT_SM2_SEC：提供可抵御 SCA 攻击和 FA 攻击的 SM2 的加解密、签名等功能。

2.2.6 iscrypt_sm4_sec.lib 的模块说明

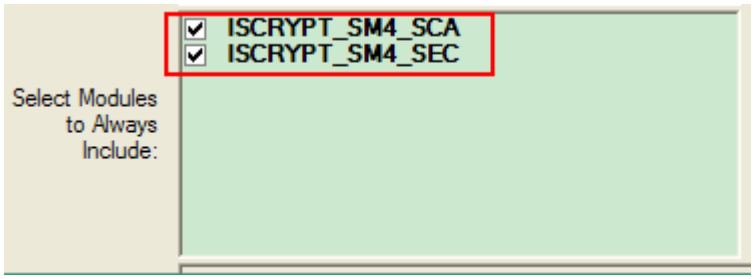


图 2.7 iscrypt_sm4_sec.lib 的模块说明

从图可知，iscrypt_sm4_sec.lib 中包含有两个模块。

ISCRYPT _SM4_SCA：提供可抵御 SCA 攻击的 SM4 加解密的功能。

ISCRYPT_SM4_SEC：提供可抵御 SCA 攻击和 FA 攻击的 SM4 的加解密功能。

外部使用者可根据自身应用环境需要选择合适的模块使用或全部使用。

注意事项：

1. **对于使用 IS8U256A 型的芯片，由于使用的 RSA 密钥生成的函数库为 iscrypt_rsa_gen_key_pair_kir.lib，Multi-BANK 配置时，必须将该库文件中的算法模块配置在 COMMON BANK，如下图，否则密钥生成函数调用时将返回错误。**

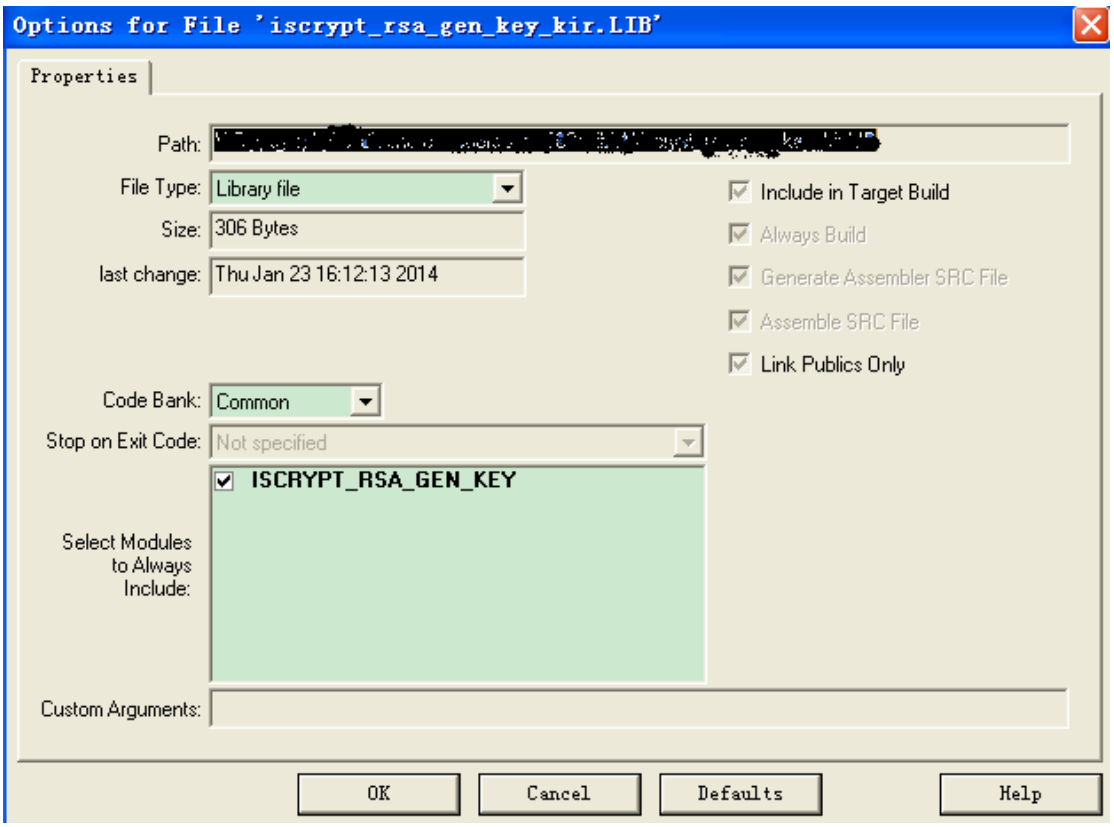


图 1　设置 iscrypt_rsa_gen_key_pair_kir 算法库文件存放的 BANK

2. IS8U256B 型的芯片使用 iscrypt_rsa_gen_key_pair_kif.lib，不能使用 iscrypt_rsa_gen_key_pair_kir.lib，且无上述的必须设置为 COMMON BANK 的要求，因此可按照一般库的使用进行设置。

3. 除上述描述的关于 RSA 生成密钥的函数库外，其他函数库使用时不应放在 COMMON BANK。

第 3 章 CSDK Sec 的函数接口说明

3.1 DES

3.1.1 DESCryptSEC

DESCryptSEC 函数用于提供可抵御 SCA 和 FA 攻击的 DES 算法的加解密功能操作。

```
#include “iscrypt_des_tdes_sec.h”
unsigned char DESCryptSEC (
    BLOCKCIPHERPARAM xdata *pBlockCipherParam        //    in,out
);
```

参数

pBlockCipherParam

分组密码算法结构体指针，由以下部分组成。

unsigned char *bOpMode*

操作类型选择，当前支持以下几种选择。

Value	Description
SYM_ECB	ECB 模式
SYM_CBC	CBC 模式
SYM_ENCRYPT	加密操作
SYM_DECRYPT	解密操作

unsigned char xdata **pbKey*

指向密钥的存储区，对于 DES 为 8 字节。

unsigned char xdata **pbInput*

输入数据的指针。

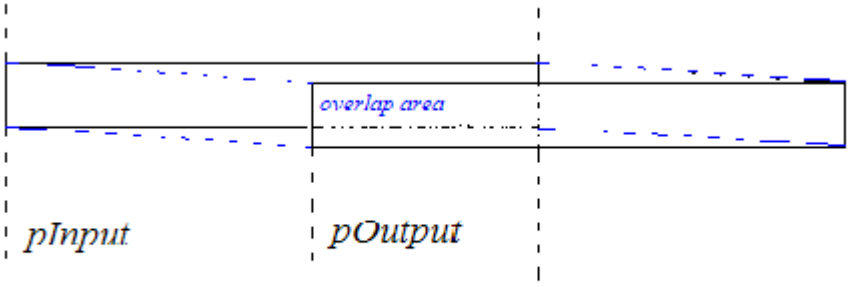
unsigned char *bTotalBlock*

输入数据的分组数。对于 DES 为以 DES_BLOCK_SIZE（定义在头文件中）个字节为一分组。注意，当该参数为 0x00 时，算法库按照 256 个分组执行。

unsigned char xdata **pbOutput*

输出数据的存放地址。对于 DES 来说，ECB 模式进行操作时，*pbInput* 可以与 *pbOutput* 相同；

同时要求，*pbOutput* 指向的存储区域不可与 *pbInput* 指向的存储区域有前向重叠。下图所示情况，在 DES 进行 ECB 模式的加解密时是不允许的，否则不保证处理结果的正确性。



unsigned char xdata **pbReserve*

保留指针，此处作为安全级别输入项，指向外部用户定义的安全级别参数。当前支持的安全级别为：

Value	**Description**
D_LOW_LEVEL_S	低安全级别
D_MEDIUM_LEVEL_S	中安全级别
D_HIGH_LEVEL_S	高安全级别

返回值

如果函数执行成功，返回值为 SUCCESS。如果失败，错误代码列举如下。

Error Code	**Description**
NOT_SUPPORT_YET_ERROR	暂不支持

FAIL	运行错误

注意

1. 由于参数 *bTotalBlock* 的限制，DESCryptSEC 函数执行一次的最大吞吐量为 256 个分组，共计 2KB 的数据量。
2. **由于防御措施的实施，需要使用 1KB 的 PKU RAM。**
3. **应用环境中，如果外部密钥每次都是变化的，则可不使用提供的 DESCryptSEC 函数。在应用环境中，对密钥有使用限制、密钥生命周期很短或密钥频繁更换的情况下，使用基本的 DES 加解密函数即可，此时不失安全性。**
4. 使用时，需要包含 iscrypt_random.lib。

快速信息

头文件：声明在 iscrypt_des_tdes_sec.h。

库文件：使用 iscrypt_des_tdes_sec.lib。

示例

```
#include <string.h>
#include "iscrypt_des_tdes_sec.h"

extern unsigned char xdata space_for_test[1024];
extern BLOCKCIPHERPARAM xdata BlockCipherParam;

unsigned char DES_SEC_TEST()
{
    unsigned char xdata ret;
    unsigned char xdata sl = D_HIGH_LEVEL_S;

    memcpy(space_for_test, DES_KEY, DES_BLOCK_SIZE);
    memcpy(space_for_test+DES_BLOCK_SIZE,DES_PLAINTEXT,DES_BLOCK_SIZ
E);
```

```
        BlockCipherParam.bOpMode = SYM_ECB | SYM_ENCRYPT;
        BlockCipherParam.pbKey = space_for_test;
        BlockCipherParam.pbInput = space_for_test+DES_BLOCK_SIZE;
        BlockCipherParam.bTotalBlock = 1;
        BlockCipherParam.pbOutput = space_for_test+2*DES_BLOCK_SIZE;
        BlockCipherParam.pbReserve = &sl;

        ret = DESCryptSEC(&BlockCipherParam);
        if (SUCCESS != ret)
        {
            return ret;
        }

        ret=memcmp(space_for_test+2*DES_BLOCK_SIZE,DES_CIPHERTEXT_ECB,DES
_BLOCK_SIZE);
        if (0 != ret)
        {
            return FAIL;
        }

        return SUCCESS;
    }
```

参阅

TDESCryptSEC、BLOCKCIPHERPARAM。

3.1.2 TDESCryptSEC

TDESCryptSEC 函数用于提供可抵御 SCA 和 FA 攻击的 Triple DES 算法的加解密功能操作。

```
#include “iscrypt_des_tdes_sec.h”
BYTE TDESCrypt SEC(
    BLOCKCIPHERPARAM xdata *pBlockCipherParam        //  in,out
);
```

参数

pBlockCipherParam

分组密码算法结构体指针。，由以下部分组成。

unsigned char *bOpMode*

操作类型选择，当前支持以下几种选择。

Value	**Description**
SYM_ECB	ECB 模式
SYM_CBC	CBC 模式
SYM_ENCRYPT	加密操作
SYM_DECRYPT	解密操作

unsigned char xdata **pbKey*

指向密钥的存储区，对于 TDES 密钥长度为 16 字节。算法库中提供的 TDES 为“DES-EDE2”模式，依次使用加密-解密-加密算法，其中第一次和第三次使用的密钥相同，即 *K1=K3*。

unsigned char xdata **pbInput*

输入数据的指针。

unsigned char *bTotalBlock*

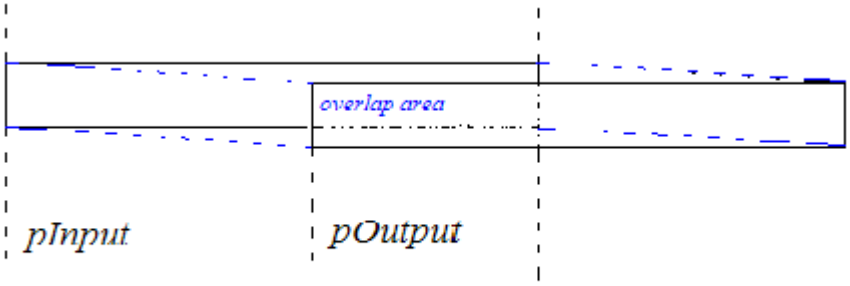
输入数据的分组数。对于 TDES 为以 TDES_BLOCK_SIZE 个字节

为一分组。注意，当该参数为 0x00 时，算法库按照 256 个分组执行。

unsigned char xdata **pbOutput*

输出数据的存放地址。对于 TDES 来说，ECB 模式进行操作时，*pbInput* 可以与 *pbOutput* 相同；

同时要求，*pbOutput* 指向的存储区域不可与 *pbInput* 指向的存储区域有前向重叠。下图所示情况，在 TDES 进行 ECB 模式的加解密时是不允许的，否则不保证处理结果的正确性。



unsigned char xdata **pbReserve*

保留指针，此处作为安全级别输入项，指向外部用户定义的安全级别参数。当前支持的安全级别为：

Value	Description
D_LOW_LEVEL_S	低安全级别
D_MEDIUM_LEVEL_S	中安全级别
D_HIGH_LEVEL_S	高安全级别

返回值

如果函数执行成功，返回值为 SUCCESS。如果失败，错误代码列举如下。

Error Code	Description
NOT_SUPPORT_YET_ERROR	暂不支持
FAIL	运行错误

注意

1. 由于参数 *bTotalBlock* 的限制，TDESCryptSEC 函数执行一次的最大吞吐量为 256 个分组，共计 2KB 的数据量。
2. **由于防御措施的实施，需要使用 1KB 的 PKU RAM。**
3. **应用环境中，如果外部密钥每次都是变化的，则可不使用提供的 TDESCryptSEC 函数。在应用环境中，对密钥有使用限制、密钥生命周期很短或密钥频繁更换的情况下，使用基本的 Triple DES 加解密函数即可，此时不失安全性。**
4. 使用时，需要包含 iscrypt_random.lib。

快速信息

头文件：声明在 iscrypt_des_tdes_sec.h。

库文件：使用 iscrypt_des_tdes_sec.lib。

示例

无。

参阅

DESCryptSEC、BLOCKCIPHERPARAM。

3.2 RSA

3.2.1 RSAGenKeyPair

RSAGenKeyPair 函数用于生成设定位长的 RSA 的公私钥，其中 RSA 的公钥指数需事先指定。

```
#include “iscrypt_rsa.h”
unsigned char RSAGenKeyPair(
    unsigned char bMode,                          // in
    RSAPUBLICKEYCTX xdata *pRSAPubKey,            // in, out
    RSAPRIVATEKEYCTX xdata *pRSAPriKey            // out
);
```

参数

bMode

RSA 操作属性，当前支持方式及位标志图如下：

Bit7~Bit4 (*index*)	Bit3 (*type-flag*)	Bit2~Bit1	Bit0 (*bit-flag*)
位长索引： 值为 0 时，为选择 1024 或 2048 位长的 RSA，此时通过 Bit0 决定。 值为 1~15 时，依次对应 1024+64*i* 位长的 RSA。	类型标志	Reserve	位长标志

Value	**Description**
RSA1024	1024 位长的 RSA
RSA2048	2048 位长的 RSA

STD_TYPE	标准的 RSA 密钥对儿结构 ((*e*,*n*), (*d*,*n*))
CRT_TYPE	使用中国剩余定理的密钥对儿结构((*e*,*n*), (*p*,*q*,*dP*,*dQ*,*qInv*))

当使用 1024 和 2048 之间的任意位长的 RSA 时，*bMode* 中的索引值需要按照以下方式计算：

设选取的位长为 *nBits*，*nBits* = 64*k* 且 1024<*nBits*<2048。

bMode.index = (*nBits*-1024)/64

当使用字节表示时，设字节长为 *nBytes*，*nBytes* = 8*k* 且 128<*nBytes*<256。

bMode.index = (*nBytes*-128)/8

pRSAPubKey

指向RSA公钥结构体 **RSAPUBLICKEYCTX**，公钥指数由函数调用者传入，位长不大于 32 位。根据标准，通常推荐的公钥指数选取 65537。

pRSAPriKey

指向RSA私钥结构体 **RSAPRIVATEKEYCTX**，根据函数调用者对当前RSA的操作属性的选择生成。RSA的私钥可以有两种形式，(*d*,*n*)和(*p*,*q*,*dP*,*dQ*,*qInv*)，后者对RSA的私钥运算有提升性能的作用。

返回值

如果函数执行成功，返回值为 SUCCESS。如果失败，错误代码列举如下。

Error Code	**Description**
MEMORY_ERROR[1]	函数存放地址错误

注意

1. 在 RAM 紧张的应用情况下，RSA 生成密钥对儿的操作属性选择 STD_TYPE,公钥结构体中的模数 *n* 可与私钥结构体中的模数 *n* 共享同一片

[1] 该错误只针对使用 IS8U256A 型的芯片，此时使用的是 iscrypt_rsa_gen_key_pair_kir.lib。

存储区。

2. **函数 RSAGenKeyPair 调用时，除使用 1KB 的 PKU RAM 外，还会独占内部申请的 IS_BUFFER_RSA[1284] 作为中间处理数据使用。IS_BUFFER_RSA[1284]仅在 RSAGenKeyPair 运行时使用，其后可复用为其它用途。**

3. **IS_BUFFER_RSA[1284]绝对地址定位于 0x0000 处。**

快速信息

头文件：声明在 iscrypt_rsa.h。

库文件：使用iscrypt_rsa_gen_key_pair_xxx.lib[2]。

示例

```
#include “iscrypt_rsa.h”
RSAPUBLICKEYCTX xdata RSAPubKey;
RSAPRIVATEKEYSTD xdata RSAPriKeySTD;
RSAPRIVATEKEYCRT xdata RSAPriKeyCRT;

unsigned char xdata space_for_test[1024];

void GENERATE_KEY_TEST()
{
    unsigned char xdata len;

    space_for_test[0] = 0x00;
    space_for_test[1] = 0x01;
    space_for_test[2] = 0x00;
    space_for_test[3] = 0x01;

    RSAPubKey.e = &space_for_test[0];
```

[2]如使用的为 IS8U256A 型芯片则选用 iscrypt_rsa_gen_key_pair_kir.lib；使用的为 IS8U256B 型则选用 iscrypt_rsa_gen_key_pair_kif.lib

```
        RSAPubKey.n = &space_for_test[4];

        RSAPriKeySTD.d = &space_for_test[4+256];
        RSAPriKeySTD.n = &space_for_test[4];

        RSAPriKeyCRT.p = &space_for_test[4+256];
        RSAPriKeyCRT.q = &space_for_test[260+128];
        RSAPriKeyCRT.dP = &space_for_test[388+128];
        RSAPriKeyCRT.dQ = &space_for_test[516+128];
        RSAPriKeyCRT.qInv = &space_for_test[644+128];

        /* RSA-1024 */
        RSAGenKeyPair(RSA1024 | STD_TYPE, &RSAPubKey, (RSAPRIVATEKEYCTX
*)&RSAPriKeySTD);
    //  RSAGenKeyPair(RSA1024 | CRT_TYPE, &RSAPubKey, (RSAPRIVATEKEYCTX
*)&RSAPriKeyCRT);

        /* RSA-1984 */
        len = 248;
        len -= 128;
        len /= 8;
        len <<= 4;
    //  RSAGenKeyPair(len | STD_TYPE, &RSAPubKey, (RSAPRIVATEKEYCTX
*)&RSAPriKeySTD);
    RSAGenKeyPair(len | CRT_TYPE, &RSAPubKey, (RSAPRIVATEKEYCTX
*)&RSAPriKeyCRT);

        return ;
    }
```

参阅

RSAPUBLICKEYCTX、RSAPRIVATEKEYCTX、RSAEncrypt、RSA DecryptSEC、RSASignSEC、RSAVerify

3.2.2 RSAEncrypt

RSAEncrypt 函数使用当前参数中的公钥，对指定的消息数据进行加密运算并输出运算结果至用户指定区域。

```
#include “iscrypt_rsa.h”
unsigned char RSAEncrypt(
    unsigned char bMode,                          // in
    RSAPUBLICKEYCTX xdata *pRSAPubKey,            // in
    unsigned char xdata *pbPlain                  // in, out
);
```

参数

bmode

RSA 操作属性，当前支持方式及位标志图如下：

Bit7~Bit4 (*index*)	Bit3 (*type-flag*)	Bit2~Bit1	Bit0 (*bit-flag*)
位长索引： 值为 0 时，为选择 1024 或 2048 位长的 RSA，此时通过 Bit0 决定。 值为 1~15 时，依次对应 1024+64*i* 位长的 RSA。	类型标志	Reserve	位长标志

Value	**Description**
RSA1024	1024 位长的 RSA
RSA2048	2048 位长的 RSA

当使用 1024 和 2048 之间的任意位长的 RSA 时，*bMode* 中的索引值需要按照以下方式计算：

设选取的位长为 *nBits*，*nBits* = 64*k* 且 1024<*nBits*<2048。

bMode.index = (*nBits*-1024)/64

当使用字节表示时，设字节长为 *nBytes*，*nBytes* = 8*k* 且 128<*nBytes*<256。

bMode.index = (*nBytes*-128)/8

pRSAPubKey

指向RSA公钥结构体 **RSAPUBLICKEYCTX**的指针。

pbPlain

RSA 明文，处理后的密文同时存储在此区域中。

返回值

如果函数执行成功，返回值为 SUCCESS。如果失败，返回值为 MESSAGE_OUT_OF_RANGE 。

注意

1. **函数运行时，将使用 1KB 的 PKU RAM。**

快速信息

头文件：声明在 iscrypt_rsa.h。

库文件：使用 iscrypt_rsa_kif.lib 或 iscrypt_rsa_kir.lib。

示例

```
#include <string.h>
#include "iscrypt_rsa.h"

unsigned char xdata space_for_test[1024];

unsigned char RSA_ENCRYPT_TEST()
{
    unsigned char xdata ret;
```

```
    space_for_test[0] = 0x00;
    space_for_test[1] = 0x01;
    space_for_test[2] = 0x00;
    space_for_test[3] = 0x01;

    /* RSA-1024 */
    memcpy(space_for_test+4, RSA1024_PUB_KEY_N, 128);
    memcpy(space_for_test+512, RSA_PLAINTEXT, 128);

    RSAPubKey.e = &space_for_test[0];
    RSAPubKey.n = &space_for_test[4];

    ret = (unsigned char)RSAEncrypt(RSA1024, &RSAPubKey, space_for_test+512);
    if (SUCCESS != ret)
    {
        goto ErrorExit;
    }

    return 0;

ErrorExit:
    return 1;
}
```

参阅

RSAPUBLICKEYCTX、RSADecryptSEC

3.2.3 RSADecryptSEC

RSADecryptSEC 函数使用当前参数中的公私钥，对指定的密文数据进行解密运算并输出运算结果，该函数能够抵御 SCA 和 FA 攻击。

```
#include “iscrypt_rsa2048_sec.h”
unsigned char RSADecryptSEC (
    unsigned char bMode,                    // in
    RSAKEYCTX xdata *pRSAKey,               //in
    unsigned char xdata * pbCiphertext      // in, out
);
```

参数

bMode

RSA 操作属性，当前仅支持以下方式。

Value	**Description**
RSA1024	1024 位长的 RSA
RSA2048	2048 位长的 RSA
CRT_TYPE	使用中国剩余定理的私钥结构 (*p*,*q*,*dP*,*dQ*,*qInv*)

pRSAKey

RSA 密钥结构体指针，由两部分组成：

pRSAPubKey

公钥结构体指针，指向外部定义的公钥结构体。

pRSAPriKey

私钥结构体指针，指向外部定义的私钥结构体。

pbCiphertext

RSA 密文，处理后的明文同时存储在此区域中。

返回值

如果函数执行成功，返回值为 SUCCESS。如果失败，错误代码列举如下。

Error Code	Description
MESSAGE_OUT_OF_RANGE	处理的消息数据超出范围

注意

1. 该 RSA 解密函数可防 SCA 和 FA 攻击，从而保护私钥在运算过程中的安全性。
2. **函数使用时请注意，该 RSADecryptSEC 函数将使用从 RAM 起始地址的 1284 字节的空间。**
3. **应用环境中常有使用临时密钥对数据进行解密的时候，此时不必使用提供的 RSADecryptSEC 函数，使用 RSADecrypt 即可不失安全性，从而提高运算速度。**

快速信息

头文件：声明在 iscrypt_rsa2048_sec.h。

库文件：使用 iscrypt_rsa2048_sec.lib。

示例

```
#include <string.h>
#include "iscrypt_rsa2048_sec.h"
unsigned char xdata space_for_test[1536];

RSAKEYCTX xdata RSAKey;
RSAPUBLICKEYCTX xdata RSAPubKey;
RSAPRIVATEKEYCRT xdata RSAPriKeyCRT;

unsigned char RSA1024_DECRYPT_SEC_TEST()
{
    unsigned char xdata ret;
```

```
memcpy(space_for_test, RSA1024_PRI_KEY_P, 64);
memcpy(space_for_test+128, RSA1024_PRI_KEY_Q, 64);
memcpy(space_for_test+256, RSA1024_PRI_KEY_DP, 64);
memcpy(space_for_test+384, RSA1024_PRI_KEY_DQ, 64);
memcpy(space_for_test+512, RSA1024_PRI_KEY_QINV, 64);
memcpy(space_for_test+640, RSA1024_CIPHERTEXT, 128);

memcpy(space_for_test+768, RSA1024_PUB_KEY_N, 128);

space_for_test[896] = 0x00;
space_for_test[897] = 0x01;
space_for_test[898] = 0x00;
space_for_test[899] = 0x01;

RSAPriKeyCRT.p = space_for_test;
RSAPriKeyCRT.q = space_for_test+128;
RSAPriKeyCRT.dP = space_for_test+256;
RSAPriKeyCRT.dQ = space_for_test+384;
RSAPriKeyCRT.qInv = space_for_test+512;

RSAPubKey.e = space_for_test+896;
RSAPubKey.n = space_for_test+768;

RSAKey.pRSAPubKey = &RSAPubKey;
RSAKey.pRSAPriKey = &RSAPriKeyCRT;

ret = RSADecryptSEC(RSA1024|CRT_TYPE, &RSAKey, space_for_test+640);

if (SUCCESS != ret)
{
    goto ErrorExit;
```

```
    }

    ret = memcmp(space_for_test+640, RSA_PLAINTEXT, 128);

    if (0 != ret)
    {
        goto ErrorExit;
    }

    return 0;

ErrorExit:
    return 1;
}
```

查阅

RSASignSEC、RSAKEYCTX、RSAPRIVATEKEYCTX、RSAPRIVATEKEYSTD、RSAPRIVATEKEYCRT。

3.2.4 RSASignSEC

RSASignSEC 函数使用当前参数中的公私钥，对指定的消息摘要进行签名运算并输出运算签名结果，该函数能够抵御 SCA 和 FA 攻击。

```
#include “iscrypt_rsa2048_sec.h”
unsigned char RSASignSEC (
    unsigned char bMode,              // in
    RSAKEYCTX xdata *pRSAKey, //in
    unsigned char xdata * pbDigest   // in, out
);
```

参数

bMode

RSA 操作属性，当前仅支持以下方式。

Value	**Description**
RSA1024	1024 位长的 RSA
RSA2048	2048 位长的 RSA
CRT_TYPE	使用中国剩余定理的私钥结构 (*p*,*q*,*dP*,*dQ*,*qInv*)

pRSAKey

RSA 密钥结构体指针，由两部分组成：

pRSAPubKey

公钥结构体指针，指向外部定义的公钥结构体。

pRSAPriKey

私钥结构体指针，指向外部定义的私钥结构体。

pbDigest

待签名的摘要，摘要的签名值存储在此区域中，注意备份摘要数据。

返回值

如果函数执行成功，返回值为 SUCCESS。如果失败，错误代码列举如下。

Error Code	Description
MESSAGE_OUT_OF_RANGE	处理的消息数据超出范围

注意

1. 该 RSA 解密函数可防 SCA 和 FA 攻击，从而保护私钥在运算过程中的安全性。
2. **函数使用时请注意，该 RSADecryptSEC 函数将使用从 RAM 起始地址的 1284 字节的空间。**
3. **应用环境中常有使用临时密钥对数据进行签名的时候，此时不必使用提供的 RSASignSEC 函数，使用 RSASign 即可不失安全性，从而提高运算速度。**

快速信息

头文件：声明在 iscrypt_rsa2048_sec.h。

库文件：使用 iscrypt_rsa2048_sec.lib。

示例

```
#include <string.h>
#include "iscrypt_rsa2048_sec.h"
unsigned char xdata space_for_test[1536];

RSAKEYCTX xdata RSAKey;
RSAPUBLICKEYCTX xdata RSAPubKey;
RSAPRIVATEKEYCRT xdata RSAPriKeyCRT;

unsigned char RSA1024_SIGN_SEC_TEST()
{
    unsigned char xdata ret;

    memcpy(space_for_test, RSA1024_PRI_KEY_P, 64);
    memcpy(space_for_test+128, RSA1024_PRI_KEY_Q, 64);
```

```
    memcpy(space_for_test+256, RSA1024_PRI_KEY_DP, 64);
    memcpy(space_for_test+384, RSA1024_PRI_KEY_DQ, 64);
    memcpy(space_for_test+512, RSA1024_PRI_KEY_QINV, 64);
    memcpy(space_for_test+640, RSA1024_CIPHERTEXT, 128);

    memcpy(space_for_test+768, RSA1024_PUB_KEY_N, 128);
    space_for_test[896] = 0x00;
    space_for_test[897] = 0x01;
    space_for_test[898] = 0x00;
    space_for_test[899] = 0x01;

    RSAPriKeyCRT.p = space_for_test;
    RSAPriKeyCRT.q = space_for_test+128;
    RSAPriKeyCRT.dP = space_for_test+256;
    RSAPriKeyCRT.dQ = space_for_test+384;
    RSAPriKeyCRT.qInv = space_for_test+512;

    RSAPubKey.e = space_for_test+896;
    RSAPubKey.n = space_for_test+768;

    RSAKey.pRSAPubKey = &RSAPubKey;
    RSAKey.pRSAPriKey = &RSAPriKeyCRT;

    ret = RSASignSEC(RSA1024|CRT_TYPE, &RSAKey, space_for_test+640);

    if (SUCCESS != ret)
    {
        goto ErrorExit;
    }

    ret = memcmp(space_for_test+640, RSA_PLAINTEXT, 128);

    if (0 != ret)
    {
        goto ErrorExit;
    }

    return 0;

ErrorExit:
    return 1;
```

}

查阅

RSADecryptSEC、RSAKEYCTX、RSAPRIVATEKEYCTX、RSAPRIVATEKEYSTD、RSAPRIVATEKEYCRT。

3.2.5 RSAVerify

RSAVerify 函数使用当前参数中的公钥，对输入消息和消息签名值进行验证。

```
#include “iscrypt_rsa.h”
unsigned char RSAVerify(
    unsigned char bMode,                          // in
    RSAPUBLICKEYCTX xdata *pRSAPubKey,            // in
    unsigned char xdata *pbSignature,             // in
    unsigned char *pbDigest                       // in
)
```

参数

bmode

RSA 操作属性，当前支持方式及位标志图如下：

Bit7~Bit4 (*index*)	Bit3 (*type-flag*)	Bit2~Bit1	Bit0 (*bit-flag*)
位长索引： 值为 0 时，为选择 1024 或 2048 位长的 RSA，此时通过 Bit0 决定。 值为 1~15 时，依次对应 1024+64*i* 位长的 RSA。	类型标志	Reserve	位长标志

Value	**Description**
RSA1024	1024 位长的 RSA
RSA2048	2048 位长的 RSA

当使用 1024 和 2048 之间的任意位长的 RSA 时，*bMode* 中的索引值需

要按照以下方式计算：

设选取的位长为 *nBits*，*nBits* = 64*k* 且 1024<*nBits*<2048。

bMode.index = (*nBits*-1024)/64

当使用字节表示时，设字节长为 *nBytes*，*nBytes* = 8*k* 且 128<*nBytes*<256。

bMode.index = (*nBytes*-128)/8

pRSAPubKey

RSA公钥结构体 **RSAPUBLICKEYCTX**指针。

pbSignature

摘要的签名值，验证后的数据同时存储在此区域中。

pbDigest

原始摘要。

返回值

如果函数执行成功，返回值为 SUCCESS。如果失败，错误代码列举如下。

Error Code	**Description**
MESSAGE_OUT_OF_RANGE	处理的消息数据超出范围
FAIL	验证失败

注意

1. **函数运行时，将使用 1KB 的 PKU RAM。**
2. 该函数实现在 iscrypt_rsa_sv.c 中，使用时需要连同 iscrypt_rsa_kif.lib 或 iscrypt_rsa_kir.lib 一同加入用户工程当中。

快速信息

头文件：声明在 iscrypt_rsa.h。

库文件：使用 iscrypt_rsa_kif.lib 或 iscrypt_rsa_kir.lib。

源文件：使用 iscrypt_rsa_sv.c

示例

```
#include <string.h>
#include "iscrypt_rsa.h"

BYTE xdata space_for_test[1500];
RSAPUBLICKEYCTX xdata RSAPubKey;

cunsigned char RSA_VERIFY_TEST()
{
    unsigned char xdata ret;
    unsigned char xdata len;

    space_for_test[0] = 0x00;
    space_for_test[1] = 0x01;
    space_for_test[2] = 0x00;
    space_for_test[3] = 0x01;

    memcpy(space_for_test+4, RSA1408_PUB_KEY_N, 176);
    memcpy(space_for_test+512, RSA_PLAINTEXT, 176);

    RSAPubKey.e = &space_for_test[0];
    RSAPubKey.n = &space_for_test[4];

    len = 176;
    len -= 128;
    len /= 8;
    len <<= 4;
    ret = RSAVerify(len, &RSAPubKey, space_for_test+512, RSA1408_CIPHERTEXT);
    if (SUCCESS != ret)
    {
        goto ErrorExit;
    }
```

```
    return 0;

ErrorExit:
    return 1;
}
```

参阅

RSAPUBLICKEYCTX、RSAVerify.

3.3 SHA

3.3.1 SHA1Init

SHA1Init 函数用于完成基于 SHA-1 算法的初始化操作。

#include “iscrypt_sha.h”

void SHA1Init();

参数

无

返回值

无

注意

1. **函数运行时，将使用 1KB 的 PKU RAM。**

快速信息

头文件：声明在 iscrypt_sha.h。

库文件：使用 iscrypt_sha.lib。

示例

请参见 **SHA1Final**一节。

参阅

SHA1Update、SHA1Final.

3.3.2 SHA1Update

SHA1Update 用以对一个 SHA-1 的消息分组中间分组进行处理。

```
#include “iscrypt_sha.h”
void SHA1Update(
    unsigned char xdata *pbMessage      //    in
);
```

参数

pbMessage

需要被哈希处理的消息分组，对于 SHA-1 算法来说为 64 字节为一组。

返回值

无

注意

1. **函数运行时，将使用 1KB 的 PKU RAM。**

快速信息

头文件：声明在 iscrypt_sha.h。

库文件：使用 iscrypt_sha.lib。

示例

请参见 **SHA1Final**一节。

参阅

SHA1Init、SHA1Final.

3.3.3 SHA1Final

SHA1Final 用以对 SHA-1 的最后消息分组进行处理。

```
#include “iscrypt_sha.h”
unsigned char SHA1Final (
    unsigned char xdata *pbMessage      //  in
    unsigned char bLength,              //  in
    unsigned char xdata *pbDigest       //  out
);
```

参数

pbMessage

需要被哈希处理的末块儿消息分组。

bLength

末块儿消息分组的字节长度，小于 64，可以为 0。

pbDigest

SHA-1 处理结束后，摘要的存储区地址。SHA-1 摘要长度为 20 字节（160 比特）。

返回值

如果函数执行成功，返回值为 SUCCESS。如果失败，错误代码列举如下。

Error Code	Description
HASH_OBJECT_ERROR	哈希对象错误

注意

1. **函数运行时，将使用 1KB 的 PKU RAM。**

快速信息

头文件：声明在 iscrypt_sha.h。

库文件：使用 iscrypt_sha.lib。

示例

```
#include <string.h>
#include "iscrypt_sha.h"

unsigned char xdata space_for_test[1024];
char SHA1_TEST()
{
    unsigned char xdata ret;
    memcpy(space_for_test, SHA_VECTOR, 64);

    ////////////////////////////////////////////////////////////////
    SHA1Init();
    SHA1Update(space_for_test);
    SHA1Update(space_for_test);
    ret = SHA1Final(space_for_test, 0, space_for_test+0x200);
    if (SUCCESS != ret)
    {
        goto ErrorExit;
    }

    ret = memcmp(space_for_test+0x200, DIGEST_SHA1_128, 20);
    if (0 != ret)
    {
        goto ErrorExit;
    }

    return 0;
ErrorExit:
```

```
    return 1;
}
```

参阅

SHA1Init、SHA1Update.

3.3.4 SHA256Init

SHA256Init 函数用于完成基于 SHA-256 算法的初始化操作。

#include “iscrypt_sha.h”

void SHA256Init();

参数

无

返回值

无

注意

1. **函数运行时，将使用 1KB 的 PKU RAM。**

快速信息

头文件：声明在 iscrypt_sha.h。

库文件：使用 iscrypt_sha.lib。

示例

请参见 **SHA256Final**一节。

参阅

SHA256Update、SHA256Final.

3.3.5 SHA256Update

SHA256Update 用以对一个 SHA-256 的消息分组中间块儿进行处理。

```
#include “iscrypt_sha.h”
void SHA256Update(
    unsigned char xdata *pbMessage      //    in
);
```

参数

pbMessage

需要被哈希处理的消息分组，对于 SHA-256 算法来说为 64 字节为一组。

返回值

无

注意

1. **函数运行时，将使用 1KB 的 PKU RAM。**

快速信息

头文件：声明在 iscrypt_sha.h。

库文件：使用 iscrypt_sha.lib。

示例

请参见 **SHA256Final**一节。

参阅

SHA256Init、SHA256Final.

3.3.6 SHA256Final

SHA256Final 用以对一个 SHA-256 的末块儿消息分组进行处理。

```
#include “iscrypt_sha.h”
unsigned char SHA256Final (
    unsigned char xdata *pbMessage      //  in
    unsigned char bLength,              //  in
    unsigned char xdata *pbDigest       //  out
);
```

参数

pbMessage

需要被哈希处理的末块儿消息分组。

bLength

末块儿消息分组的字节长度，小于 64，可以为 0。

pbDigest

SHA2561 处理结束后，摘要的存储区地址。SHA-256 摘要长度为 32 字节（256 比特）。

返回值

如果函数执行成功，返回值为 SUCCESS。如果失败，错误代码列举如下。

Error Code	Description
HASH_OBJECT_ERROR	哈希对象错误

注意

1. **函数运行时，将使用 1KB 的 PKU RAM。**

快速信息

头文件：声明在 iscrypt_sha.h。

库文件：使用 iscrypt_sha.lib。

示例

```
#include <string.h>
#include "iscrypt_sha.h"
unsigned char xdata space_for_test[1024];

char SHA256_TEST()
{
    unsigned char xdata ret;
    memcpy(space_for_test, SHA_VECTOR, 64);

    ////////////////////////////////////////////////////////////////
    SHA256Init();
    SHA256Update(space_for_test);
    SHA256Update(space_for_test);
    ret = SHA256Final(space_for_test, 0, space_for_test+0x200);
    if (SUCCESS != ret)
    {
        goto ErrorExit;
    }

    ret = memcmp(space_for_test+0x200, DIGEST_SHA256_128, 32);
    if (0 != ret)
    {
        goto ErrorExit;
    }

    return 0;
ErrorExit:
```

```
    return 1;
}
```

参阅

SHA256Init、SHA256Update.

3.4 SM1

3.4.1 SM1CryptSEC

SM1CryptSEC 函数用于提供可抵御 SCA 和 FA 攻击的 SM1 算法的加解密功能操作。

```
#include “iscrypt_sm1_sec.h”
unsigned char SM1Crypt SEC(
    BLOCKCIPHERPARAM xdata *pBlockCipherParam          //    in,out
);
```

参数

pBlockCipherParam

分组密码算法结构体指针，由以下部分组成。

unsigned char *bOpMode*

操作类型选择，当前支持以下几种选择。

Value	**Description**
SYM_ECB	ECB 模式
SYM_CBC	CBC 模式
SYM_ENCRYPT	加密操作
SYM_DECRYPT	解密操作

unsigned char xdata **pbKey*

指向密钥的存储区，对于 SM1 为 16 字节。

unsigned char xdata **pbInput*

输入数据的指针。

unsigned char *bTotalBlock*

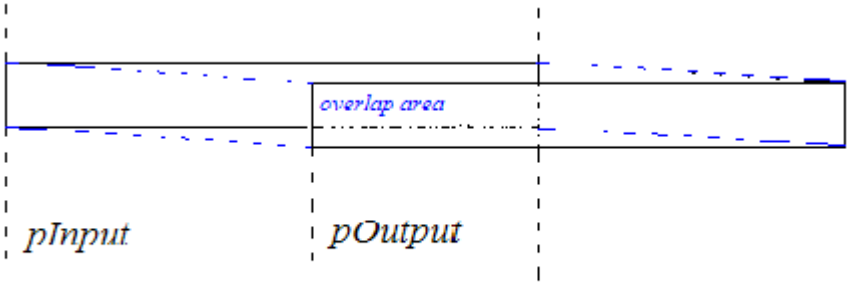
输入数据的分组数。对于 SM1 为以 SM1_BLOCK_SIZE（定义在头文件中）个字节为一分组。注意，当该参数为 0x00 时，算法库按照 256

个分组执行。

unsigned char xdata **pbOutput*

输出数据的存放地址。对于 SM1 来说，ECB 模式进行操作时，*pbInput* 可以与 *pbOutput* 相同；

同时要求，*pbOutput* 指向的存储区域不可与 *pbInput* 指向的存储区域有前向重叠。下图所示情况，在 SM1 进行 ECB 模式的加解密时是不允许的，否则不保证处理结果的正确性。



unsigned char xdata **pbReserve*

保留指针，此处作为安全级别输入项，指向外部用户定义的安全级别参数。当前支持的安全级别为：

Value	**Description**
D_LOW_LEVEL_S	低安全级别
D_MEDIUM_LEVEL_S	中安全级别
D_HIGH_LEVEL_S	高安全级别

返回值

如果函数执行成功，返回值为 SUCCESS。如果失败，错误代码列举如下。

Error Code	**Description**
NOT_SUPPORT_YET_ERROR	暂不支持
FAIL	运行错误

注意

1. 由于参数 *bTotalBlock* 的限制，SM1CryptSEC 函数执行一次的最大吞吐量为 256 个分组，共计 4KB 的数据量。
2. **由于防御措施的实施，需要使用 1KB 的 PKU RAM。**
3. **应用环境中，如果外部密钥每次都是变化的，则可不使用提供的 SM1CryptSEC 函数。在应用环境中，对密钥有使用限制、密钥生命周期很短或密钥频繁更换的情况下，使用基本的 SM1 加解密函数即可，此时不失安全性。**
4. 使用时，需要包含 iscrypt_random.lib。

快速信息

头文件：声明在 iscrypt_sm1_sec.h。

库文件：使用 iscrypt_sm1_sec.lib。

示例

```
#include <string.h>
#include "iscrypt_sm1_sec.h"

extern unsigned char xdata space_for_test[1024];
extern BLOCKCIPHERPARAM xdata BlockCipherParam;

char SM1_TEST()
{
    unsigned char ret;
    unsigned char xdata sl;

    sl = D_HIGH_LEVEL_S;

    memcpy(space_for_test, BLOCK_CRYPT_KEY, SM1_BLOCK_SIZE);
    memcpy(space_for_test+SM1_BLOCK_SIZE,BLOCK_CRYPT_IV,SM1_BLOCK_SI
ZE);

    ////////////////////////////////////////////////////////////////
```

```
        memcpy(space_for_test+2*SM1_BLOCK_SIZE,      BLOCK_CRYPT_PLAINTEXT,
4*SM1_BLOCK_SIZE);

        BlockCipherParam.bOpMode = SYM_ECB | SYM_ENCRYPT;
        BlockCipherParam.pbKey = space_for_test;
        BlockCipherParam.pbIV = space_for_test+SM1_BLOCK_SIZE;

        BlockCipherParam.pbInput = space_for_test+2*SM1_BLOCK_SIZE;
        BlockCipherParam.bTotalBlock = 4;
        BlockCipherParam.pbOutput = space_for_test+2*SM1_BLOCK_SIZE;
        BlockCipherParam.pbReserve = &sl;

        ret = SM1CryptSEC(&BlockCipherParam);
        if (SUCCESS != ret)
        {
            goto ErrorExit;
        }

        ret                =                memcmp(space_for_test+2*SM1_BLOCK_SIZE,
BLOCK_CRYPT_CIPHERTEXT_SM1_ECB, 4*SM1_BLOCK_SIZE);

        if (0 != ret)
        {
            goto ErrorExit;
        }

        ////////////////////////////////////////////////////////////////
        memcpy(space_for_test+2*SM1_BLOCK_SIZE,
BLOCK_CRYPT_CIPHERTEXT_SM1_ECB, 4*SM1_BLOCK_SIZE);

        BlockCipherParam.bOpMode = SYM_ECB | SYM_DECRYPT;
        BlockCipherParam.pbKey = space_for_test;
```

```
        BlockCipherParam.pbIV = space_for_test+SM1_BLOCK_SIZE;

        BlockCipherParam.pbInput = space_for_test+2*SM1_BLOCK_SIZE;
        BlockCipherParam.bTotalBlock = 4;
        BlockCipherParam.pbOutput = space_for_test+2*SM1_BLOCK_SIZE;
        BlockCipherParam.pbReserve = &sl;

        ret = SM1CryptSEC(&BlockCipherParam);
        if (SUCCESS != ret)
        {
            goto ErrorExit;
        }

        ret = memcmp(space_for_test+2*SM1_BLOCK_SIZE,
BLOCK_CRYPT_PLAINTEXT,  4*SM1_BLOCK_SIZE);

        if (0 != ret)
        {
            goto ErrorExit;
        }
        return 0;

    ErrorExit:
        return 1;
    }
```

参阅

BLOCKCIPHERPARAM、SM1CryptSEC.

3.5 SM2

3.5.1 SM2GenKeyPairSEC

SM2GenKeyPairSEC 函数用于安全地生成 SM2 算法的公私钥。

#include “iscrypt_sm2_sec.h”

unsigned char SM2GenKeyPairSEC(

**SM2PUBLICKEYCTX xdata ***pSM2PubKey*, // out

**SM2PRIVATEKEYCTX xdata ***pSM2PriKey* // out

);

参数

pSM2PubKey

指向SM2 公钥结构体 **SM2PUBLICKEYCTX**的指针。

pSM2PriKey

指向SM2 私钥结构体 **SM2PRIVATEKEYCTX**的指针。

返回值

该函数要求自身必须执行成功，返回 SUCCESS。

注意

1. 函数要求使用者自行开辟存储密钥的 RAM 区域。
2. **函数运行时，将使用 1KB 的 PKU RAM。**

快速信息

头文件：声明在 iscrypt_sm2_sec.h。

库文件：使用 iscrypt_sm2_sec.lib。

示例

#include <string.h>

```
#include "iscrypt_sm2_sec.h"
#include "iscrypt_sm2_test.h"

extern unsigned char xdata space_for_test[1024];

SM2PUBLICKEYCTX xdata PubKey_A;
SM2PRIVATEKEYCTX xdata PriKey_A;

char GENERATE_KEY_TEST()
{
    unsigned char ret;

    ////////////////////////////////////////////////////////////
    PubKey_A.x = space_for_test;
    PubKey_A.y = space_for_test+32;

    PriKey_A.d = space_for_test+64;

    ret = SM2GenKeyPairSEC(&PubKey_A, &PriKey_A);
    if (SUCCESS != ret)
    {
        goto ErrorExit;
    }

    return 0;

ErrorExit:
    return 1;
}
```

参阅

SM2PUBLICKEYCTX、SM2PRIVATEKEYCTX、 SM2EncryptSEC、

SM2DecryptSEC、 SM2SignSEC、SM2Verify.

3.5.2 SM2EncryptSEC

SM2EncryptSEC 函数使用当前参数中的公钥，对指定的消息数据进行加密运算并输出运算结果至用户指定区域。

```
#include “iscrypt_sm2_sec.h”
unsigned char SM2EncryptSEC(
    SM2PUBLICKEYCTX xdata *pSM2PubKey,        // in
    SM2PLAINTEXT xdata *pPlaintext,           // in
    SM2CIPHERTEXT xdata *pCiphertext          // out
);
```

参数

pSM2PubKey

指向SM2 公钥结构体 **SM2PUBLICKEYCTX**的指针。

pPlaintext

指向SM2 明文结构体 **SM2PLAINTEXT**的指针，该结构体由明文数据指针和数据长度两部分组成。

pCiphertext

指向SM2 密文的结构体 **SM2CIPHERTEXT**的指针，由四部分组成。

返回值

如果函数执行成功，返回值为 SUCCESS。如果失败，错误代码列举如下。

Error Code	Description
IN_DATA_LENGTH_ERROR	明文数据长度错误

注意

1. 从节省 RAM 资源的角度考虑，密文结构体中的 *pbCipher* 可以与明

文结构体中的 *pbData* 共享同一片存储区。这样使用时，当原始数据需要重用的话，则请注意备份。

2. 当前加密的数据长度限定为 1-32 字节。
3. **函数运行时，将使用 1KB 的 PKU RAM。**

快速信息

头文件：声明在 iscrypt_sm_sec2.h。

库文件：使用 iscrypt_sm2_sec.lib。

示例

```
#include <string.h>
#include "iscrypt_sm2_sec.h"
#include "iscrypt_sm2_test.h"

unsigned char xdata space_for_test[1024];

SM2POINT xdata Point;
SM2PUBLICKEYCTX xdata PubKey_A;
SM2PRIVATEKEYCTX xdata PriKey_A;

SM2PLAINTEXT xdata Plain;
SM2CIPHERTEXT xdata Cipher;

char code SM2Plaintext[] = "encryption standard";

char SM2_ENCRYPT_DECRYPT_TEST()
{
    unsigned char ret;

    ////////////////////////////////////////////////////////////////

    // prepare key
```

```
memcpy(space_for_test, SM2PublicKey, 64);
memcpy(space_for_test+64, SM2PrivateKey, 32);

PubKey_A.x = space_for_test;
PubKey_A.y = space_for_test+32;

PriKey_A.d = space_for_test+64;

//////////////////////////////////////////////////////////////

// prepare plaintext
memcpy(space_for_test+0x100, SM2Plaintext, strlen(SM2Plaintext));
Plain.pd = space_for_test+0x100;
Plain.len = strlen(SM2Plaintext);

// prepare ciphertext
Point.x = space_for_test+0x200;
Point.y = space_for_test+0x220;

Cipher.p = &Point;
Cipher.c = space_for_test+0x260;
Cipher.h = space_for_test+0x240;

//
ret = SM2EncryptSEC(&PubKey_A, &Plain, &Cipher);
if (SUCCESS != ret)
{
    goto ErrorExit;
}

//////////////////////////////////////////////////////////////
Plain.pd = space_for_test+0x300;
```

```
    Plain.len = 0x00;

    ret = SM2DecryptSEC(&PriKey_A, &Cipher, &Plain);

    if (SUCCESS != ret)
    {
        goto ErrorExit;
    }

    Plain.pd[Plain.len] = '\0';
    ret = strcmp(SM2Plaintext, Plain.pd);

    if (0 != ret)
    {
        goto ErrorExit;
    }

    return 0;

ErrorExit:
    return 1;
}
```

参阅

SM2PUBLICKEYCTX、SM2PLAINTEXT、SM2CIPHERTEXT、SM2DecryptSEC.

3.5.3 SM2DecryptSEC

SM2DecryptSEC 函数使用当前参数中的私钥，对指定的密文数据进行解密运算并输出运算结果至用户指定区域。

```
#include “iscrypt_sm2_sec.h”
unsigned char SM2DecryptSEC (
    SM2PRIVATEKEYCTX xdata *pSM2PriKey,      // in
    SM2CIPHERTEXT xdata *pCiphertext,        // in
    SM2PLAINTEXT xdata *pPlaintext           // out
);
```

参数

pSM2PriKey

指向SM2 私钥结构体 **SM2PRIVATEKEYCTX**的指针。

pCiphertext

指向SM2 密文结构体 **SM2CIPHERTEXT**的指针。

pPlaintext

指向SM2 明文结构体 **SM2PLAINTEXT**的指针。

返回值

如果函数执行成功，返回值为 SUCCESS。如果失败，错误代码列举如下。

Error Code	Description
INVALID_PARA	密文数据的参数错误
IN_DATA_LENGTH_ERROR	密文数据的长度错误
HASH_ERROR	密文中的哈希值错误

注意

1. 从节省 RAM 资源的角度考虑，密文结构体中的 *pbCipher* 可以与明文结构体中的 *pbData* 共享同一片存储区。

2. 当前加密的数据长度限定为 1-32 字节。

3. **函数运行时，将使用 1KB 的 PKU RAM。**

快速信息

头文件：声明在 iscrypt_sm2_sec.h。

库文件：使用 iscrypt_sm2_sec.lib。

示例

见 SM2EncryptSEC.

参阅

SM2PRIVATEKEYCTX、SM2CIPHERTEXT、SM2PLAINTEXT、SM2EncryptSEC.

3.5.4 SM2SignSEC

SM2SignSEC 函数使用当前参数中的私钥，对输入消息进行签名运算，并生成签名结果存放在指定区域。

```
#include “iscrypt_sm2_sec.h”
unsigned char SM2SignSEC(
    SM2PRIVATEKEYCTX xdata *pSM2PriKey,      // in
    unsigned char xdata *pbDigest,           // in
    SM2SIGNATURE xdata *pSignature           // out
);
```

参数

pSM2PriKey

指向用于签名运算的SM2 私钥结构体的指针 **SM2PRIVATEKEYCTX**。

pbDigest

待签名数据，该数据通常应为消息数据的摘要值。按照《SM2 公钥密码算法标准》中的规定，该值应为$Hash_{256}$（Z_A || M）；其中$Hash_{256}$应选用SM3 算法。

pSignature

消息的摘要的签名值结构体指针 **SM2SIGNATURE**，由两部分组成。

返回值

要求函数必须执行成功，返回值为 **SUCCESS**。

注意

1. **函数运行时，将使用 1KB 的 PKU RAM。**

快速信息

头文件：声明在 iscrypt_sm2_sec.h。

库文件：使用 iscrypt_sm2_sec.lib。

示例

```
#include <string.h>
#include "iscrypt_sm2.h"
#include "iscrypt_sm3.h"
#include "iscrypt_sm2_sec.h"

extern unsigned char xdata space_for_test[1024];

SM2PUBLICKEYCTX xdata PubKey_A;
SM2PRIVATEKEYCTX xdata PriKey_A;

SM2SIGNATURE xdata Signature;

char code SM2Message[] = "message digest";

char SM2_SIGN_VERIFY_TEST()
{
    unsigned char ret;
    unsigned char xdata *tmp_msg;
    unsigned char xdata tmp_msg_len;

    ////////////////////////////////////////////////////////////////
    // prepare key
    memcpy(space_for_test, SM2PublicKey, 64);
    memcpy(space_for_test+64, SM2PrivateKey, 32);

    PubKey_A.x = space_for_test;
    PubKey_A.y = space_for_test+32;

    PriKey_A.d = space_for_test+64;
```

```
////////////////////////////////////////////////////////////////
// precompute step 1: Za
memcpy(space_for_test+0x100, SM2KeyExchangeParam+384, 16);

IDInfo_A.pd = space_for_test+0x100;
IDInfo_A.len = 16;

ret = SM2GetZ(&PubKey_A, &IDInfo_A, space_for_test+0x100);

if (SUCCESS != ret)
{
    goto ErrorExit;
}

tmp_msg = space_for_test+0x100;
tmp_msg_len = 32;

// M~ = ZA || M
memcpy(space_for_test+0x120, SM2Message, strlen(SM2Message));
tmp_msg_len += strlen(SM2Message);


// precompute step 2: e = Hash(Za || M)
SM3Init();

while (tmp_msg_len >= 64)
{
    SM3Update(tmp_msg);

    tmp_msg += 64;
    tmp_msg_len -= 64;
```

```
    }

    SM3Final(tmp_msg, tmp_msg_len, space_for_test+0x100);

    // SM2 sign
    Signature.r = space_for_test+0x200;
    Signature.s = space_for_test+0x220;

    ret = SM2SignSEC(&PriKey_A, space_for_test+0x100, &Signature);
    if (SUCCESS != ret)
    {
        goto ErrorExit;
    }

    ////////////////////////////////////////////////////////////////

    ret = SM2Verify(&PubKey_A, space_for_test+0x100, &Signature);

    if (SUCCESS != ret)
    {
        goto ErrorExit;
    }

    return 0;

ErrorExit:
    return 1;
}
```

参阅

SM2PRIVATEKEYCTX、SM2SIGNATURE、SM2Verify.

3.5.5 SM2Verify

SM2Verify 函数使用当前参数中的 SM2 公钥，对输入消息及其签名值，进行验证。

```
#include “iscrypt_sm2_sec.h”
unsigned char SM2Verify(
    SM2PUBLICKEYCTX xdata *pSM2PubKey,// in
    unsigned char xdata *pbDigest,                  // in
    SM2SIGNATURE xdata *pSignature                  // in
);
```

参数

pSM2PubKey

指向用于签名运算的SM2 公钥结构体指针 **SM2PUBLICKEYCTX**。

pbDigest

待签名数据，该数据通常应为消息数据的摘要值。按照《SM2 公钥密码算法标准》中的规定，该值应为$Hash_{256}$（Z_A || M）；其中$Hash_{256}$应选用 SM3 算法。

pSignature

指向SM2 签名结构体 **SM2SIGNATURE**的指针，表示待验证消息摘要的签名值。

返回值

如果函数执行成功，返回值为 SUCCESS。如果失败，返回值为 FAIL。

注意

1. **函数运行时，将使用 1KB 的 PKU RAM。**

快速信息

头文件：声明在 iscrypt_sm2_sec.h。

库文件：使用 iscrypt_sm2_sec.lib。

示例

请参见 SM2SignSEC。

参阅

SM2PUBLICKEYCTX、SM2SIGNATURE、SM2SignSEC.

3.5.6 SM2KeyExchange

SM2KeyExchange 函数使用当前参数进行密钥交换运算，并生成交换密钥输出至用户指定区域。

```
#include “iscrypt_sm2_sec.h”
#include “iscrypt_sm3.h”
unsigned char SM2KeyExchange(
SM2KEYEXCHANGEPARAM xdata *pSM2KeyExchangeParam,      //in
    unsigned char xdata *pbKey                          // out
);
```

参数

pSM2KeyExchangeParam

指向用于传送密钥交换所需要的具体参数的密钥交换结构体 **SM2KEYEXCHANGEPARAM**的指针。

pbKey

密钥交换运算成功后，生成的指定长度的密钥。

返回值

如果函数执行成功，返回值为 SUCCESS。如果失败，返回 FAIL。

注意

1. 期望交换密钥的长度 *bKeyLenExpected* 通常选用 8、16、32 三种长度，同时支持不大于 128 字节的任意字节长的交换密钥生成。
2. **函数运行时，将使用 1KB 的 PKU RAM。**

快速信息

头文件：声明在 iscrypt_sm2_sec.h。

库文件：使用 iscrypt_sm2_sec.lib。

示例

```
#include <string.h>
#include "iscrypt_sm2_sec.h"
#include "iscrypt_sm2_test.h"

extern unsigned char xdata space_for_test[1024];

SM2PUBLICKEYCTX xdata PubKey_A;
SM2PUBLICKEYCTX xdata PubKey_B;
SM2PUBLICKEYCTX xdata TempPubKey_A;
SM2PUBLICKEYCTX xdata TempPubKey_B;

SM2PRIVATEKEYCTX xdata PriKey_A;
SM2PRIVATEKEYCTX xdata PriKey_B;
SM2PRIVATEKEYCTX xdata TempPriKey_A;
SM2PRIVATEKEYCTX xdata TempPriKey_B;

IDINFO xdata IDInfo_A, xdata IDInfo_B;

SM2KEYEXCHANGEPARAM xdata KeyExchangeParam;

char SM2_KEY_EXCHANGE_TEST()
{
    unsigned char ret;

    unsigned char xdata *pZa, xdata *pZb;

    // Just as alias
    unsigned char xdata *pbKeyAB, xdata *pbKeyBA;

    ////////////////////////////////////////////////////////////////
```

```
memcpy(space_for_test, SM2KeyExchangeParam, sizeof (SM2KeyExchangeParam));

//
TempPubKey_A.x = space_for_test;
TempPubKey_A.y = space_for_test+32;
TempPriKey_A.d = space_for_test+64;

PubKey_A.x = space_for_test+96;
PubKey_A.y = space_for_test+128;
PriKey_A.d = space_for_test+160;

TempPubKey_B.x = space_for_test+192;
TempPubKey_B.y = space_for_test+224;
TempPriKey_B.d = space_for_test+256;

PubKey_B.x = space_for_test+288;
PubKey_B.y = space_for_test+320;
PriKey_B.d = space_for_test+352;

IDInfo_A.pd = space_for_test+384;
IDInfo_A.len = 16;

IDInfo_B.pd = space_for_test+400;
IDInfo_B.len = 16;

// precompute Za
pZa = space_for_test+432;

ret = SM2GetZ(&PubKey_A, &IDInfo_A, pZa);
if (SUCCESS != ret)
{
    goto ErrorExit;
```

```
}

// precompute Zb
pZb = space_for_test+464;

ret = SM2GetZ(&PubKey_B, &IDInfo_B, pZb);
if (SUCCESS != ret)
{
    goto ErrorExit;
}

// A is initiator, and B is responder

//////////////////////////////////////////////////////////////
memset(&KeyExchangeParam, 0x00, sizeof (KeyExchangeParam));

KeyExchangeParam.role = INITIATOR;
KeyExchangeParam.keyLenExpected = 128;
KeyExchangeParam.pSelfTempPubKey = &TempPubKey_A;
KeyExchangeParam.pSelfTempPriKey = &TempPriKey_A;
KeyExchangeParam.pSelfPubKey = &PubKey_A;
KeyExchangeParam.pSelfPriKey = &PriKey_A;
KeyExchangeParam.pOtherTempPubKey = &TempPubKey_B;
KeyExchangeParam.pOtherPubKey = &PubKey_B;
KeyExchangeParam.pSelfZ = pZa;
KeyExchangeParam.pOtherZ = pZb;

pbKeyAB = space_for_test + 0x200;

ret = SM2KeyExchange(&KeyExchangeParam, pbKeyAB);
if (SUCCESS != ret)
{
```

```
    goto ErrorExit;
}

//////////////////////////////////////////////////////////////
memset(&KeyExchangeParam, 0x00, sizeof (KeyExchangeParam));

KeyExchangeParam.role = RESPONDER;
KeyExchangeParam.keyLenExpected = 128;
KeyExchangeParam.pSelfTempPubKey = &TempPubKey_B;
KeyExchangeParam.pSelfTempPriKey = &TempPriKey_B;
KeyExchangeParam.pSelfPubKey = &PubKey_B;
KeyExchangeParam.pSelfPriKey = &PriKey_B;
KeyExchangeParam.pOtherTempPubKey = &TempPubKey_A;
KeyExchangeParam.pOtherPubKey = &PubKey_A;
KeyExchangeParam.pSelfZ = pZb;
KeyExchangeParam.pOtherZ = pZa;

pbKeyBA = space_for_test + 0x300;

ret = SM2KeyExchange(&KeyExchangeParam, pbKeyBA);
if (SUCCESS != ret)
{
    goto ErrorExit;
}

//////////////////////////////////////////////////////////////
ret = memcmp(pbKeyAB, pbKeyBA, 128);
if (0 != ret)
{
    goto ErrorExit;
}
```

```
    return 0;

ErrorExit:
    return 1;
}
```

参阅

SM2KEYEXCHANGEPARAM、SM2GetZ.

3.5.7 SM2GetZ

SM2GetZ 函数生成签名验证运算和密钥协商运算中需要使用的数据。

```
#include “iscrypt_sm2_sec.h”
unsigned char SM2GetZ(
    SM2PUBLICKEYCTX xdata *pSM2PubKey,// in
    IDINFO xdata *pIDInfo,                    // in
    unsigned char xdata *pbZValue             // out
);
```

参数

pSM2PubKey

SM2 公钥结构体指针 **SM2PUBLICKEYCTX**。

pIDInfo

用户的ID信息结构体 **IDINFO**指针，分成两部分。

pZValue

用于存放生成签名认证运算和密钥协商运算中需要使用的数据的地址。

返回值

如果函数执行成功，返回值为 SUCCESS。如果失败，错误代码列举如下。

Error Code	Description
IN_DATA_LENGTH_ERROR	ID 的长度错误。

注意

1. 按照《SM2 椭圆曲线公钥密码算法》，使用 **SM2SignSEC** 和 **SM2Verify**，以及 **SM2KeyExchange**等函数时，需要用户的身份信息的杂凑值的参与。其中用户身份信息的杂凑值计算方式为

$Z_A=H_{256}(ENTL_A||ID_A||a||b||x_G||y_G||x_A||y_A)$。

2. *pIDInfo* 中 ID 的长度，限制范围 1~32 个字节。

3. **函数运行时，将使用 1KB 的 PKU RAM。**

快速信息

头文件：声明在 iscrypt_sm2_sec.h。

库文件：使用 iscrypt_sm2_sec.lib。

示例

```
#include <string.h>
#include "iscrypt_sm2_sec.h"
#include "iscrypt_sm2_test.h"

extern unsigned char xdata space_for_test[1024];

SM2PUBLICKEYCTX xdata PubKey_A;
IDINFO xdata IDInfo_A;

char SM2_KEY_GET_Z_TEST()
{
    unsigned char ret;
    unsigned char xdata *pbZ;

    ////////////////////////////////////////////////////////////////
    memcpy(space_for_test, SM2KeyExchangeParam, sizeof (SM2KeyExchangeParam));

    PubKey_A.x = space_for_test+96;
    PubKey_A.y = space_for_test+128;
    PriKey_A.d = space_for_test+160;


    IDInfo_A.pd = space_for_test+384;
```

```
    IDInfo_A.len = 16;

    pbZ = space_for_test+0x200;

    ret = SM2GetZ(&PubKey_A, &IDInfo_A, pbZ);
    if (SUCCESS != ret)
    {
        goto ErrorExit;
    }

    return 0;

ErrorExit:
    return 1;
}
```

参阅

SM2PUBLICKEYCTX、IDINFO.

3.6 SM3

3.6.1 SM3Init

SM3Init 函数用于完成基于 SM3 算法的初始化操作。

#include “iscrypt_sm3.h”

void SM3Init();

参数

无

返回值

无

注意

1. **函数运行时，将使用 1KB 的 PKU RAM。**

快速信息

头文件：声明在 iscrypt_sm3.h。

库文件：使用 iscrypt_sm3.lib。

示例

请参见 **SM3Final**一节。

参阅

SM3Update，SM3Final.

3.6.2 SM3Update

SM3Update 用以对一个 SM3 的消息分组中间块儿进行处理。

```
#include “iscrypt_sm3.h”
void SM3Update(
    unsigned char xdata *pbMessage      //in
);
```

参数

pbMessage

需要被哈希处理的消息分组，对于 SM3 算法来说为 64 字节为一组。

返回值

无

注意

1. **函数运行时，将使用 1KB 的 PKU RAM。**

快速信息

头文件：声明在 iscrypt_sm3.h。

库文件：使用 iscrypt_sm3.lib。

示例

请参见 **SM3Final**一节。

参阅

SM3Init、SM3Final.

3.6.3 SM3Final

SM3Final 用以对一个 SM3 的末块儿消息分组进行处理。

```
#include “iscrypt_sm3.h”
unsigned char SM3Final (
    unsigned char xdata *pbMessage      //  in
    unsigned char bLength,              //  in
    unsigned char xdata *pbDigest       //  out
);
```

参数

pbMessage

需要被哈希处理的末块儿消息分组。

bLength

末块儿消息分组的字节长度，小于 64，可以为 0。

pbDigest

SM3 处理结束后，摘要的存储区地址。SM3 摘要长度为 32 字节（256 比特）。

返回值

如果函数执行成功，返回值为 SUCCESS。如果失败，错误代码列举如下。

Error Code	Description
HASH_OBJECT_ERROR	哈希对象错误

注意

1. **函数运行时，将使用 1KB 的 PKU RAM。**

快速信息

头文件：声明在 iscrypt_sm3.h。

库文件：使用 iscrypt_sm3.lib。

示例

```
#include <string.h>
#include "iscrypt_sm3.h"
#include "iscrypt_sm3_test.h"

unsigned char xdata space_for_test[1024];

char SM3_TEST()
{
    unsigned char xdata ret;
    memcpy(space_for_test, SM3_VECTOR, 64);

    ////////////////////////////////////////////////////////////////
    SM3Init();

    ret = SM3Final(space_for_test, 3, space_for_test+0x200);
    if (SUCCESS != ret)
    {
        goto ErrorExit;
    }

    ret = memcmp(space_for_test+0x200, DIGEST_SM3_3, 32);
    if (0 != ret)
    {
        goto ErrorExit;
    }

    ////////////////////////////////////////////////////////////////
```

```
SM3Init();

ret = SM3Final(space_for_test, 56, space_for_test+0x200);
if (SUCCESS != ret)
{
    goto ErrorExit;
}

ret = memcmp(space_for_test+0x200, DIGEST_SM3_56, 32);
if (0 != ret)
{
    goto ErrorExit;
}

//////////////////////////////////////////////////////////////
SM3Init();

SM3Update(space_for_test);

ret = SM3Final(space_for_test, 0, space_for_test+0x200);
if (SUCCESS != ret)
{
    goto ErrorExit;
}

ret = memcmp(space_for_test+0x200, DIGEST_SM3_64, 32);
if (0 != ret)
{
    goto ErrorExit;
}

//////////////////////////////////////////////////////////////
```

```
    SM3Init();

    SM3Update(space_for_test);

    SM3Update(space_for_test);

    ret = SM3Final(space_for_test, 0, space_for_test+0x200);
    if (SUCCESS != ret)
    {
        goto ErrorExit;
    }

    ret = memcmp(space_for_test+0x200, DIGEST_SM3_128, 32);
    if (0 != ret)
    {
        goto ErrorExit;
    }

    return 0;
ErrorExit:
    return 1;
}
```

参阅

SM3Init、SM3Update.

3.7 SM4

3.7.1 SM4CryptSEC

SM4CryptSEC 函数用于提供可抵御 SCA 和 FA 攻击的 SM4 算法的加解密功能操作。

#include “iscrypt_sm4_sec.h”

unsigned char SM4Crypt SEC(

**BLOCKCIPHERPARAM xdata ***_pBlockCipherParam_ // in,out

);

参数

pBlockCipherParam

分组密码算法结构体指针，由以下部分组成。

unsigned char *bOpMode*

操作类型选择，当前支持以下几种选择。

Value	**Description**
SYM_ECB	ECB 模式
SYM_CBC	CBC 模式
SYM_ENCRYPT	加密操作
SYM_DECRYPT	解密操作

unsigned char xdata **pbKey*

指向密钥的存储区，对于 SM4 为 16 字节。

unsigned char xdata **pbInput*

输入数据的指针。

unsigned char *bTotalBlock*

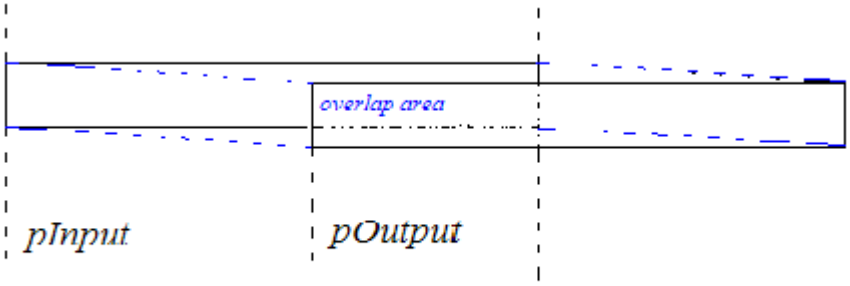
输入数据的分组数。对于 SM4 为以 SM4_BLOCK_SIZE（定义在头

文件中）个字节为一分组。注意，当该参数为 0x00 时，算法库按照 256 个分组执行。

unsigned char xdata **pbOutput*

输出数据的存放地址。对于 SM4 来说，ECB 模式进行操作时，*pbInput* 可以与 *pbOutput* 相同；

同时要求，*pbOutput* 指向的存储区域不可与 *pbInput* 指向的存储区域有前向重叠。下图所示情况，在 SM4 进行 ECB 模式的加解密时是不允许的，否则不保证处理结果的正确性。



unsigned char xdata **pbReserve*

保留指针，此处作为安全级别输入项，指向外部用户定义的安全级别参数。当前支持的安全级别为：

Value	Description
D_LOW_LEVEL_S	低安全级别
D_MEDIUM_LEVEL_S	中安全级别
D_HIGH_LEVEL_S	高安全级别

返回值

如果函数执行成功，返回值为 SUCCESS。如果失败，错误代码列举如下。

Error Code	Description
NOT_SUPPORT_YET_ERROR	暂不支持
FAIL	运行错误

注意

1. 由于参数 *bTotalBlock* 的限制，SM4CryptSEC 函数执行一次的最大吞吐量为 256 个分组，共计 4KB 的数据量。
2. **由于防御措施的实施，需要使用 1KB 的 PKU RAM。**
3. **应用环境中，如果外部密钥每次都是变化的，则可不使用提供的 SM4CryptSEC 函数。在应用环境中，对密钥有使用限制、密钥生命周期很短或密钥频繁更换的情况下，使用基本的 SM4 加解密函数即可，此时不失安全性。**
4. 使用时，需要包含 iscrypt_random.lib。

快速信息

头文件：声明在 iscrypt_sm4_sec.h。

库文件：使用 iscrypt_sm4_sec.lib。

示例

```
#include <string.h>
#include "iscrypt_sm4_sec.h"

extern unsigned char xdata space_for_test[1024];
extern BLOCKCIPHERPARAM xdata BlockCipherParam;

char SM4_TEST()
{
    unsigned char ret;
    unsigned char xdata sl = D_HIGH_LEVEL_S;

    memcpy(space_for_test, BLOCK_CRYPT_KEY, SM4_BLOCK_SIZE);
    memcpy(space_for_test+SM4_BLOCK_SIZE, BLOCK_CRYPT_IV,
SM4_BLOCK_SIZE);

    ////////////////////////////////////////////////////////////
    memcpy(space_for_test+2*SM4_BLOCK_SIZE, BLOCK_CRYPT_PLAINTEXT,
```

```
5*SM4_BLOCK_SIZE);


    BlockCipherParam.bOpMode = SYM_ECB | SYM_ENCRYPT;
    BlockCipherParam.pbKey = space_for_test;
    BlockCipherParam.pbIV = space_for_test+SM4_BLOCK_SIZE;

    BlockCipherParam.pbInput = space_for_test+2*SM4_BLOCK_SIZE;
    BlockCipherParam.bTotalBlock = 5;
    BlockCipherParam.pbOutput = space_for_test+2*SM4_BLOCK_SIZE;
    BlockCipherParam.pbReserve = &sl;

    ret = SM4CryptSEC(&BlockCipherParam);
    if (SUCCESS != ret)
    {
        goto ErrorExit;
    }

    ret                    =                    memcmp(space_for_test+2*SM4_BLOCK_SIZE,
BLOCK_CRYPT_CIPHERTEXT_SM4_ECB, 5*SM4_BLOCK_SIZE);

    if (0 != ret)
    {
        goto ErrorExit;
    }

    ////////////////////////////////////////////////////////////////
    memcpy(space_for_test+2*SM4_BLOCK_SIZE,
BLOCK_CRYPT_CIPHERTEXT_SM4_ECB, 5*SM4_BLOCK_SIZE);


    BlockCipherParam.bOpMode = SYM_ECB | SYM_DECRYPT;
```

```
    BlockCipherParam.pbKey = space_for_test;
    BlockCipherParam.pbIV = space_for_test+SM4_BLOCK_SIZE;

    BlockCipherParam.pbInput = space_for_test+2*SM4_BLOCK_SIZE;
    BlockCipherParam.bTotalBlock = 5;
    BlockCipherParam.pbOutput = space_for_test+2*SM4_BLOCK_SIZE;
    BlockCipherParam.pbReserve = &sl;

    ret = SM4CryptSEC(&BlockCipherParam);
    if (SUCCESS != ret)
    {
        goto ErrorExit;
    }

    ret = memcmp(space_for_test+2*SM4_BLOCK_SIZE, BLOCK_CRYPT_PLAINTEXT,
5*SM4_BLOCK_SIZE);

    if (0 != ret)
    {
        goto ErrorExit;
    }
    return 0;

ErrorExit:
    return 1;
}
```

参阅

BLOCKCIPHERPARAM、SM1CryptSEC.

3.8 RNG

3.8.1 GenRandom

生成指定长度的随机数。

```
#include “iscrypt_random.h”
void GenRandom(
    unsigned short bRandomLen,          // in
    unsigned char xdata *pbRandom       // out
);
```

参数

bRandomLen

需要生成随机数的字节长度。

pbRandom

存储随机数的缓冲区地址。

返回值

无

注意

无

示例

```
#include "iscrypt_random.h"

void main()
{
 unsigned char xdata space_for_test[256] = {0x00};
 GenRandom(16, space_for_test);
 while(1) ;
```

```
}
```

参阅

无

3.8.2 GenRandomSEC

生成指定长度的随机数，每次运行可保证随机数均在软硬件资源正常运转的情况下产生。

```
#include “iscrypt_random_sec.h”
unsigned char GenRandomSEC(
    unsigned short bRandomLen,          //  in
    unsigned char xdata *pbRandom       //  out
);
```

参数

bRandomLen

指定的需要生成随机数长度，由其数据类型可知，长度应小于65536。

pbRandom

存储随机数的缓冲区地址。

返回值

函数执行成功则返回 SUCCESS，如错误则返回 FAIL。

注意

1. 函数运行时，会内部调用 RNG_OL_TOT_Test 函数。
2. 调用库 iscrypt_random_sec.lib 时，同时需要添加随机数库 iscrypt_random.lib。

快速信息

头文件：声明在 iscrypt_random_sec.h。

库文件：使用 iscrypt_random_sec.lib。

示例

```
#include "iscrypt_random_sec.h"

int main()
{
    unsigned char xdata space_for_test[64] = {0x00};

    GenRandomSEC(16, space_for_test);

    return 0;
}
```

参阅

RNG_StartupTest、RNG_OL_TOT_Test.

3.8.3 RNG_StartupTest

随机数相关软硬件资源的上电测试，由芯片每次上电时调用。

#include “iscrypt_random_sec.h”

unsigned char RNG_StartupTest (void);

参数

无。

返回值

函数执行成功则返回 SUCCESS，如错误则返回 FAIL。

注意

无。

快速信息

头文件：声明在 iscrypt_random_sec.h。

库文件：使用 iscrypt_random_sec.lib。

示例

无。

参阅

GenRandomSEC、RNG_OL_TOT_Test.

3.8.4 RNG_OL_TOT_Test

随机数相关软硬件资源的在线测试，由芯片每次进行随机数生成时内部调用，外部用户可根据需要随时调用。

#include “iscrypt_random_sec.h”

unsigned char RNG_OL_TOT_Test(void);

参数

无。

返回值

函数执行成功则返回 SUCCESS，如错误则返回 FAIL。

注意

无。

快速信息

头文件：声明在 iscrypt_random_sec.h。

库文件：使用 iscrypt_random_sec.lib。

示例

无。

参阅

GenRandomSEC、RNG_StartupTest.

3.8.5 GetFactoryCode

读取芯片工厂码。

```
#include “iscrypt_random.h”
void GetFactoryCode(
    unsigned char xdata *pbFtyCode       // out
);
```

参数

pbFtyCode

存储芯片序列号的缓冲区地址，芯片序列号只有 13 个字节。

返回值

无

注意

无

示例

```
#include "iscrypt_random.h"

void main()
{
 unsigned char xdata space_for_test[8];
 GetFactoryCode(space_for_test);
 while(1) ;
}
```

参阅

无

3.9 其他

3.9.1 HASHSetBkpt

```
#include "iscrypt_hash_bkpt_rst.h"
unsigned char HASHSetBkpt(
    unsigned char ChoiceOfHASH,                 // in
    unsigned char xdata *pbRareIV,              // out
    unsigned char xdata *pLengthHasProcessed    // out
);
```

参数

ChoiceOfHASH

HASH 算法选择，支持以下几种。

Value	Description
SHA1	SHA-1
SHA256	SHA-256
SM3	国标 SM3 哈希算法

pbRareIV

当前 HASH 算法的中间向量，不同 HASH 算法的中间向量的长度不同。

pMsgLengthHasProcessed

当前 HASH 算法已经处理过的消息数据的长度，不同 HASH 算法的长度表示范围和所占用的空间大小不同。

返回值

无

注意

针对不同的 HASH 算法，对其进行中断续处理时，需要备份的数据的长度不同。因此，函数调用者需保证已经申请了足够的空间进行这些内容的

保护。

示例

请参见 **HASHRestore**的示例

参阅

HASHRestore

3.9.1 HASHRestore

```
#include "iscrypt_hash_bkpt_rst.h"
unsigned char HASHRestore(
    unsigned char ChoiceOfHASH,                 // in
    unsigned char xdata *pbRareIV,              // in
    unsigned char xdata *pLengthHasProcessed    // in
);
```

参数

ChoiceOfHASH

HASH 算法选择，支持以下几种。

Value	**Description**
SHA1	SHA-1
SHA256	SHA-256
SM3	国标 SM3 哈希算法

pbRareIV

当前 HASH 算法的中间向量，不同 HASH 算法的中间向量的长度不同。

pMsgLengthHasProcessed

当前 HASH 算法已经处理过的消息数据的长度，不同 HASH 算法的长度表示范围和所占用的空间大小不同。

返回值

无

注意

无

示例

```
#include <string.h>
#include "iscrypt_sha.h"
#include “iscrypt_sm3.h”
#include "iscrypt_hash_bkpt_rst.h"
unsigned char xdata space_for_test[1024];

char HASH_BKPT_RST_TEST()
{
    unsigned char xdata ret;
    unsigned char xdata sha_backup_iv[64];
    unsigned char xdata sha_backup_msg_length[16];

    memcpy(space_for_test, SHA_VECTOR, 64);

    ////////////////////////////////////////////////////////////
    //SHA1
    SHA1Init();

    SHA1Update(space_for_test);
    ret = HASHSetBkpt(SHA1, sha_backup_iv, sha_backup_msg_length);

    // Do other things
    SHA256Init();

    //
    ret = HASHRestore(SHA1, sha_backup_iv, sha_backup_msg_length);
    SHA1Update(space_for_test);

    ret = SHA1Final(space_for_test, 0, space_for_test+0x200);
    if (SUCCESS != ret)
    {
        goto ErrorExit;
```

```
    }

    ret = memcmp(space_for_test+0x200, DIGEST_SHA1_128, 20);
    if (0 != ret)
    {
        goto ErrorExit;
    }

    return SUCCESS;

ErrorExit:
    return FAIL;
}
```

参阅

HASHSetBkpt

附　录

附录 1　结构体定义说明

iscrypt_sec 中提供的算法函数使用过程中，需要定义相应的算法结构体，以下是需要被使用的结构体定义。

表 1　iscrypt_sec 算法结构体列表

算法类别	结构体名	使用函数
对称算法	BLOCKCIPHERPARAM	DESCryptSEC TDESCryptSEC SM1CryptSEC SM4CryptSEC
非对称算法 RSA	RSAPUBLICKEYCTX	RSAGenKeyPair RSAEncrypt

		RSAVerify
	RSAPRIVATEKEYCTX	RSAGenKeyPair RSADecryptSEC RSASignSEC
	RSAPRIVATEKEYCRT	RSAGenKeyPair RSADecryptSEC RSASignSEC
	RSAPRIVATEKEYSTD	RSAGenKeyPair RSADecryptSEC RSASignSEC
	RSAKEYCTX	保留
非对称算法 SM2	SM2PUBLICKEYCTX	SM2GenKeyPairSEC SM2EncryptSEC SM2Verify
	SM2POINT	SM2EncryptSEC SM2DecryptSEC
	SM2PRIVATEKEYCTX	SM2GenKeyPairSEC SM2DecryptSEC SM2SignSEC
	SM2PLAINTEXT	SM2EncryptSEC SM2DecryptSEC
	SM2CIPHERTEXT	SM2EncryptSEC SM2DecryptSEC
	SM2SIGNATURE	SM2SignSEC SM2Verify
	SM2KEYEXCHANGEPARAM	SM2KeyExchange
	IDINFO	SM2GetZ
保留	BIGINTEGER	保留

BLOCKCIPHERPARAM

```
typedef struct Struct_BLOCKCIPHERPARAM
{
    unsigned char bOpMode;                  //  in
    unsigned char xdata *pbKey;             //  in
    unsigned char xdata *pbIV;              //  in
    unsigned char xdata *pbInput;           //  in
    unsigned char bTotalBlock;              //  in
    unsigned char xdata *pbOutput;          //  out
    unsigned char xdata *pbReserve;         //  unused
}BLOCKCIPHERPARAM, *PBLOCKCIPHERPARAM;
```

成员

bOpMode

操作模式类型选择，当前支持以下几种选择。

Value	Description
SYM_ECB	ECB 模式
SYM_CBC	CBC 模式
SYM_CFB	CFB 模式
SYM_OFB	OFB 模式
SYM_CTR	CTR 模式
SYM_ENCRYPT	加密操作
SYM_DECRYPT	解密操作

pbKey

指向所使用的对称加密算法的密钥。

pbIV

如果对称算法使用 CBC/CFB 等模式进行加解密，则需要对此参数赋值，指向初始向量。

pbInput

输入数据的存放地址，加密时指向明文数据，解密时指向密文数据。

bTotalBlock

输入数据的分组数，分组长度由当前选用的对称算法决定，当该参数为 0x00 时，算法库内部按照 256 个分组执行。

pbOutput

输出数据的存放地址，加密时指向密文的存放地址，解密时指向明文的存放地址。

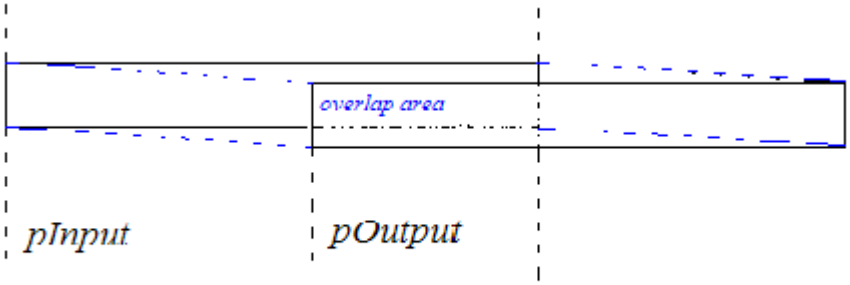
pbReserve

保留指针，留待以后扩展使用。

注意

对于对称算法来说，ECB 模式进行操作时，*pbInput* 可以与 *pbOutput* 相同；CBC 模式操作时，二者不能相同。

同时要求，*pbOutput* 指向的存储区域不可与 *pbInput* 指向的存储区域有前向重叠。下图所示情况，在对称算法进行 ECB 和 CBC 模式的加解密时是不允许的，否则不保证处理结果的正确性。



快速信息

头文件：声明在 iscrypt_symmetric.h

参阅

DESCryptSEC、TDESCryptSEC.

RSAPRIVATEKEYCRT

```
typedef struct Struct_RSAPrivateKeyCRT
{
    unsigned char xdata *p;
    unsigned char xdata *q;
    unsigned char xdata *dP;
    unsigned char xdata *dQ;
    unsigned char xdata *qInv;
}RSAPRIVATEKEYCRT;
```

成员

p

使用 CRT 的 RSA 私钥形式中的素数 *p*。

q

使用 CRT 的 RSA 私钥形式中的素数 *q*。

dP

使用 CRT 的 RSA 私钥形式中素数 *p* 的 CRT 指数。

dQ

使用 CRT 的 RSA 私钥形式中素数 *q* 的 CRT 指数。

qInv

使用 CRT 的 RSA 私钥形式中的 CRT 协因子。

注意

由于结构体成员均为指针，因此存储空间需要外部调用者事先分配好，分配空间的大小需要根据当前使用的 RSA 位长决定。一般地，设所选 RSA 位长为 *n* 比特（指的是 RSA 的模长），则 CRT 的 RSA 私钥形式中的各成员所需的存储空间分别为 *n*/2 比特。

快速信息

头文件：声明在 iscrypt_rsa.h

参阅

RSAPRIVATEKEYSTD、RSAPRIVATEKEYCTX、RSADecryptSEC、RSASignSEC.

RSAPRIVATEKEYSTD

```
typedef struct Struct_RSAPrivateKeySTD
{
    unsigned char xdata *d;
    unsigned char xdata *n;
}RSAPRIVATEKEYSTD;
```

成员

d

使用标准的 RSA 私钥形式中的私钥指数 *d*。

n

使用标准的 RSA 私钥形式中的模数 *n*。

注意

由于结构体成员均为指针，因此存储空间需要外部调用者事先分配好，分配空间的大小需要根据当前使用的 RSA 位长决定。一般地，设所选 RSA 位长为 *n* 比特（指的是 RSA 的模长），则标准的 RSA 私钥形式中的各成员所需的存储空间分别为 *n* 比特。

快速信息

头文件：声明在 iscrypt_rsa.h

参阅

RSAPRIVATEKEYCRT、RSAPRIVATEKEYCTX、RSADecryptSEC、RSASignSEC.

RSAPRIVATEKEYCTX

#define RSAPRIVATEKEYCTX void

该定义是为统一 RSA 的相关函数参数定义时所设置，并非单独的结构体，可根据所使用的 RSA 的模式查看相应的私钥形式。

参阅

RSAKEYCTX 、RSAPRIVATEKEYCRT、RSADecryptSEC、RSASignSEC.

RSAPUBLICKEYCTX

```
typedef struct Struct_RSAPublicKey
{
    unsigned char xdata *e;
    unsigned char xdata *n;
}RSAPUBLICKEYCTX;
```

成员

e

RSA 公钥中的指数 *e*。

n

RSA 公钥中的模数 *n*。

注意

由于结构体成员均为指针，因此存储空间需要外部调用者事先分配好，分配空间的大小需要根据当前使用的 RSA 位长决定。

公钥指数 *e* 的存储空间需要四个字节，需事先指定好，通常推荐的公钥指数选取 *e*=65537。

快速信息

头文件：声明在 iscrypt_rsa.h

参阅

RSAKEYCTX。

RSAKEYCTX

```
typedef struct Struct_RSAKey
{
    RSAPUBLICKEYCTX xdata *pRSAPubKey;
    RSAPRIVATEKEYCTX xdata *pRSAPriKey;
}RSAKEYCTX;
```

成员

pRSAPubKey

指向 RSA 公钥结构体 RSAPUBLICKEYCTX。

pRSAPriKey

指向 RSA 私钥结构体 RSAPRIVATEKEYCTX。

快速信息

头文件：声明在 iscrypt_rsa.h

参阅

RSAPRIVATEKEYCTX、RSAPRIVATEKEYCRT、RSAPRIVATEKEYCTX、RSADecryptSEC、RSASignSEC.

SM2PUBLICKEYCTX

```
typedef struct Struct_SM2_PubKey
{
    unsigned char xdata *x;
    unsigned char xdata *y;
}SM2PUBLICKEYCTX, SM2POINT;
```

成员

x

SM2 公钥中的 *x* 坐标。

y

SM2 公钥中的 *y* 坐标。

注意

由于结构体成员均为指针，因此存储空间需要外部调用者事先分配好，分配空间的大小需要根据当前使用的曲线位长决定。SM2 使用的 256 位的素域曲线，因此各成员变量所需的存储空间为 256 比特。

快速信息

头文件：声明在 iscrypt_sm2.h

参阅

SM2GenKeyPairSEC、SM2EncryptSEC、SM2SignSEC、SM2GetZ.

SM2PRIVATEKEYCTX

```
typedef struct Struct_SM2_PriKey
{
    unsigned char xdata *d;
}SM2PRIVATEKEYCTX;
```

成员

d

SM2 私钥标量 *d*。

注意

由于结构体成员均为指针，因此存储空间需要外部调用者事先分配好，分配空间的大小需要根据当前使用的曲线位长决定。SM2 使用的 256 位的素域曲线，因此各成员变量所需的存储空间为 256 比特。

快速信息

头文件：声明在 iscrypt_sm2.h

参阅

SM2GenKeyPairSEC、SM2DecryptSEC、SM2SignSEC.

SM2PLAINTEXT

```
typedef struct Struct_BigInteger
{
    unsigned char xdata *pd;
    unsigned char len;
}BIGINTEGER, SM2PLAINTEXT, IDINFO;
```

成员

pd

指向明文数据。

len

明文数据的长度。

注意

由于结构体成员均为指针，因此存储空间需要外部调用者事先分配好.

该结构体用于 SM2EncryptSEC和 SM2DecryptSEC函数中，仅针对 32 字节的数据的加解密。

快速信息

头文件：声明在 iscrypt_sm2.h

参阅

SM2EncryptSEC、SM2DecryptSEC、SM2GetZ.

SM2CIPHERTEXT

```
typedef struct Struct_SM2_Ciphertext
{
    SM2POINT xdata *p;
    unsigned char xdata *c;
    unsigned char xdata *h;
    unsigned char clen;
}SM2CIPHERTEXT;
```

成员

p

指向 SM2 曲线上一点的结构体的指针，其形式与 SM2 公钥结构体一致。

c

密文数据的存储地址。

h

指向校验值。

clen

密文数据的长度，即成员 ***c*** 的长度。

注意

由于结构体成员均为指针，因此存储空间需要外部调用者事先分配好，分配空间的大小需要根据当前使用的曲线位长决定。SM2 使用的 256 位的素域曲线，因此各成员变量所需的存储空间为 256 比特。

该结构体用于 SM2EncryptSEC和 SM2DecryptSEC函数中，仅针对 32 字节的数据的加解密。

快速信息

头文件：声明在 iscrypt_sm2.h

参阅

SM2EncryptSEC、SM2DecryptSEC、SM2POINT.

SM2SIGNATURE

```
typedef struct Struct_SM2_Signature
{
    unsigned char xdata *r;
    unsigned char xdata *s;
}SM2SIGNATURE;
```

成员

r

指向签名值 *r*。

s

指向签名值 *s*。

注意

由于结构体成员均为指针，因此存储空间需要外部调用者事先分配好，分配空间的大小需要根据当前使用的曲线位长决定。SM2 使用的 256 位的素域曲线，因此各成员变量所需的存储空间为 256 比特。

快速信息

头文件：声明在 iscrypt_sm2.h

参阅

SM2SignSEC、SM2Verify.

SM2KEYEXCHANGEPARAM

```
typedef struct Struct_SM2_KeyExchange
{
    unsigned char role;
    unsigned char keyLenExpected;
    SM2PUBLICKEYCTX xdata *pSelfTempPubKey;
    SM2PRIVATEKEYCTX xdata *pSelfTempPriKey;
    SM2PUBLICKEYCTX xdata *pSelfPubKey;
    SM2PRIVATEKEYCTX xdata *pSelfPriKey;
    SM2PUBLICKEYCTX xdata *pOtherTempPubKey;
    SM2PUBLICKEYCTX xdata *pOtherPubKey;
    unsigned char xdata *pSelfZ;
    unsigned char xdata *pOtherZ;
}SM2KEYEXCHANGEPARAM;
```

成员

role

密钥交换中的角色，分为两种：

Value	Description
INITIATOR	发起方
RESPONDER	响应方

keyLenExpected

期望交换密钥的长度。

pSelfTempPubKey

己方临时公钥的结构体指针。

pSelfTempPriKey

己方临时私钥的结构体指针。

pSelfPubKey

己方固有公钥的结构体指针。

pSelfPriKey

己方固有私钥的结构体指针。

pOtherTempPubKey

对方临时公钥的结构体指针。

pOtherPubKey

对方固有公钥的结构体指针。

pSelfZ

指向己方的预计算数据 Z 值。

pOtherZ

指向对方的预计算数据 Z 值。

注意

由于结构体成员均为指针，因此存储空间需要外部调用者事先分配好，分配空间的大小需要根据当前使用的曲线位长决定。SM2 使用的 256 位的素域曲线，因此各成员变量所需的存储空间为 256 比特。

keyLenExpected 一般选择 8、16、32 三种长度，但支持不大于 128 字节的任意长度密钥截取。

*pSelfZ*和*pOtherZ*所指向的Z值需要事先调用 SM2GetZ进行计算，其长度均为 256 比特。

快速信息

头文件：声明在 iscrypt_sm2.h

参阅

SM2PUBLICKEYCTX、SM2PRIVATEKEYCTX、SM2KeyExchange、SM2GetZ.

附　表

附表 A　密码算法接口错误定义和说明

表 2　CSDK 算法函数返回值定义说明

宏描述	预定义值	说明
SUCCESS	0x00	成功
FAIL	0x01	失败
UNKNOWN_ERROR	0x02	未知错误
NOT_SUPPORT_YET_ERROR	0x03	功能不支持的错误
NOT_INITIALIZE_ERROR	0x04	未初始化错误
OBJECT_ERROR	0x05	对象错误
MEMORY_ERROR	0x06	内存错误
IN_DATA_LEN_ERROR	0x07	输入数据错误
IN_DATA_ERROR	0x08	输入数据的长度错误
HASH_OBJECT_ERROR	0x09	哈希对象错误
HASH_ERROR	0x0A	哈希运算错误
HASH_NOT_EQUAL_ERROR	0x0B	哈希值不相等错误
MESSAGE_OUT_OF_RANGE	0x0C	消息数据超出约束范围错误
INVALID_PARA	0x0D	无效的参数
NO_ROOM	0x30	空间不足