**PCIeSM API 手册**

**C语言版**

**V2.0**



北京江南天安科技有限公司

2021年09月6日

**版本：**

**当前版本：V2.0.0**

**历史版本：**

|  |  |  |  |
| --- | --- | --- | --- |
| **版本号** | **日期** | **作者** | **描述** |
| 1.0.0 | 2021/06/04 | 李国亮 | 初始版本，加密卡使用和SDF接口函数说明 |
| 2.0.0 | 2021/09/06 | 李国亮 | 1. 增加RSA、SM1、ECC等算法接口功能； 2. 引入配置文件，补充优化文档说明。 |

目录

[1 概述 6](#_Toc26309)

[2 全局说明 6](#_Toc5691)

[2.1.1 密码卡驱动 6](#_Toc27340)

[2.2 数据类型定义 7](#_Toc13870)

[2.2.1 常用数据类型 7](#_Toc25336)

[2.2.2 结构体数据类型 7](#_Toc12597)

[3 动态库文件 10](#_Toc1118)

[3.1 发布包目录 10](#_Toc24120)

[4 配置文件 10](#_Toc28636)

[5 接口使用说明 10](#_Toc20936)

[5.1 编译 10](#_Toc18984)

[6 SDF接口说明 11](#_Toc7817)

[6.1 设备管理 11](#_Toc24854)

[6.1.1 设备资源初始化 11](#_Toc12753)

[6.1.2 设备资源释放 11](#_Toc12330)

[6.1.3 创建设备会话 11](#_Toc30326)

[6.1.4 会话资源释放 12](#_Toc26299)

[6.1.5 获取设备信息 12](#_Toc30436)

[6.1.6 产生随机数 14](#_Toc25648)

[6.1.7 获取指定私钥使用权 14](#_Toc16616)

[6.1.8 释放指定私钥使用权 15](#_Toc797)

[6.2 密钥管理 16](#_Toc8408)

[6.2.1 导出指定RSA签名公钥 16](#_Toc4680)

[6.2.2 导出指定RSA加密公钥 16](#_Toc14081)

[6.2.3 产生指定模长的RSA密钥对(明文) 17](#_Toc4836)

[6.2.4 生成会话密钥并用内部RSA加密公钥加密 19](#_Toc16773)

[6.2.5 生成会话密钥并用外部RSA加密公钥加密 20](#_Toc19833)

[6.2.6 导入会话密钥并用内部RSA私钥解密 21](#_Toc1539)

[6.2.7 会话密钥转加密(RSA) 23](#_Toc7008)

[6.2.8 导出指定SM2签名公钥 24](#_Toc25036)

[6.2.9 导出指定SM2加密公钥 25](#_Toc28850)

[6.2.10 产生指定模长的SM2密钥对 26](#_Toc31854)

[6.2.11 生成会话密钥并用内部SM2加密公钥加密 27](#_Toc21277)

[6.2.12 生成会话密钥并用外部SM2加密公钥加密 28](#_Toc14296)

[6.2.13 导入会话密钥并用内部SM2私钥解密 30](#_Toc3073)

[6.2.14 计算会话密钥(协商方发起协商) 31](#_Toc4713)

[6.2.15 计算会话密钥(协商方响应协商) 33](#_Toc6328)

[6.2.16 计算会话密钥(响应方响应协商) 34](#_Toc4772)

[6.2.17 会话密钥转加密(SM2) 36](#_Toc12805)

[6.2.18 生成会话密钥并用对称密钥加密 38](#_Toc20835)

[6.2.19 导入会话密钥并用对称密钥解密 39](#_Toc19442)

[6.2.20 导入明文会话密钥 41](#_Toc7869)

[6.2.21 销毁会话密钥 41](#_Toc14758)

[6.3 非对称密钥运算 42](#_Toc24314)

[6.3.1 外部RSA公钥运算 42](#_Toc15619)

[6.3.2 内部RSA公钥运算 43](#_Toc26698)

[6.3.3 内部RSA私钥运算 44](#_Toc18435)

[6.3.4 外部SM2私钥签名运算 46](#_Toc12060)

[6.3.5 内部SM2私钥签名运算 46](#_Toc9777)

[6.3.6 外部SM2公钥验签运算 48](#_Toc15945)

[6.3.7 内部SM2公钥验签运算 49](#_Toc13996)

[6.3.8 外部SM2公钥加密运算 50](#_Toc8556)

[6.3.9 外部SM2私钥解密运算 51](#_Toc25504)

[6.3.10 内部SM2公钥加密运算 52](#_Toc23323)

[6.3.11 内部SM2私钥解密运算 53](#_Toc10770)

[6.4 对称密钥运算 54](#_Toc3559)

[6.4.1 数据加密运算 54](#_Toc23916)

[6.4.2 数据解密运算 56](#_Toc30275)

[6.4.3 数据MAC运算 58](#_Toc27708)

[6.5 摘要算法 61](#_Toc1223)

[6.5.1 摘要运算初始化 61](#_Toc22517)

[6.5.2 摘要Update运算 62](#_Toc9067)

[6.5.3 摘要运算结束 63](#_Toc24507)

[6.6 文件操作 64](#_Toc17831)

[6.6.1 创建用户数据文件 64](#_Toc7802)

[6.6.2 读取用户数据文件 64](#_Toc8871)

[6.6.3 写入用户数据文件 66](#_Toc7903)

[6.6.4 删除用户数据文件 67](#_Toc12777)

[7 错误码 67](#_Toc15338)

[7.1 SDF标准错误码 67](#_Toc1534)

# 概述

SJK1829 PCI-E 密码卡是江南天安科技有限公司自主研制，是支持国产密码算法（SM1、SM2、SM3、SM4、SM7）的密码设备，并且符合《密码设备应用接口规范》，可以提供多种国产密码算法，核心部件采用国家密码管理局批准的芯片，具备完善的密钥管理机制，可以提供高性能的数据加解密服务，又可以作为主机数据安全存储系统、身份认证系统以及对称、非对称密钥理

系统的主要密码设备和核心构件。可广泛的应用于银行、保险、证券、交通、邮政、电子商务、移动通信等行业的安全业务应用系统。

# 全局说明

本章节对接口依赖的密码卡驱动信息，以及接口的基础数据对象加以说明。

### 密码卡驱动

在使用密码卡的SDK接口库之前，需要先安装密码卡的驱动，安装成功后才可正常使用，windows和驱动和linux的有所不同：

1. Windows环境：需要安装windows环境下的密码卡的驱动，向江南天安技术人员索要相应的驱动文件和详细安装说明；

2、Linux环境：密码卡驱动名称为tass\_pci.ko，由于该驱动编译时依赖操作系统内核，不同内核版本的操作系统编译的驱动有所不同，所以在使用加密卡的SDK接口库之前，要向江南天安技术人员提供使用的操作系统内核版本，便于提供对应内核版本的密码卡驱动。在拿到linux的驱动后按照如下步骤操作：

使用root权限安装该驱动，在驱动的当前路径执行命令insmod tass\_pci.ko进行测试安装；

正式使用时按照如下步骤安装驱动：

1）将1829密码卡插到机器内的PCIE槽位上，给机器加电卡的红灯“灭”，绿灯“闪”，表示工作状态正常；

2）将tass\_pci.ko上传到机器的/root目录下，在root目录下执行以下命令

创建驱动目录：

mkdir /lib/modules/`uname -r`/kernel/drivers/tass/

拷贝驱动：

cp tass\_pci.ko /lib/modules/`uname -r`/kernel/drivers/tass

安装驱动：

depmod -a

modprobe tass\_pci

3） 检查驱动版本

modinfo tass\_pci

若version 字段跟预期版本一致，则认为驱动安装成功

## 数据类型定义

### 常用数据类型

|  |  |
| --- | --- |
| **数据类型** | **描述** |
| unsigned char | 字符串类型 |
| void \* | 无类型指针，这里主要用于设备句柄使用 |
| int | 整形 |
| unsigned int | 无符号整形 |
| unsigned long | 无符号长整型 |

### 结构体数据类型

typedef struct DeviceInfo\_st {

unsigned char IssuerName[40];

unsigned char DeviceName[16];

unsigned char DeviceSerial[16];

unsigned int DeviceVersion;

unsigned int StandardVersion;

unsigned int AsymAlgAbility[2];

unsigned int SymAlgAbility;

unsigned int HashAlgAbility;

unsigned int BufferSize;

}DEVICEINFO;

|  |  |
| --- | --- |
| **结构体成员** | **描述** |
| IssuerName | 设备生产厂商名称 |
| DeviceName | 设备型号 |
| DeviceSerial | 设备编号，包含日期（8字符）、批次号（3字符）、流水号（5字符） |
| DeviceVersion | 密码设备内部软件版本号 |
| StandardVersion | 密码设备支持的接口规范版本号 |
| AsymAlgAbility | 非对称算法能力，前4字节表示支持的算法，非对称算法标识按位异或，后4字节表示算法的最大模长，表示方法为支持模长按位异或的结果 |
| SymAlgAbility | 对称算法能力，对称算法标识按位异或 |
| HashAlgAbility | 杂凑算法能力，杂凑算法标识按位异或 |
| BufferSize | 支持的最大文件存储空间（单位字节） |

#define RSAref\_MAX\_BITS 4096

#define RSAref\_MAX\_LEN ((RSAref\_MAX\_BITS + 7) / 8)

#define RSAref\_MAX\_PBITS ((RSAref\_MAX\_BITS + 1) / 2)

#define RSAref\_MAX\_PLEN ((RSAref\_MAX\_PBITS + 7) / 8)

typedef struct RSArefPublicKey\_st {

unsigned int bits;

unsigned char m[RSAref\_MAX\_LEN];

unsigned char e[RSAref\_MAX\_LEN];

}RSArefPublicKey;

|  |  |
| --- | --- |
| **结构体成员** | **描述** |
| bits | 模长 |
| m | 模N |
| e | 公钥指数 |

typedef struct RSArefPrivateKey\_st {

unsigned int bits;

unsigned char m[RSAref\_MAX\_LEN];

unsigned char e[RSAref\_MAX\_LEN];

unsigned char d[RSAref\_MAX\_LEN];

unsigned char prime[2][RSAref\_MAX\_PLEN];

unsigned char pexp[2][RSAref\_MAX\_PLEN];

unsigned char coef[RSAref\_MAX\_PLEN];

}RSArefPrivateKey;

|  |  |
| --- | --- |
| **结构体成员** | **描述** |
| bits | 模长 |
| m | 模N |
| e | 公钥指数 |
| d | 私钥指数 |
| prime | prime[0]为私钥P分量，prime[1]为私钥Q分量 |
| pexp | pexp[0]为私钥Dp分量，pexp[1]为私钥Dq分量 |
| coef | 系数i |

#define ECCref\_MAX\_BITS 512

#define ECCref\_MAX\_LEN ((ECCref\_MAX\_BITS + 7) / 8)

typedef struct ECCrefPublicKey\_st {

unsigned int bits;

unsigned char x[ECCref\_MAX\_LEN];

unsigned char y[ECCref\_MAX\_LEN];

}ECCrefPublicKey;

|  |  |
| --- | --- |
| **结构体成员** | **描述** |
| bits | 密钥位长 |
| x | 公钥x坐标 |
| y | 公钥y坐标 |

typedef struct ECCrefPrivateKey\_st {

unsigned int bits;

unsigned char K[ECCref\_MAX\_LEN];

}ECCrefPrivateKey;

|  |  |
| --- | --- |
| **结构体成员** | **描述** |
| bits | 密钥位长 |
| k | 私钥 |

typedef struct ECCCipher\_st {

unsigned char x[ECCref\_MAX\_LEN];

unsigned char y[ECCref\_MAX\_LEN];

unsigned char M[32];

unsigned int L;

unsigned char C[1];

}ECCCipher;

|  |  |
| --- | --- |
| **结构体成员** | **描述** |
| x | X分量 |
| y | Y分量 |
| M | 明文的杂凑值 |
| L | 密文数据长度 |
| c | 密文数据 |

typedef struct ECCSignature\_st {

unsigned char r[ECCref\_MAX\_LEN];

unsigned char s[ECCref\_MAX\_LEN];

}ECCSignature;

|  |  |
| --- | --- |
| **结构体成员** | **描述** |
| r | 签名的r部分 |
| s | 签名的s部分 |

typedef ECCCipher ECCCIPHERBLOB;

typedef ECCrefPublicKey ECCPUBLICKEYBLOB;

typedef struct SDF\_ENVELOPEDKEYBLOB {

unsigned long ulAsymmAlgID;

unsigned long ulSymmAlgID;

ECCCIPHERBLOB ECCCipherBlob;

ECCPUBLICKEYBLOB PubKey;

unsigned char cbEncryptedPriKey[64];

}ENVELOPEDKEYBLOB, \* PENVELOPEDKEYBLOB;

|  |  |
| --- | --- |
| **结构体成员** | **描述** |
| ulAsymmAlgID | 保护对称密钥的非对称算法标识 |
| ulSymmAlgID | 对称算法标识 |
| ECCCipherBlob | 对称密钥明文 |
| PubKey | 加密密钥对的公钥 |
| cbEncryptedPriKey | 加密密钥对的私钥密文，其有效长度为原文的(ulBits+7)/8 |

# 动态库文件

SJK1829 PCI-E 密码卡是以动态库形式对外提供服务，目前接口（SDF接口，Tass通用接口）支持的系统有Windows、Linux的32位和64位系统。

## 发布包目录

所有动态库文件均保存在“libs”目录中，位置和说明见下表：**libTassSDF4PCIeSM**.\*是SDF动态库，其他为依赖库

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **目 录** | **文 件** | **说 明** |
| 1 | cfg | tassPCIeCfg.ini | 配置文件 |
| 2 | demo | PCIeSDF\_Test.cpp | 接口调用实例 |
| 3 | doc | PCIeSM API 手册 C语言版 | 接口文档 |
| 4 | include | SDF4PCIeSM.h  TassAPI4PCIeSM.h  TassType4PCIeSM.h | SDF接口定义文件  通用接口头文件  结构体定义相关头文件 |
| 5 | libs/linux32 | 暂无 | linux 32位操作系统动态库文件  暂不支持 |
| 6 | libs/linux64 | libtass\_pcie\_api.so  libTassSDF4PCIeSM.so | linux 64位操作系统动态库文件 |
| 7 | libs/windows32 | 暂无 | windows 32位操作系统动态库文件  暂不支持 |
| 8 | libs/windows64 | 暂无 | windows 64位操作系统动态库文件  暂不支持 |

# 配置文件

## 配置文件说明

SDF接口配置文件tassPCIeCfg.ini，用于配置使用密码设备的基本信息：

timeout为超时时间，单位为秒；

filePath为读写文件的方式；

filePath = 0 时表示读写文件的接口函数读写到falsh内；

filePath = 路径 时表示读写文件到系统的该路径下。

配置文件放在应用程序的同目录下，也可通过如下的接口函数或环境变量设置路径，三种方式任选其一即可。

## 通过环境变量设置配置文件路径

此步骤为可选项，如有需要可按照下面描述操作：

首先设置配置文件的环境变量，环境变量的名字为TASS\_GHVSM\_CFG\_PATH，

该环境变量指定配置文件的路径，路径只需包含配置文件所在的目录即可，不用包含配置文件的名字，如Linux下配置文件为/etc/tassPCIeCfg.ini，环境变量设置如下：

例如设置临时环境变量：命令行执行export TASS\_PCIE\_CFG\_PATH=/etc

例如设置永久环境变量：在/etc/profile文件最后输入

export TASS\_PCIE\_CFG\_PATH=/etc，然后命令行执行source /etc/profile即可生效。

以上设置环境变量方式仅供参考，也可由其他方式设置，使用时可自行选择。

## 通过接口函数设置配置文件路径

int TassSetCfgPath(const char\* cfgPath);

在调用SDF\_OpenDevice之前调用该函数，参数传入配置文件的路径（不含配置文件名字）即可。

# 接口使用说明

## 编译

* Windows环境VS开发工具静态加载：将libTassSDF4PCIeSM.lib导入到工程中，在工程配置属性->连接器->输入->附加依赖项中输入libTassSDF4PCIeSM.lib所在目录，如：“./lib/libTassSDF4PCIeSM.lib”。
* Linux环境gcc编译加载：动态库链接参数-L. –TassSDF4PCIeSM。
  + 编译Linux的C语言程序：

gcc test.c -o test -I. -L. -lTassSDF4PCIeSM -Wl,-rpath,.

* + 编译提供的demo，Linux环境：

g++ -std=c++11 PCIeSDF\_Test.cpp -o test -I. -L. -lTassSDF4PCIeSM -Wl,-rpath,.

# SDF接口说明

全部接口的入参和出参均封装成对象输入和输出，本文档内请求参数和返回参数针对各个字段作出解释。

## 设备管理

### 设备资源初始化

* **功能介绍**

该接口实现设备的初始化，即打开密码设备。

* **接口定义**

int SDF\_OpenDevice(void\*\* phDeviceHandle);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| phDeviceHandle | void\*\* | 是 | 输出，返回的设备句柄。 |

* **返回参数**

成功返回0，失败返回非0。

### 设备资源释放

* **功能介绍**

该接口实现关闭密码设备，并释放相关资源的功能。

* **接口定义**

int SDF\_CloseDevice(void\* hDeviceHandle);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hDeviceHandle | void\* | 是 | 输入，已打开的设备句柄。 |

* **返回参数**

成功返回0，失败返回非0。

### 创建设备会话

* **功能介绍**

该接口实现创建与密码设备会话资源的功能。

* **接口定义**

int SDF\_OpenSession(void\* hDeviceHandle, void\*\* phSessionHandle);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hDeviceHandle | void\* | 是 | 输入，已打开的设备句柄。 |
| phSessionHandle | void\*\* | 是 | 输出，返回与密码设备建立的新会话句柄 |

* **返回参数**

成功返回0，失败返回非0。

### 会话资源释放

* **功能介绍**

该接口实现关闭与密码设备已建立的会话，并释放相关资源的功能。

* **接口定义**

int SDF\_CloseSession(void\* hSessionHandle);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void\* | 是 | 输入，与密码设备已建立的会话句柄 |

* **返回参数**

成功返回0，失败返回非0。

### 获取设备信息

* **功能介绍**

该接口实现获取密码设备能力描述的功能。

* **接口定义**

int SDF\_GetDeviceInfo(

void\* hSessionHandle,

DEVICEINFO\* pstDeviceInfo);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void\* | 是 | 输入，与密码设备已建立的会话句柄 |
| pstDeviceInfo | DEVICEINFO\* | 是 | 输出，设备能力描述信息 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

int main()

{

void \*hDev = NULL, \*hSess = NULL;

DEVICEINFO devInfo = { 0 };

/// 设备初始化

int rt = SDF\_OpenDevice(&hDev);

if (rt)

{

printf("SDF\_OpenDevice failed %#08x\n", rt);

goto END;

}

/// 会话资源初始化

rt = SDF\_OpenSession(hDev, &hSess);

if (rt)

{

printf("SDF\_OpenSession failed %#08x\n", rt);

goto END;

}

/// 获取设备信息

rt = SDF\_GetDeviceInfo(hSess, &devInfo);

if (rt)

{

printf("\nSDF\_GetDeviceInfo failed %d | 0x%08x\n", rt, rt);

goto END;

}

else

{

printf("\nSDF\_GetDeviceInfo success \n");

printf("IssuerName: %s\n", devInfo.IssuerName);

printf("DeviceName: %s\n", devInfo.DeviceName);

printf("DeviceSerial: %s\n", devInfo.DeviceSerial);

printf("DeviceVersion: %d\n", devInfo.DeviceVersion);

printf("StandardVersion: %d\n", devInfo.StandardVersion);

printf("AsymAlgAbility[0]: %d\n", devInfo.AsymAlgAbility[0]);

printf("AsymAlgAbility[1]: %d\n", devInfo.AsymAlgAbility[1]);

printf("SymAlgAbility: %d\n", devInfo.SymAlgAbility);

printf("HashAlgAbility: %d\n", devInfo.HashAlgAbility);

printf("BufferSize: %d\n", devInfo.BufferSize);

}

/// 测试结束，释放会话资源及设备资源

if (hSess)

SDF\_CloseSession(hSess);

if (hDev)

SDF\_CloseDevice(hDev);

return 0;

END：

if (hSess)

SDF\_CloseSession(hSess);

if (hDev)

SDF\_CloseDevice(hDev);

return -1;

}

### 产生随机数

* **功能介绍**

该接口实现产生随机数的功能。

* **接口定义**

int SDF\_GenerateRandom(

void\* hSessionHandle,

unsigned int uiLength,

unsigned char\* pucRandom);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void\* | 是 | 输入，与密码设备已建立的会话句柄 |
| uiLength | unsigned int | 是 | 输入，欲获取的随机数长度 |
| pucRandom | unsigned char\* | 是 | 输出，随机数字符串 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

unsigned char random[32 + 1] = { 0 };

int rt = SDF\_GenerateRandom(hSess, 32, random);

if (rt)

{

printf("\nSDF\_GenerateRandom failed %d | 0x%08x\n", rt, rt);

return;

}

else

{

printf("\nSDF\_GenerateRandom success\n");

printHex("random", random, 32);

}

### 获取指定私钥使用权

* **功能介绍**

该接口实现获取密码设备内部存储的指定索引私钥使用权的功能。

* **接口定义**

int SDF\_GetPrivateKeyAccessRight(

void\* hSessionHandle,

unsigned int uiKeyIndex,

unsigned char\* pucPassword,

unsigned int uiPwdLength);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void\* | 是 | 输入，与密码设备已建立的会话句柄 |
| uiKeyIndex | unsigned int | 是 | 输入，密码设备存储私钥的索引值，  正数 代表sm2的密钥索引，  负数代表rsa的密钥索引 |
| pucPassword | unsigned char\* | 是 | 输入，使用私钥权限的识别码 |
| uiPwdLength | unsigned int | 是 | 输入，私钥访问控制码长度，不少于8字节 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

int rt = SDF\_GetPrivateKeyAccessRight(

hSess, 0 - 600,

"a1234567", 8);

if (rt)

{

printf("\nGetPrivateKeyAccessRight failed %d | 0x%08x\n",rt,rt);

return;

}

else

printf("\nSDF\_GetPrivateKeyAccessRight success\n");

### 释放指定私钥使用权

* **功能介绍**

该接口实现释放密码设备存储的指定索引私钥使用授权的功能。

* **接口定义**

int SDF\_ReleasePrivateKeyAccessRight(

void\* hSessionHandle,

unsigned int uiKeyIndex);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void\* | 是 | 输入，与密码设备已建立的会话句柄 |
| uiKeyIndex | unsigned int | 是 | 输入，密码设备存储私钥的索引值，  正数 代表sm2的密钥索引，  负数代表rsa的密钥索引 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

SDF\_ReleasePrivateKeyAccessRight(hSess, 0 - 600);

## 密钥管理

### 导出指定RSA签名公钥

* **功能介绍**

该接口实现导出密码设备内部存储的指定索引位置RSA签名公钥的功能。

* **接口定义**

int SDF\_ExportSignPublicKey\_RSA(

void\* hSessionHandle,

unsigned int uiKeyIndex,

RSArefPublicKey\* pucPublicKey);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void\* | 是 | 输入，与密码设备已建立的会话句柄 |
| uiKeyIndex | unsigned int | 是 | 输入，密码设备存储RSA公钥对索引值 |
| pucPublicKey | RSArefPublicKey\* | 是 | 输出，RSA公钥结构 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

RSArefPublicKey rsaSignPubKey = { 0 };

int rt = SDF\_ExportSignPublicKey\_RSA(hSess, 600, &rsaSignPubKey);

if (rt)

{

printf("\nSDF\_ExportSignPublicKey\_RSA failed %d | 0x%08x\n",rt,rt);

return;

}

else

{

printf("\nSDF\_ExportSignPublicKey\_RSA success\n");

printf("SignPubKey.bits: %d\n", rsaSignPubKey.bits);

printf("SignPubKey.m: %s\n", Bin2String(rsaSignPubKey.m, sizeof(rsaSignPubKey.m), true).data());

printf("SignPubKey.e: %s\n", Bin2String(rsaSignPubKey.e, sizeof(rsaSignPubKey.e), true).data());

};

### 导出指定RSA加密公钥

* **功能介绍**

该接口实现导出密码设备内部存储的指定索引位置的加密RSA公钥的功能。

* **接口定义**

int SDF\_ExportEncPublicKey\_RSA(

void\* hSessionHandle,

unsigned int uiKeyIndex,

RSArefPublicKey\* pucPublicKey);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void\* | 是 | 输入，与密码设备已建立的会话句柄 |
| uiKeyIndex | unsigned int | 是 | 输入，密码设备存储RSA公钥对索引值 |
| pucPublicKey | RSArefPublicKey\* | 是 | 输出，RSA公钥结构 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

RSArefPublicKey rsaEncPubKey = { 0 };

int rt = SDF\_ExportEncPublicKey\_RSA(hSess, 600, &rsaSignPubKey);

if (rt)

{

printf("\nSDF\_ExportEncPublicKey\_RSA failed %d | 0x%08x\n",rt,rt);

return;

}

else

{

printf("\nSDF\_ExportEncPublicKey\_RSA success\n");

printf("EncPubKey.bits: %d\n", rsaEncPubKey.bits);

printf("EncPubKey.m: %s\n", Bin2String(rsaEncPubKey.m, sizeof(rsaEncPubKey.m), true).data());

printf("EncPubKey.e: %s\n", Bin2String(rsaEncPubKey.e, sizeof(rsaEncPubKey.e), true).data());

}

### 产生指定模长的RSA密钥对(明文)

* **功能介绍**

该接口实现密码设备产生指定模长的RSA密钥对（明文）的功能。

* **接口定义**

int SDF\_GenerateKeyPair\_RSA(

void\* hSessionHandle,

unsigned int uiKeyBits,

RSArefPublicKey\* pucPublicKey,

RSArefPrivateKey\* pucPrivateKey);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void\* | 是 | 输入，与密码设备已建立的会话句柄 |
| uiKeyBits | unsigned int | 是 | 输入，指定密钥长度 |
| pucPublicKey | RSArefPublicKey \* | 是 | 输出，RSA公钥结构  若需要保存到加密机内部加密密钥对，可设置pucPublicKey->bits为索引值，此时不输出pucPrivateKey |
| pucPrivateKey | RSArefPrivateKey \* | 是 | 输出，RSA私钥结构  若需要保存到加密机内部签名密钥对，可设置pucPrivateKey->bits为索引值，此时不输出pucPrivateKey |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

RSArefPublicKey rsaPubKey = { 0 };

RSArefPrivateKey rsaPriKey = { 0 };

int rt = SDF\_GenerateKeyPair\_RSA(hSess,2048,&rsaPubKey,&rsaPriKey);

if (rt)

{

printf("\nSDF\_GenerateKeyPair\_RSA failed %d | 0x%08x\n", rt, rt);

return;

}

else

{

printf("\nSDF\_GenerateKeyPair\_RSA success\n");

printf("PubKey.bits: %d\n", rsaPubKey.bits);

printf("PubKey.m: %s\n", Bin2String(rsaPubKey.m, sizeof(rsaPubKey.m), true).data());

printf("PubKey.e: %s\n", Bin2String(rsaPubKey.e, sizeof(rsaPubKey.e), true).data());

printf("PriKey.bits: %d\n", rsaPriKey.bits);

printf("PriKey.m: %s\n", Bin2String(rsaPriKey.m, sizeof(rsaPriKey.m), true).data());

printf("PriKey.e: %s\n", Bin2String(rsaPriKey.e, sizeof(rsaPriKey.e), true).data());

printf("PriKey.d: %s\n", Bin2String(rsaPriKey.d, sizeof(rsaPriKey.d), true).data());

printf("PriKey.prime[0]: %s\n", Bin2String(rsaPriKey.prime[0], sizeof(rsaPriKey.prime[0]), true).data());

printf("PriKey.prime[1]: %s\n", Bin2String(rsaPriKey.prime[1], sizeof(rsaPriKey.prime[1]), true).data());

printf("PriKey.pexp[0]: %s\n", Bin2String(rsaPriKey.pexp[0], sizeof(rsaPriKey.pexp[0]), true).data());

printf("PriKey.pexp[1]: %s\n", Bin2String(rsaPriKey.pexp[1], sizeof(rsaPriKey.pexp[1]), true).data());

printf("PriKey.coef: %s\n", Bin2String(rsaPriKey.coef, sizeof(rsaPriKey.coef), true).data());

}

### 生成会话密钥并用内部RSA加密公钥加密

* **功能介绍**

该接口实现密码设备生成会话密钥并用指定索引的内部RSA加密公钥加密输出，同时返回密钥句柄的功能。公钥加密数据是填充方式按照PKCS#1 v1.5的要求进行。

* **接口定义**

int SDF\_GenerateKeyWithIPK\_RSA(

void\* hSessionHandle,

unsigned int uiIPKIndex,

unsigned int uiKeyBits,

unsigned char\* pucKey,

unsigned int\* puiKeyLength,

void\*\* phKeyHandle);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void\* | 是 | 输入，与密码设备已建立的会话句柄 |
| uiIPKIndex | unsigned int | 是 | 输入，密码设备内部存储公钥的索引值 |
| UiKeyBits | unsigned int | 是 | 输入，指定产生的会话密钥长度，只支持128bit（16字节） |
| pucKey | unsigned char \* | 是 | 输出，缓冲区指针，用于存放返回的密钥密文 |
| puiKeyLength | unsigned int \* | 是 | 输出，返回的密钥密文长度 |
| phKeyHandle | void\*\* | 是 | 输出，返回的密钥句柄，传入前需先赋值为NULL  若需要保存到加密机内部，可使用如下方式：  如保存到1号索引密钥，则int idx = 1; void\* pIdx = &idx; 传入 &pIdx 即可，此时phKeyHandle不能调用SDF\_DestroyKey释放 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

unsigned char encKey[1024 + 1] = { 0 };

unsigned int encKeyLen = sizeof(encKey);

void\* hKeyHdl = NULL;

int rt = SDF\_GenerateKeyWithIPK\_RSA(hSess,600,128,encKey,&encKeyLen,&hKeyHdl);

if (rt)

{

printf("\nSDF\_GenerateKeyWithIPK\_RSA failed %d | 0x%08x\n",rt,rt);

return;

}

else

{

printf("\nSDF\_GenerateKeyWithIPK\_RSA success\n");

printf("key 2: %s\n",Bin2String(encKey,encKeyLen,true).data());

rt = SDF\_DestroyKey(hSess, hKeyHdl);

if (rt)

{

printf("\nSDF\_DestroyKey failed %d | 0x%08x\n",rt,rt);

return;

}

printf("\nSDF\_DestroyKey success\n");

hKeyHdl = NULL;

}

### 生成会话密钥并用外部RSA加密公钥加密

* **功能介绍**

该接口实现密码设备生成会话密钥并用外部RSA公钥加密输出，同时返回密钥句柄的功能。公钥加密数据是填充方式按照PKCS#1 v1.5的要求进行。

* **接口定义**

int SDF\_GenerateKeyWithEPK\_RSA(

void\* hSessionHandle,

unsigned int uiKeyBits,

RSArefPublicKey\* pucPublicKey,

unsigned char\* pucKey,

unsigned int\* puiKeyLength,

void\*\* phKeyHandle);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void\* | 是 | 输入，与密码设备已建立的会话句柄 |
| uiKeyBits | unsigned int | 是 | 输入，指定产生的会话密钥长度，只支持128bit（16字节） |
| pucPublicKey | RSArefPublicKey \* | 是 | 输入的外部RSA公钥结构 |
| pucKey | unsigned char \* | 是 | 输出，缓冲区指针，用于存放返回的密钥密文 |
| puiKeyLength | unsigned int \* | 是 | 输出，返回的密钥密文长度 |
| phKeyHandle | void\*\* | 是 | 输出，返回的密钥句柄，传入前需先赋值为NULL |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

unsigned char encKey[1024 + 1] = { 0 };

unsigned int encKeyLen = sizeof(encKey);

void\* hKeyHdl = NULL;

RSArefPublicKey rsaPubKey = { 0 };

RSArefPrivateKey rsaPriKey = { 0 };

//获取公钥

rsaPubKey.bits = 600;

rsaPriKey.bits = 600;

SDF\_GenerateKeyPair\_RSA(hSess, 2048, &rsaPubKey, &rsaPriKey);

//生成会话密钥 外部公钥加密输出

int rt = SDF\_GenerateKeyWithEPK\_RSA(hSess,128,&rsaPubKey,encKey,&encKeyLen,&hKeyHdl);

if (rt)

{

printf("\nSDF\_GenerateKeyWithEPK\_RSA failed %d | 0x%08x\n", rt, rt);

return;

}

else

{

printf("\nSDF\_GenerateKeyWithEPK\_RSA success\n");

printf("key 1: %s\n", Bin2String(encKey, encKeyLen, true).data());

rt = SDF\_DestroyKey(hSess, hKeyHdl);

if (rt)

{

printf("\nSDF\_DestroyKey failed %d | 0x%08x\n", rt, rt);

return;

}

printf("\nSDF\_DestroyKey success\n");

hKeyHdl = NULL;

}

### 导入会话密钥并用内部RSA私钥解密

* **功能介绍**

该接口实现密码设备导入会话密钥并用内部RSA私钥解密，同时返回密钥句柄的功能。公钥加密数据是填充方式按照PKCS#1 v1.5的要求进行。

* **接口定义**

int SDF\_ImportKeyWithISK\_RSA(

void\* hSessionHandle,

unsigned int uiISKIndex,

unsigned char\* pucKey,

unsigned int uiKeyLength,

void\*\* phKeyHandle);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| uiISKIndex | unsigned int | 是 | 输入，密码设备内部存储加密私钥的索引值，对应于加密时的公钥 |
| pucKey | unsigned char \* | 是 | 输入，缓冲区指针，用于存放输入的密钥密文 |
| uiKeyLength | unsigned int | 是 | 输入，输入密钥密文长度 |
| phKeyHandle | void\*\* | 是 | 输出，返回的密钥句柄，传入前需先赋值为NULL |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

unsigned char encKey[1024 + 1] = { 0 };

unsigned int encKeyLen = sizeof(encKey);

void\* hKeyHdl = NULL;

RSArefPublicKey rsaPubKey = { 0 };

RSArefPrivateKey rsaPriKey = { 0 };

//获取公钥

//生成会话密钥 内部公钥加密输出

//获取指定私钥使用权

//导入会话密钥 内部私钥解密

int rt = SDF\_ImportKeyWithISK\_RSA(hSess,600,encKey,encKeyLen,&hKeyHdl);

if (rt)

{

printf("\nSDF\_ImportKeyWithISK\_RSA failed %d | 0x%08x\n",rt,rt);

SDF\_ReleasePrivateKeyAccessRight(hSess, 0 - 600);

return;

}

else

{

printf("\nSDF\_ImportKeyWithISK\_RSA success\n");

SDF\_ReleasePrivateKeyAccessRight(hSess, 0 - 600);

rt = SDF\_DestroyKey(hSess, hKeyHdl);

if (rt)

{

printf("\nSDF\_DestroyKey failed %d | 0x%08x\n", rt, rt);

return;

}

printf("\nSDF\_DestroyKey success\n");

hKeyHdl = NULL;

}

### 会话密钥转加密(RSA)

* **功能介绍**

该接口实现密码设备将由内部RSA公钥加密的会话密钥转换为由外部指定的RSA公钥加密的功能，可用于数字信封转换。公钥加密数据是填充方式按照PKCS#1 v1.5的要求进行。

* **接口定义**

int SDF\_ExchangeDigitEnvelopeBaseOnRSA(

void\* hSessionHandle,

unsigned int uiKeyIndex,

RSArefPublicKey\* pucPublicKey,

unsigned char\* pucDEInput,

unsigned int uiDELength,

unsigned char\* pucDEOuput,

unsigned int\* puiDELength);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| uiISKIndex | unsigned int | 是 | 输入，密码设备存储的内部RSA密钥对索引值 |
| pucPublicKey | RSArefPublicKey \* | 是 | 输入，外部RSA公钥结构 |
| pucDEInput | unsigned char \* | 是 | 输入，缓冲区指针，用于存放输入的会话密钥密文 |
| uiDELength | unsigned int | 是 | 输入，输入的会话密钥密文长度 |
| pucDEOuput | unsigned char \* | 是 | 输出，缓冲区指针，用于存放输出的会话密钥密文 |
| puiDELength | unsigned int \* | 是 | 输出，会话密钥密文长度 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

unsigned char encKey[1024 + 1] = { 0 };

unsigned int encKeyLen = sizeof(encKey);

unsigned char encDEOutKey[1024 + 1] = { 0 };

unsigned int encDEOutKeyLen = sizeof(encDEOutKey);

RSArefPublicKey rsaExPubKey = { 0 };

//获取公钥

//生成会话密钥 外部公钥加密输出

//将外部公钥加密的会话密钥转加密为内部公钥加密

int rt = SDF\_ExchangeDigitEnvelopeBaseOnRSA(hSess,600,rsaExPubKey ,encKey,encKeyLen,encDEOutKey,&encDEOutKeyLen);

if (rt)

{

printf("\nSDF\_ExchangeDigitEnvelopeBaseOnRSA failed %d | 0x%08x\n",rt,rt);

return;

}

else

{

printf("\nSDF\_ExchangeDigitEnvelopeBaseOnRSA success\n");

}

### 导出指定SM2签名公钥

* **功能介绍**

该接口实现密码设备导出密码设备内部存储的指定索引位置SM2签名公钥的功能。

* **接口定义**

int SDF\_ExportSignPublicKey\_ECC(

void\* hSessionHandle,

unsigned int uiKeyIndex,

ECCrefPublicKey\* pucPublicKey);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| uiISKIndex | unsigned int | 是 | 输入，密码设备存储的SM2密钥对索引值 |
| pucPublicKey | ECCrefPublicKey \* | 是 | 输出，SM2公钥结构 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

ECCrefPublicKey signPubKey = { 0 };

int rt = SDF\_ExportSignPublicKey\_ECC(hSess,600,&signPubKey);

if (rt)

{

printf("\nSDF\_ExportSignPublicKey\_ECC failed %d | 0x%08x\n",rt,rt);

return;

}

else

{

printf("\nSDF\_ExportSignPublicKey\_ECC success\n");

printf("SignPubKey.bits: %d\n", signPubKey.bits);

printf("SignPubKey.x: %s\n",Bin2String(signPubKey.x,sizeof(signPubKey.x),true).data());

printf("SignPubKey.y: %s\n",Bin2String(signPubKey.y,sizeof(signPubKey.y),true).data());

}

### 导出指定SM2加密公钥

* **功能介绍**

该接口实现密码设备导出密码设备内部存储的指定索引位置SM2加密公钥的功能。

* **接口定义**

int SDF\_ExportEncPublicKey\_ECC(

void\* hSessionHandle,

unsigned int uiKeyIndex,

ECCrefPublicKey\* pucPublicKey);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| uiISKIndex | unsigned int | 是 | 输入，密码设备存储的SM2密钥对索引值 |
| pucPublicKey | ECCrefPublicKey \* | 是 | 输出，SM2公钥结构 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

ECCrefPublicKey encPubKey = { 0 };

int rt = SDF\_ExportEncPublicKey\_ECC(hSess, 600, &encPubKey);

if (rt)

{

printf("\nSDF\_ExportSignPublicKey\_ECC failed %d | 0x%08x\n",rt,rt);

return;

}

else

{

printf("\nSDF\_ExportSignPublicKey\_ECC success\n");

printf("EncPubKey.bits: %d\n", encPubKey.bits);

printf("EncPubKey.x: %s\n",Bin2String(encPubKey.x,sizeof(encPubKey.x),true).data());

printf("EncPubKey.y: %s\n",Bin2String(encPubKey.y,sizeof(encPubKey.y),true).data());

}

### 产生指定模长的SM2密钥对

* **功能介绍**

该接口实现密码设备请求密码设备产生指定和模长的SM2密钥对的功能。

* **接口定义**

int SDF\_GenerateKeyPair\_ECC(

void\* hSessionHandle,

unsigned int uiAlgID,

unsigned int uiKeyBits,

ECCrefPublicKey\* pucPublicKey,

ECCrefPrivateKey\* pucPrivateKey);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| uiAlgID | unsigned int | 是 | 输入，指定算法标识  SGD\_SM2 – SM2密码算法  SGD\_SM2\_1 – SM2签名算法  SGD\_SM2\_2 – SM2密钥交换协议  SGD\_SM2\_3 – SM2加密算法 |
| uiKeyBits | unsigned int | 是 | 输入，指定密钥模长，只支持256bit（32字节） |
| pucPublicKey | ECCrefPublicKey \* | 是 | 输出，SM2公钥结构  若需要保存到加密机内部加密密钥对，可设置pucPublicKey->bits为索引值，此时不输出pucPrivateKey |
| pucPrivateKey | ECCrefPrivateKey \* | 是 | 输出，SM2私钥结构  若需要保存到加密机内部签名密钥对，可设置pucPrivateKey->bits为索引值，此时不输出pucPrivateKey |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

ECCrefPublicKey pubKey = { 0 };

ECCrefPrivateKey priKey = { 0 };

unsigned int algID = SGD\_SM2;

int rt = SDF\_GenerateKeyPair\_ECC(hSess,algID,256,&pubKey,&priKey);

if (rt)

{

printf("\nSDF\_GenerateKeyPair\_ECC failed %d | 0x%08x\n",rt,rt);

return;

}

else

{

printf("\nSDF\_GenerateKeyPair\_ECC success\n");

printf("PubKey.bits: %d\n", pubKey.bits);

printf("PubKey.x:%s\n",Bin2String(pubKey.x,sizeof(pubKey.x),true).data());

printf("PubKey.y:%s\n",Bin2String(pubKey.y,sizeof(pubKey.y),true).data());

printf("PriKey.bits: %d\n", priKey.bits);

printf("PriKey.K:%s\n",Bin2String(priKey.K,sizeof(priKey.K),true).data());

}

### 生成会话密钥并用内部SM2加密公钥加密

* **功能介绍**

该接口实现密码设备生成会话密钥并用指定索引的内部SM2加密公钥加密输出，同时返回密钥句柄的功能。公钥加密数据是填充方式按照PKCS#1 v1.5的要求进行。

* **接口定义**

int SDF\_GenerateKeyWithIPK\_ECC(

void\* hSessionHandle,

unsigned int uiIPKIndex,

unsigned int uiKeyBits,

ECCCipher\* pucKey,

void\*\* phKeyHandle);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void\* | 是 | 与密码设备已建立的会话句柄 |
| uiIPKIndex | unsigned int | 是 | 密码设备内部存储公钥的索引值 |
| UiKeyBits | unsigned int | 是 | 指定产生的会话密钥长度，只支持128bit（16字节） |
| pucKey | ECCCipher \* | 是 | 输出，缓冲区指针，用于存放返回的密钥密文 |
| phKeyHandle | void\*\* | 是 | 输出，返回的密钥句柄，传入前需先赋值为NULL  若需要保存到加密机内部，可使用如下方式：  如保存到1号索引密钥，则int idx = 1; void\* pIdx = &idx; 传入 &pIdx 即可，此时phKeyHandle不能调用SDF\_DestroyKey释放 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

ECCCipher cipherKey = { 0 };

void\* hKeyHdl = NULL;

//生成会话密钥 内部公钥加密输出

rt = SDF\_GenerateKeyWithIPK\_ECC(hSess,600,128,&cipherKey,&hKeyHdl);

if (rt)

{

printf("\nSDF\_GenerateKeyWithIPK\_ECC failed %d | 0x%08x\n",rt,rt);

return;

}

else

{

printf("\nSDF\_GenerateKeyWithIPK\_ECC success\n");

printf("cipherKey.x:%s\n",Bin2String(cipherKey.x,sizeof(cipherKey.x),true).data());

printf("cipherKey.y:%s\n",Bin2String(cipherKey.y,sizeof(cipherKey.y),true).data());

printf("cipherKey.M:%s\n",Bin2String(cipherKey.M,sizeof(cipherKey.M),true).data());

printf("cipherKey.L:%d\n",cipherKey.L);

printf("cipherKey.C:%s\n",Bin2String(cipherKey.C,cipherKey.L,true).data());

rt = SDF\_DestroyKey(hSess, hKeyHdl);

if (rt) {

printf("\nSDF\_DestroyKey failed %d | 0x%08x\n", rt, rt);

return;

}

printf("\nSDF\_DestroyKey success\n");

hKeyHdl = NULL;

}

### 生成会话密钥并用外部SM2加密公钥加密

* **功能介绍**

该接口实现密码设备生成会话密钥并用外部SM2公钥加密输出，同时返回密钥句柄的功能。公钥加密数据是填充方式按照PKCS#1 v1.5的要求进行。

* **接口定义**

int SDF\_GenerateKeyWithEPK\_ECC(

void\* hSessionHandle,

unsigned int uiKeyBits,

unsigned int uiAlgID,

ECCrefPublicKey\* pucPublicKey,

ECCCipher\* pucKey,

void\*\* phKeyHandle);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void\* | 是 | 输入，与密码设备已建立的会话句柄 |
| uiKeyBits | unsigned int | 是 | 输入，指定产生的会话密钥长度，只支持128bit（16字节） |
| uiAlgID | unsigned int | 是 | 输入，外部SM2公钥的算法标识  SGD\_SM2 – SM2密码算法  SGD\_SM2\_1 – SM2签名算法  SGD\_SM2\_2 – SM2密钥交换协议  SGD\_SM2\_3 – SM2加密算法 |
| pucPublicKey | ECCrefPublicKey \* | 是 | 输入，输入的外部SM2公钥结构 |
| pucKey | ECCCipher \* | 是 | 输出，缓冲区指针，用于存放返回的密钥密文 |
| phKeyHandle | void\*\* | 是 | 输出，返回的密钥句柄，传入前需先赋值为NULL |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

ECCCipher cipherKey = { 0 };

ECCrefPublicKey pubKey = { 0 };

void\* hKeyHdl = NULL;

unsigned int algID = SGD\_SM2;

//生成非对称密钥

//生成会话密钥 外部公钥加密输出

int rt = SDF\_GenerateKeyWithEPK\_ECC(hSess,128,algID,&pubKey,&cipherKey,&hKeyHdl);

if (rt)

{

printf("\nSDF\_GenerateKeyWithIPK\_ECC failed %d | 0x%08x\n", rt, rt);

return;

}

else

{

printf("\nSDF\_GenerateKeyWithIPK\_ECC success\n");

printf("cipherKey.x:%s\n",Bin2String(cipherKey.x,sizeof(cipherKey.x),true).data());

printf("cipherKey.y:%s\n",Bin2String(cipherKey.y,sizeof(cipherKey.y),true).data());

printf("cipherKey.M:%s\n",Bin2String(cipherKey.M,sizeof(cipherKey.M),true).data());

printf("cipherKey.L:%d\n",cipherKey.L);

printf("cipherKey.C:%s\n",Bin2String(cipherKey.C,cipherKey.L,true).data());

rt = SDF\_DestroyKey(hSess, hKeyHdl);

if (rt) {

printf("\nSDF\_DestroyKey failed %d | 0x%08x\n", rt, rt);

return;

}

printf("\nSDF\_DestroyKey success\n");

hKeyHdl = NULL;

}

### 导入会话密钥并用内部SM2私钥解密

* **功能介绍**

该接口实现密码设备导入会话密钥并用内部SM2私钥解密，同时返回密钥句柄的功能。公钥加密数据是填充方式按照PKCS#1 v1.5的要求进行。

* **接口定义**

int SDF\_ImportKeyWithISK\_ECC(

void\* hSessionHandle,

unsigned int uiISKIndex,

ECCCipher\* pucKey,

void\*\* phKeyHandle);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| uiISKIndex | unsigned int | 是 | 输入，密码设备内部存储加密私钥的索引值，对应于加密时的公钥 |
| pucKey | ECCCipher \* | 是 | 输入，缓冲区指针，用于存放输入的密钥密文 |
| phKeyHandle | void\*\* | 是 | 输出，返回的密钥句柄，传入前需先赋值为NULL |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

ECCCipher cipherKey = { 0 };

ECCrefPublicKey pubKey = { 0 };

void\* hKeyHdl = NULL;

unsigned int algID = SGD\_SM2;

//生成非对称密钥

//生成会话密钥 外部公钥加密输出

//获取指定SM2私钥使用权

//导入会话密钥 内部私钥解密

int rt = SDF\_ImportKeyWithISK\_ECC(hSess,600,&cipherKey,&hKeyHdl);

if (rt)

{

printf("\nSDF\_ImportKeyWithISK\_ECC failed %d | 0x%08x\n",rt,rt);

SDF\_ReleasePrivateKeyAccessRight(hSess, keyAsymIndex);

return;

}

else

{

printf("\nSDF\_ImportKeyWithISK\_ECC success\n");

SDF\_ReleasePrivateKeyAccessRight(hSess, keyAsymIndex);

rt = SDF\_DestroyKey(hSess, hKeyHdl);

if (rt) {

printf("\nSDF\_DestroyKey failed %d | 0x%08x\n", rt, rt);

return;

}

printf("\nSDF\_DestroyKey success\n");

hKeyHdl = NULL;

}

### 计算会话密钥(协商方发起协商)

* **功能介绍**

该接口实现密码设备使用SM2密钥协商算法，为计算会话密钥而产生协商参数，同时返回指定索引位置的SM2公钥、临时SM2密钥对的公钥及协商句柄的功能。为协商会话密钥，协商的发起方应首先调用本函数。

* **接口定义**

int SDF\_GenerateAgreementDataWithECC(

void\* hSessionHandle,

unsigned int uiISKIndex,

unsigned int uiKeyBits,

unsigned char\* pucSponsorID,

unsigned int uiSponsorIDLength,

ECCrefPublicKey\* pucSponsorPublicKey,

ECCrefPublicKey\* pucSponsorTmpPublicKey,

void\*\* phAgreementHandle);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| uiISKIndex | unsigned int | 是 | 输入，密码设备内部存储加密私钥的索引值，该私钥用于参与密钥协商 |
| uiKeyBits | unsigned int | 是 | 输入，要求协商的密钥长度，只支持128bit（16字节） |
| pucSponsorID | unsigned char \* | 是 | 输入，参与密钥协商的发起方ID |
| uiSponsorIDLength | unsigned int | 是 | 输入，发起方ID长度 |
| pucSponsorPublicKey | ECCrefPublicKey \* | 是 | 输出，返回的发起方ECC公钥结构 |
| pucSponsorTmpPublicKey | ECCrefPublicKey \* | 是 | 输出，返回的发起方临时SM2公钥结构 |
| phKeyHandle | void\*\* | 是 | 输出，返回的协商句柄，用于计算协商密钥 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

void\* hAgreeHdl = NULL;

ECCrefPublicKey reqPubKey = { 0 };

ECCrefPublicKey reqTmpPubKey = { 0 };

unsigned char reqID[] = "111111";

//获取私钥授权

//发起协商 获取公钥协商参数

int rt = SDF\_GenerateAgreementDataWithECC(hSess,600,128,reqID,strlen((char\*)reqID),&reqPubKey,&reqTmpPubKey,&hAgreeHdl);

if (rt)

{

printf("\nSDF\_GenerateAgreementDataWithECC failed %d | 0x%08x\n", rt, rt);

SDF\_ReleasePrivateKeyAccessRight(hSess, 600);

return;

}

else

{

printf("\nSDF\_GenerateAgreementDataWithECC success\n");

printf("ReqPubKey.bits:%d\n", reqPubKey.bits);

printf("ReqPubKey.x:%s\n", Bin2String(reqPubKey.x,sizeof(reqPubKey.x),true).data());

printf("ReqPubKey.y:%s\n", Bin2String(reqPubKey.y,sizeof(reqPubKey.y),true).data());

printf("ReqTmpPubKey.bits:%d\n", reqTmpPubKey.bits);

printf("ReqTmpPubKey.x:%s\n", Bin2String(reqTmpPubKey.x,sizeof(reqTmpPubKey.x),true).data());

printf("ReqTmpPubKey.y:%s\n", Bin2String(reqTmpPubKey.y,sizeof(reqTmpPubKey.y),true).data());

}

### 计算会话密钥(协商方响应协商)

* **功能介绍**

该接口实现密码设备使用SM2密钥协商算法，使用自身协商句柄和响应方的协商参数计算会话密钥，同时返回会话密钥句柄的功能。协商的发起方获得响应方的协商参数后调用本函数，计算会话密钥，使用SM2算法计算会话密钥的过程见GM/T 0009。

* **接口定义**

int SDF\_GenerateKeyWithECC(

void\* hSessionHandle,

unsigned char\* pucResponseID,

unsigned int uiResponseIDLength,

ECCrefPublicKey\* pucResponsePublicKey,

ECCrefPublicKey\* pucResponseTmpPublicKey,

void\* hAgreementHandle,

void\*\* phKeyHandle);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| pucResponseID | unsigned char \* | 是 | 输入，外部输入的响应方ID值 |
| uiResponseIDLength | unsigned int | 是 | 输入，外部输入的响应方ID长度 |
| pucResponsePublicKey | ECCrefPublicKey \* | 是 | 输入，外部输入的响应方SM2公钥结构 |
| pucResponseTmpPublicKey | ECCrefPublicKey \* | 是 | 输入，外部输入的响应方临时SM2公钥结构 |
| hAgreementHandle | void \* | 是 | 输入，协商句柄，用于计算协商密钥 |
| phKeyHandle | void \*\* | 是 | 输出，返回的密钥句柄，传入前需先赋值为NULL |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

void\* hAgreeHdl = NULL;

void\* hKeyHdl = NULL;

ECCrefPublicKey rspPubKey = { 0 };

ECCrefPublicKey rspTmpPubKey = { 0 };

unsigned char rspID[] = "222222";

//获取私钥授权

//发起协商 获取公钥协商参数

//等待响应方发回协商参数，并根据响应方协商参数计算会话密钥

int rt = SDF\_GenerateKeyWithECC(hSess,rspID,strlen((char\*)rspID),&rspPubKey,&rspTmpPubKey,hAgreeHdl,&hKeyHdl);

if (rt)

{

printf("\nSDF\_GenerateKeyWithECC failed %d | 0x%08x\n", rt, rt);

SDF\_ReleasePrivateKeyAccessRight(hSess, 600);

return;

}

else

{

printf("SDF\_GenerateKeyWithECC success\n");

SDF\_ReleasePrivateKeyAccessRight(hSess, 600);

}

### 计算会话密钥(响应方响应协商)

* **功能介绍**

该接口实现密码设备使用SM2密钥协商算法，产生协商参数并计算会话密钥，同时返回产生的协商参数和密钥句柄的功能。本函数由响应方调用。使用SM2算法计算会话密钥的过程见GM/T 0009。

* **接口定义**

int SDF\_GenerateAgreementDataAndKeyWithECC(

void\* hSessionHandle,

unsigned int uiISKIndex,

unsigned int uiKeyBits,

unsigned char\* pucResponseID,

unsigned int uiResponseIDLength,

unsigned char\* pucSponsorID,

unsigned int uiSponsorIDLength,

ECCrefPublicKey\* pucSponsorPublicKey,

ECCrefPublicKey\* pucSponsorTmpPublicKey,

ECCrefPublicKey\* pucResponsePublicKey,

ECCrefPublicKey\* pucResponseTmpPublicKey,

void\*\* phKeyHandle);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| uiISKIndex | unsigned int | 是 | 输入，密码设备内部存储加密私钥的索引值，该私钥用于参与密钥协商 |
| uiKeyBits | unsigned int | 是 | 输入，要求协商的密钥长度，只支持128bit（16字节） |
| pucResponseID | unsigned char \* | 是 | 输入，响应方ID |
| uiResponseIDLength | unsigned int | 是 | 输入，响应方ID长度 |
| pucSponsorID | unsigned char \* | 是 | 输入，发起方ID |
| uiSponsorIDLength | unsigned int | 是 | 输入，发起方ID长度 |
| pucSponsorPublicKey | ECCrefPublicKey \* | 是 | 输入，外部输入的发起方SM2公钥结构 |
| pucSponsorTmpPublicKey | ECCrefPublicKey \* | 是 | 输入，外部输入的发起方临时SM2公钥结构 |
| pucResponsePublicKey | ECCrefPublicKey \* | 是 | 输出，返回的响应方SM2公钥结构 |
| pucResponseTmpPublicKey | ECCrefPublicKey \* | 是 | 输出，返回的响应方临时SM2公钥结构 |
| phKeyHandle | void\*\* | 是 | 输出，返回的密钥句柄，传入前需先赋值为NULL |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

void\* hKeyHdl = NULL;

ECCrefPublicKey reqPubKey = { 0 };

ECCrefPublicKey reqTmpPubKey = { 0 };

ECCrefPublicKey rspPubKey = { 0 };

ECCrefPublicKey rspTmpPubKey = { 0 };

unsigned char reqID[] = "111111";

unsigned char rspID[] = "222222";

//获取私钥授权

//根据协商方发来的协商参数，用本地公钥进行协商，并返回自己的协商参数

int rt = SDF\_GenerateAgreementDataAndKeyWithECC(

hSess,

600,

128,

rspID,strlen((char\*)rspID),

reqID,strlen((char\*)reqID),

&reqPubKey,&reqTmpPubKey,

&rspPubKey,&rspTmpPubKey,

&hKeyHdl);

if (rt)

{

printf("\nSDF\_GenerateAgreementDataAndKeyWithECC failed %d | 0x%08x\n",rt,rt);

SDF\_ReleasePrivateKeyAccessRight(hSess, 600);

return;

}

else

{

printf("\nSDF\_GenerateAgreementDataAndKeyWithECC success\n");

printf("RspPubKey.bits:%d\n", rspPubKey.bits);

printf("RspPubKey.x:%s\n", Bin2String(rspPubKey.x,sizeof(rspPubKey.x),true).data());

printf("RspPubKey.y:%s\n", Bin2String(rspPubKey.y,sizeof(rspPubKey.y),true).data());

printf("RspTmpPubKey.bits:%d\n", rspTmpPubKey.bits);

printf("RspTmpPubKey.x:%s\n", Bin2String(rspTmpPubKey.x,sizeof(rspTmpPubKey.x),true).data());

printf("RspTmpPubKey.y:%s\n", Bin2String(rspTmpPubKey.y,sizeof(rspTmpPubKey.y),true).data());

rt = SDF\_DestroyKey(hSess, hKeyHdl);

if (rt)

{

printf("\nSDF\_DestroyKey failed %d | 0x%08x\n", rt, rt);

return;

}

printf("\nSDF\_DestroyKey success\n");

hKeyHdl = NULL;

}

### 会话密钥转加密(SM2)

* **功能介绍**

该接口实现密码设备将由内部SM2公钥加密的会话密钥转换为由外部指定的SM2公钥加密的功能，可用于数字信封转换。公钥加密数据是填充方式按照PKCS#1 v1.5的要求进行。

* **接口定义**

int SDF\_ExchangeDigitEnvelopeBaseOnECC(

void\* hSessionHandle,

unsigned int uiKeyIndex,

unsigned int uiAlgID,

ECCrefPublicKey\* pucPublicKey,

ECCCipher\* pucEncDataIn,

ECCCipher\* pucEncDataOut);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| uiKeyIndex | unsigned int | 是 | 密码设备存储的内部ECC密钥对索引值 |
| uiAlgID | unsigned int | 是 | 输入，外部SM2公钥的算法标识  SGD\_SM2 – SM2密码算法  SGD\_SM2\_1 – SM2签名算法  SGD\_SM2\_2 – SM2密钥交换协议  SGD\_SM2\_3 – SM2加密算法 |
| pucPublicKey | ECCrefPublicKey \* | 是 | 输入，外部SM2公钥结构 |
| pucEncDataIn | ECCCipher \* | 是 | 输入，缓冲区指针，用于存放输入的会话密钥密文 |
| pucEncDataOut | ECCCipher \* | 是 | 输出，缓冲区指针，用于存放输出的会话密钥密文 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

ECCrefPublicKey encPubKey = { 0 };

ECCCipher cipherByOutPubKey = { 0 };

ECCCipher cipherByInPubKey = { 0 };

unsigned int algID = SGD\_SM2;

//获取公钥

//生成会话密钥 内部公钥加密输出

//获取私钥授权

//将内部公钥加密的会话密钥转为外部公钥加密的会话密钥

int rt = SDF\_ExchangeDigitEnvelopeBaseOnECC(

hSess,600,

algID,&encPubKey,

&cipherByInPubKey,&cipherByOutPubKey);

if (rt)

{

printf("\nSDF\_ExchangeDigitEnvelopeBaseOnECC %d | 0x%08x\n",rt,rt);

SDF\_ReleasePrivateKeyAccessRight(hSess, 600);

return;

}

else

{

printf("\nSDF\_ExchangeDigitEnvelopeBaseOnECC success\n");

printf("cipherByOutPubKey.x:%s\n",Bin2String(cipherByOutPubKey.x,sizeof(cipherByOutPubKey.x),true).data());

printf("cipherByOutPubKey.y:%s\n",Bin2String(cipherByOutPubKey.y,sizeof(cipherByOutPubKey.y),true).data());

printf("cipherByOutPubKey.M:%s\n",Bin2String(cipherByOutPubKey.M,sizeof(cipherByOutPubKey.M),true).data());

printf("cipherByOutPubKey.L:%d\n",cipherByOutPubKey.L);

printf("cipherByOutPubKey.C:%s\n",Bin2String(cipherByOutPubKey.C,cipherByOutPubKey.L,true).data());

SDF\_ReleasePrivateKeyAccessRight(hSess, 600);

}

### 生成会话密钥并用对称密钥加密

* **功能介绍**

该接口实现密码设备生成会话密钥并用密钥加密密钥加密输出，同时返回密钥句柄的功能。加密模式使用ECB模式。

* **接口定义**

int SDF\_GenerateKeyWithKEK(

void\* hSessionHandle,

unsigned int uiKeyBits,

unsigned int uiAlgID,

unsigned int uiKEKIndex,

unsigned char\* pucKey,

unsigned int\* puiKeyLength,

void\*\* phKeyHandle);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void\* | 是 | 输入，与密码设备已建立的会话句柄 |
| uiKeyBits | unsigned int | 是 | 输入，指定产生的会话密钥长度，只支持128bit（16字节） |
| uiAlgID | unsigned int | 是 | 输入，算法标识，指定对称加密算法  SGD\_SM1\_ECB - SM1  SGD\_SSF33\_ECB - SSF33  SGD\_SM4\_ECB - SM4  SGD\_ZUC\_EEA3 - ZUC祖冲之 |
| uiKEKIndex | unsigned int | 是 | 输入，密码设备内部存储的加密密钥的索引值 |
| pucKey | unsigned char \* | 是 | 输出，缓冲区指针，用于存放返回的密钥密文 |
| puiKeyLength | unsigned int \* | 是 | 输出，返回的密钥密文长度 |
| phKeyHandle | void\*\* | 是 | 输出，返回的密钥句柄，传入前需先赋值为NULL |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

void\* hKeyHdl = NULL;

unsigned char encKey[BUF] = { 0 };

unsigned int encKeyLen = sizeof(encKey);

//生成会话密钥 加密输出

int rt = SDF\_GenerateKeyWithKEK(hSess,128,algID,601,encKey,&encKeyLen,&hKeyHdl);

if (rt)

{

printf("\n[algID = %s]SDF\_GenerateKeyWithKEK failed 0x%08x\n",GetAlgStr(algID),rt);

return;

}

else

{

printf("\n[algID = %s]SDF\_GenerateKeyWithKEK success\n",GetAlgStr(algID));

printf("encKey:%s\n",Bin2String(encKey, encKeyLen, true).data());

//使用会话密钥句柄做一些操作......

rt = SDF\_DestroyKey(hSess, hKeyHdl);

if (rt)

{

printf("\nSDF\_DestroyKey failed %d | 0x%08x\n", rt, rt);

return;

}

printf("\nSDF\_DestroyKey success\n");

hKeyHdl = NULL;

}

### 导入会话密钥并用对称密钥解密

* **功能介绍**

该接口实现密码设备导入会话密钥并用对称密钥解密，同时返回密钥句柄的功能。解密模式使用ECB模式。

* **接口定义**

int SDF\_ImportKeyWithKEK(

void\* hSessionHandle,

unsigned int uiAlgID,

unsigned int uiKEKIndex,

unsigned char\* pucKey,

unsigned int uiKeyLength,

void\*\* phKeyHandle);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| uiAlgID | unsigned int | 是 | 算法标识，指定对称加密算法  SGD\_SM1\_ECB - SM1  SGD\_SSF33\_ECB - SSF33  SGD\_SM4\_ECB - SM4  SGD\_ZUC\_EEA3 - ZUC祖冲之 |
| uiKEKIndex | unsigned int | 是 | 输入，密码设备内部存储的加密密钥的索引值 |
| pucKey | unsigned char \* | 是 | 输入，缓冲区指针，用于存放输入的密钥密文 |
| uiKeyLength | unsigned int | 是 | 输入，输入密钥密文长度 |
| phKeyHandle | void\*\* | 是 | 输出，返回的密钥句柄，传入前需先赋值为NULL |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

void\* hKeyHdl = NULL;

unsigned char encKey[BUF] = { 0 };

unsigned int encKeyLen = sizeof(encKey);

//生成会话密钥 加密输出

//导入加密的会话密钥，用对称密钥解密

int rt = SDF\_ImportKeyWithKEK(hSess,algID,601,encKey,encKeyLen,&hKeyHdl);

if (rt)

{

printf("\n[algID = %s]SDF\_ImportKeyWithKEK failed 0x%08x\n", GetAlgStr(algID), rt);

return;

}

else

{

printf("\n[algID = %s]SDF\_ImportKeyWithKEK success\n", GetAlgStr(algID));

//使用会话密钥句柄做一些操作......

rt = SDF\_DestroyKey(hSess, hKeyHdl);

if (rt)

{

printf("\nSDF\_DestroyKey failed %d | 0x%08x\n", rt, rt);

return;

}

printf("\nSDF\_DestroyKey success\n");

hKeyHdl = NULL;

}

### 导入明文会话密钥

* **功能介绍**

该接口实现密码设备导入明文会话密钥并用对称密钥解密，同时返回密钥句柄的功能。

* **接口定义**

int SDF\_ImportKey(

void\* hSessionHandle,

unsigned char\* pucKey,

unsigned int uiKeyLength,

void\*\* phKeyHandle);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| pucKey | unsigned char \* | 是 | 输入，缓冲区指针，用于存放输入的密钥密文 |
| uiKeyLength | unsigned int | 是 | 输入，输入密钥密文长度 |
| phKeyHandle | void\*\* | 是 | 输出，返回的密钥句柄，传入前需先赋值为NULL |

* **返回参数**

成功返回0，失败返回非0。

### 销毁会话密钥

* **功能介绍**

该接口实现密码设备销毁会话密钥，并释放为密钥句柄分配的内存等资源的功能。在对称算法运算完成后，应调用本函数销毁会话密钥。

* **接口定义**

int SDF\_DestroyKey(

void\* hSessionHandle,

void\* hKeyHandle);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| hKeyHandle | void \* | 是 | 输入，输入的密钥句柄 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

void\* hKeyHdl = NULL;

//获取会话密钥

//销毁会话密钥

rt = SDF\_DestroyKey(hSess, hKeyHdl);

if (rt)

{

printf("\nSDF\_DestroyKey failed %d | 0x%08x\n", rt, rt);

return;

}

printf("\nSDF\_DestroyKey success\n");

hKeyHdl = NULL;

## 非对称密钥运算

### 外部RSA公钥运算

* **功能介绍**

该接口实现密码设备使用指定RSA外部公钥对数据进行运算的功能。数据格式由应用层封装。

* **接口定义**

int SDF\_ExternalPublicKeyOperation\_RSA(

void\* hSessionHandle,

RSArefPublicKey\* pucPublicKey,

unsigned char\* pucDataInput,

unsigned int uiInputLength,

unsigned char\* pucDataOutput,

unsigned int\* puiOutputLength);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| pucPublicKey | RSArefPublicKey \* | 是 | 输入，外部RSA公钥结构 |
| pucDataInput | unsigned char \* | 是 | 输入，缓冲区指针，用于存放输入的数据 |
| uiInputLength | unsigned int | 是 | 输入，输入的数据长度 |
| pucDataOutput | unsigned char \* | 是 | 输出，缓冲区指针，用于存放输出的数据 |
| puiOutputLength | unsigned int \* | 是 | 输出，输出的数据长度 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

RSArefPublicKey rsaPubKey = { 0 };

unsigned char dataPlain[1024] = { 0 };

unsigned int dataPlainLen = 512;

unsigned char dataEnc[1024] = { 0 };

unsigned int dataEncLen = sizeof(dataEnc);

for (int i = 0; i < dataPlainLen; i++)

dataPlain[i] = 'a';

//获取公钥

//外部RSA公钥运算

int rt = SDF\_ExternalPublicKeyOperation\_RSA(

hSess, &rsaPubKey,

dataPlain, dataPlainLen,

dataEnc, &dataEncLen);

if (rt)

{

printf("\nSDF\_ExternalPublicKeyOperation\_RSA failed0x%08x\n", rt);

return;

}

else

{

printf("\nSDF\_ExternalPublicKeyOperation\_RSA success\n");

printf("dataEnc:%s\n",Bin2String(dataEnc,dataEncLen,true).data());

}

### 内部RSA公钥运算

* **功能介绍**

该接口实现密码设备使用内部指定索引的RSA公钥对数据进行运算的功能。索引范围仅限于内部签名密钥对，数据格式由应用层封装。

* **接口定义**

int SDF\_InternalPublicKeyOperation\_RSA(

void\* hSessionHandle,

unsigned int uiKeyIndex,

unsigned char\* pucDataInput,

unsigned int uiInputLength,

unsigned char\* pucDataOutput,

unsigned int\* puiOutputLength);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| uiKeyIndex | unsigned int | 是 | 输入，密码设备内部存储RSA公钥的索引值 |
| pucDataInput | unsigned char \* | 是 | 输入，缓冲区指针，用于存放输入的数据 |
| uiInputLength | unsigned int | 是 | 输入，输入的数据长度 |
| pucDataOutput | unsigned char \* | 是 | 输出，缓冲区指针，用于存放输出的数据 |
| puiOutputLength | unsigned int \* | 是 | 输出，输出的数据长度 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

unsigned char dataPlain[1024] = { 0 };

unsigned int dataPlainLen = 512;

unsigned char dataEnc[1024] = { 0 };

unsigned int dataEncLen = sizeof(dataEnc);

for (int i = 0; i < dataPlainLen; i++)

dataPlain[i] = 'a';

//内部公钥运算

int rt = SDF\_InternalPublicKeyOperation\_RSA(

hSess, 600,

dataPlain, dataPlainLen,

dataEnc, &dataEncLen);

if (rt)

{

printf("\nSDF\_InternalPublicKeyOperation\_RSA failed %d | 0x%08x\n", rt, rt);

return;

}

else

{

printf("\nSDF\_InternalPublicKeyOperation\_RSA success\n");

printf("dataEnc: %s\n", Bin2String(dataEnc, dataEncLen, true).data());

}

### 内部RSA私钥运算

* **功能介绍**

该接口实现密码设备使用内部指定索引的私钥对数据进行运算的功能。索引范围仅限于内部签名密钥对，数据格式由应用层封装。

* **接口定义**

int SDF\_InternalPrivateKeyOperation\_RSA(

void\* hSessionHandle,

unsigned int uiKeyIndex,

unsigned char\* pucDataInput,

unsigned int uiInputLength,

unsigned char\* pucDataOutput,

unsigned int\* puiOutputLength);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| uiKeyIndex | unsigned int | 是 | 输入，密码设备内部存储RSA私钥的索引值 |
| pucDataInput | unsigned char \* | 是 | 输入，缓冲区指针，用于存放输入的数据 |
| uiInputLength | unsigned int | 是 | 输入，输入的数据长度 |
| pucDataOutput | unsigned char \* | 是 | 输出，缓冲区指针，用于存放输出的数据 |
| puiOutputLength | unsigned int \* | 是 | 输出，输出的数据长度 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

unsigned char dataPlain[1024] = { 0 };

unsigned int dataPlainLen = 512;

unsigned char dataEnc[1024] = { 0 };

unsigned int dataEncLen = sizeof(dataEnc);

unsigned char dataOut[1024] = { 0 };

unsigned int dataOutLen = sizeof(dataOut);

for (int i = 0; i < dataPlainLen; i++)

dataPlain[i] = 'a';

//公钥运算

//获取私钥授权

//私钥运算

int rt = SDF\_InternalPrivateKeyOperation\_RSA(

hSess, 600,

dataEnc, dataEncLen,

dataOut, &dataOutLen);

if (rt)

{

printf("\nSDF\_InternalPrivateKeyOperation\_RSA failed %d | 0x%08x\n", rt, rt);

SDF\_ReleasePrivateKeyAccessRight(hSess, 0 - 600);

return;

}

else

{

printf("\nSDF\_InternalPrivateKeyOperation\_RSA success\n");

printf("dataOut : %s\n", dataOut);

printf("dataPlain: %s\n", dataPlain);

if (memcmp(dataPlain, dataOut, dataOutLen) != 0)

{

printf("RSA Key Operation failed. \n");

}

}

SDF\_ReleasePrivateKeyAccessRight(hSess, 0 - 600);

### 外部SM2私钥签名运算

* **功能介绍**

该接口实现密码设备使用外部公钥对数据进行签名运算的功能。

* **接口定义**

int SDF\_SDF\_ExternalSign\_ECC(

void\* hSessionHandle,

unsigned int uiAlgID,

ECCrefPrivateKey\* pucPrivateKey,

unsigned char\* pucData,

unsigned int uiDataLength,

ECCSignature\* pucSignature);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| uiAlgID | unsigned int | 是 | 输入，算法标识，指定使用的 ECC 算法 |
| pucPrivateKey | ECCrefPrivateKey\* | 是 | 外部 ECC 私钥结构 |
| pucData | unsigned char \* | 是 | 输入，缓冲区指针，用于存放输入的数据 |
| uiDataLength | unsigned int | 是 | 输入，输入的数据长度 |
| pucSignature | ECCSignature \* | 是 | 输出，缓冲区指针，用于存放输出的签名值数据 |

* **返回参数**

成功返回0，失败返回非0。

### 内部SM2私钥签名运算

* **功能介绍**

该接口实现密码设备使用内部SM2私钥对数据进行签名运算的功能。输入数据为待签数据的杂凑值。当使用SM2算法时，该输入数据位待签数据经过SM2签名预处理的结果，预处理过程见GM/T0009。

* **接口定义**

int SDF\_InternalSign\_ECC(

void\* hSessionHandle,

unsigned int uiISKIndex,

unsigned char\* pucData,

unsigned int uiDataLength,

ECCSignature\* pucSignature);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| uiISKIndex | unsigned int | 是 | 输入，密码设备内部存储的ECC签名私钥的索引值 |
| pucData | unsigned char \* | 是 | 输入，缓冲区指针，用于存放输入的数据 |
| uiDataLength | unsigned int | 是 | 输入，输入的数据长度 |
| pucSignature | ECCSignature \* | 是 | 输出，缓冲区指针，用于存放输出的签名值数据 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

ECCSignature sign = { 0 };

unsigned char data[] = "12345678123456781234567812345678";

unsigned int dataLen = strlen((char\*)data);

//获取私钥授权

//私钥签名

int rt = SDF\_InternalSign\_ECC(hSess, 600, data, dataLen, &sign);

if (rt)

{

printf("\nSDF\_InternalSign\_ECC failed %d | 0x%08x\n", rt, rt);

SDF\_ReleasePrivateKeyAccessRight(hSess, 600);

return;

}

else

{

printf("\nSDF\_InternalSign\_ECC success\n");

printf("sign.r: %s\n", Bin2String(sign.r, sizeof(sign.r), true).data());

printf("sign.s: %s\n", Bin2String(sign.s, sizeof(sign.s), true).data());

SDF\_ReleasePrivateKeyAccessRight(hSess, 600);

}

### 外部SM2公钥验签运算

* **功能介绍**

该接口实现密码设备使用外部SM2公钥对SM2签名值进行验证运算的功能。输入数据为待签数据的杂凑值。当使用SM2算法时，该输入数据位待签数据经过SM2签名预处理的结果，预处理过程见GM/T0009。

* **接口定义**

int SDF\_ExternalVerify\_ECC(

void\* hSessionHandle,

unsigned int uiAlgID,

ECCrefPublicKey\* pucPublicKey,

unsigned char\* pucData,

unsigned int uiDataLength,

ECCSignature\* pucSignature);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| uiAlgID | unsigned int | 是 | 输入，算法标识，指定使用的ECC算法  SGD\_SM2 – SM2密码算法  SGD\_SM2\_1 – SM2签名算法  SGD\_SM2\_2 – SM2密钥交换协议  SGD\_SM2\_3 – SM2加密算法 |
| pucPublicKey | ECCrefPublicKey \* | 是 | 输入，外部ECC公钥结构 |
| pucData | unsigned char\* | 是 | 输入，缓冲区指针，用于存放输入的数据 |
| uiDataLength | unsigned int | 是 | 输入，输入的数据长度 |
| pucSignature | ECCSignature \* | 是 | 输入，缓冲区指针，用于存放输入的签名值数据 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

ECCrefPublicKey pubKey = { 0 };

unsigned int algID = SGD\_SM2\_1;

ECCSignature sign = { 0 };

unsigned char data[] = "12345678123456781234567812345678";

unsigned int dataLen = strlen((char\*)data);

//获取公钥

//私钥签名

//内部公钥验签

int rt = SDF\_ExternalVerify\_ECC(hSess,algID,&pubKey,data,dataLen,&sign);

if (rt)

{

printf("\nSDF\_ExternalVerify\_ECC failed %d | 0x%08x\n", rt, rt);

return;

}

else

printf("\nSDF\_ExternalVerify\_ECC success\n");

### 内部SM2公钥验签运算

* **功能介绍**

该接口实现密码设备使用内部SM2公钥对SM2签名值进行验证运算的功能。输入数据为待签数据的杂凑值。当使用SM2算法时，该输入数据位待签数据经过SM2签名预处理的结果，预处理过程见GM/T0009。

* **接口定义**

int SDF\_InternalVerify\_ECC(

void\* hSessionHandle,

unsigned int uiIPKIndex,

unsigned char\* pucData,

unsigned int uiDataLength,

ECCSignature\* pucSignature);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| uiIPKIndex | unsigned int | 是 | 输入，密码设备内部存储的ECC签名公钥的索引值 |
| pucData | unsigned char\* | 是 | 输入，缓冲区指针，用于存放输入的数据 |
| uiDataLength | unsigned int | 是 | 输入，输入的数据长度 |
| pucSignature | ECCSignature \* | 是 | 输入，缓冲区指针，用于存放输入的签名值数据 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

ECCSignature sign = { 0 };

unsigned char data[] = "12345678123456781234567812345678";

unsigned int dataLen = strlen((char\*)data);

//获取私钥授权

//私钥签名

//内部公钥验签

int rt = SDF\_InternalVerify\_ECC(hSess, 600, data, dataLen, &sign);

if (rt)

{

printf("\nSDF\_InternalVerify\_ECC failed %d | 0x%08x\n", rt, rt);

return;

}

else

printf("\nSDF\_InternalVerify\_ECC success\n");

### 外部SM2公钥加密运算

* **功能介绍**

该接口实现密码设备使用外部ECC公钥对数据进行加密运算的功能。

* **接口定义**

int SDF\_ExternalEncrypt\_ECC(

void\* hSessionHandle,

unsigned int uiAlgID,

ECCrefPublicKey\* pucPublicKey,

unsigned char\* pucData,

unsigned int uiDataLength,

ECCCipher\* pucEncData);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| uiAlgID | unsigned int | 是 | 输入，算法标识，指定使用的ECC算法  SGD\_SM2 – SM2密码算法  SGD\_SM2\_1 – SM2签名算法  SGD\_SM2\_2 – SM2密钥交换协议  SGD\_SM2\_3 – SM2加密算法 |
| pucPublicKey | ECCrefPublicKey \* | 是 | 输入，外部ECC公钥结构 |
| pucData | unsigned char\* | 是 | 输入，缓冲区指针，用于存放输入的数据 |
| uiDataLength | unsigned int | 是 | 输入，输入的数据长度 |
| pucEncData | ECCCipher\* | 是 | 输出，缓冲区指针，用于存放输出的数据密文 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

ECCCipher encData = { 0 };

ECCrefPublicKey pubKey = { 0 };

unsigned char dataPlain[] = "12345678123456781234567812345678";

unsigned int dataPlainLen = strlen((char\*)dataPlain);

unsigned int algID = SGD\_SM2\_3;

//生成非对称密钥

//使用外部ECC公钥 加密运算

int rt = SDF\_ExternalEncrypt\_ECC(

hSess,algID,

&pubKey,dataPlain,

dataPlainLen,&encData);

if (rt)

{

printf("\nSDF\_ExternalEncrypt\_ECC failed %d | 0x%08x\n", rt, rt);

return;

}

else

{

printf("\nSDF\_ExternalEncrypt\_ECC success\n");

printf("encData.x: %s\n", Bin2String(encData.x, sizeof(encData.x), true).data());

printf("encData.y: %s\n", Bin2String(encData.y, sizeof(encData.y), true).data());

printf("encData.M: %s\n", Bin2String(encData.M, sizeof(encData.M), true).data());

printf("encData.L: %d\n", encData.L);

printf("encData.C: %s\n", Bin2String(encData.C, encData.L, true).data());

}

### 外部SM2私钥解密运算

* **功能介绍**

该接口实现密码设备使用外部ECC私钥对数据进行解密运算的功能。

int SDF\_ExternalDecrypt\_ECC(

void\* hSessionHandle,

unsigned int uiAlgID,

ECCrefPrivateKey\* pucPrivateKey,

ECCCipher\* pucEncData,

unsigned char\* pucData,

unsigned int\* puiDataLength);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| uiAlgID | unsigned int | 是 | 输入，算法标识，指定使用的ECC算法  SGD\_SM2 – SM2密码算法  SGD\_SM2\_1 – SM2签名算法  SGD\_SM2\_2 – SM2密钥交换协议  SGD\_SM2\_3 – SM2加密算法 |
| pucPrivateKey | ECCrefPrivateKey\* | 是 | 输入，外部ECC私钥结构 |
| pucEncData | ECCCipher\* | 是 | 输入，缓冲区指针，用于存放输入的数据密文 |
| pucData | unsigned char\* | 是 | 输出，缓冲区指针，用于存放输出的数据明文 |
| puiDataLength | unsigned int\* | 是 | 输出，缓冲区指针，输出的数据明文长度 |

* **返回参数**

成功返回0，失败返回非0。

### 内部SM2公钥加密运算

* **功能介绍**

该接口实现密码设备使用外部ECC公钥对数据进行加密运算的功能。

* **接口定义**

int SDF\_InternalEncrypt\_ECC(

void\* hSessionHandle,

unsigned int uiIPKIndex,

unsigned char\* pucData,

unsigned int uiDataLength,

ECCCipher\* pucEncData);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| uiIPKIndex | unsigned int | 是 | 输入，密码设备内部存储的ECC加密公钥的索引值，负数时使用签名密钥对 |
| pucData | unsigned char\* | 是 | 输入，缓冲区指针，用于存放输入的数据 |
| uiDataLength | unsigned int | 是 | 输入，输入的数据长度 |
| pucEncData | ECCCipher\* | 是 | 输出，缓冲区指针，用于存放输出的数据密文 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

ECCCipher encData = { 0 };

unsigned char dataPlain[] = "12345678123456781234567812345678";

unsigned int dataPlainLen = strlen((char\*)dataPlain);

//使用内部ECC公钥 加密运算

int rt = SDF\_InternalEncrypt\_ECC(

hSess, 600,

dataPlain, dataPlainLen, &encData);

if (rt)

{

printf("\nSDF\_InternalEncrypt\_ECC failed %d | 0x%08x\n", rt, rt);

SDF\_ReleasePrivateKeyAccessRight(hSess, 600);

return;

}

else

{

printf("\nSDF\_InternalEncrypt\_ECC success\n");

printf("encData.x: %s\n", Bin2String(encData.x, sizeof(encData.x), true).data());

printf("encData.y: %s\n", Bin2String(encData.y, sizeof(encData.y), true).data());

printf("encData.M: %s\n", Bin2String(encData.M, sizeof(encData.M), true).data());

printf("encData.L: %d\n", encData.L);

printf("encData.C: %s\n", Bin2String(encData.C, encData.L, true).data());

}

### 内部SM2私钥解密运算

* **功能介绍**

该接口实现密码设备使用内部ECC私钥对数据进行解密运算的功能。

* **接口定义**

int SDF\_InternalDecrypt\_ECC(

void\* hSessionHandle,

unsigned int uiISKIndex,

ECCCipher\* pucEncData,

unsigned char\* pucData,

unsigned int\* puiDataLength);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| uiISKIndex | unsigned int | 是 | 输入，密码设备内部存储的ECC解密私钥的索引值，负数时使用签名密钥对 |
| pucEncData | ECCCipher \* | 是 | 输入，缓冲区指针，用于存放输入的数据密文 |
| pucData | unsigned char \* | 是 | 输出，缓冲区指针，用于存放输出的数据 |
| puiDataLength | unsigned int \* | 是 | 输出，输出的数据长度 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

ECCCipher encData = { 0 };

unsigned char dataOut[1024] = { 0 };

unsigned int dataOutLen = sizeof(dataOut);

//公钥加密

//获取私钥授权

//使用内部ECC私钥 解密运算

int rt = SDF\_InternalDecrypt\_ECC(hSess, 600, &encData, dataOut, &dataOutLen);

if (rt)

{

printf("\nSDF\_InternalDecrypt\_ECC failed %d | 0x%08x\n", rt, rt);

SDF\_ReleasePrivateKeyAccessRight(hSess, 600);

return;

}

else

{

printf("\nSDF\_InternalDecrypt\_ECC success\n");

printf("dataOut : %s\n", Bin2String(dataOut, dataOutLen, true).data());

printf("dataPlain: %s\n", Bin2String(dataPlain, dataPlainLen, true).data());

SDF\_ReleasePrivateKeyAccessRight(hSess, 600);

}

## 对称密钥运算

### 数据加密运算

* **功能介绍**

该接口实现密码设备使用指定的密钥句柄和IV对数据进行对称加密运算的功能。此函数不对数据进行填充处理，输入的数据必须是指定算法分组长度的整数倍。

* **接口定义**

int SDF\_Encrypt(

void\* hSessionHandle,

void\* hKeyHandle,

unsigned int uiAlgID,

unsigned char\* pucIV,

unsigned char\* pucData,

unsigned int uiDataLength,

unsigned char\* pucEncData,

unsigned int\* puiEncDataLength);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| hKeyHandle | void \* | 是 | 输入，指定的密钥句柄，若需要使输入，用加密机内部密钥，可使用如下方式  如使用1号索引密钥，则int idx = 1; 传入 &idx 即可 |
| uiAlgID | unsigned int | 是 | 输入，算法标识，指定的对称加密算法  SGD\_SM1\_ECB - SM1算法ECB加密模式  SGD\_SM1\_CBC - SM1算法CBC加密模式  SGD\_SM1\_CFB - SM1算法CFB加密模式  SGD\_SM1\_OFB - SM1算法OFB加密模式  SGD\_SSF33\_ECB - SSF33算法ECB加密模式  SGD\_SSF33\_CBC - SSF33算法CBC加密模式  SGD\_SSF33\_CFB - SSF33算法CFB加密模式  SGD\_SSF33\_OFB - SSF33算法OFB加密模式  SGD\_SM4\_ECB - SM4算法ECB加密模式  SGD\_SM4\_CBC - SM4算法CBC加密模式  SGD\_SM4\_CFB - SM4算法CFB加密模式  SGD\_SM4\_OFB - SM4算法OFB加密模式  SGD\_ZUC\_EEA3 - ZUC祖冲之机密性算法128-EEA3算法 |
| pucIV | unsigned char \* | 是 | 输入，缓冲区指针，用于存放输入和返回的IV数据 |
| pucData | unsigned char \* | 是 | 输入，缓冲区指针，用于存放输入的数据明文 |
| uiDataLength | unsigned int | 是 | 输入，输入的数据明文长度 |
| pucEncData | unsigned char \* | 是 | 输出，缓冲区指针，用于存放输出的数据密文 |
| puiEncDataLength | unsigned int \* | 是 | 输出，输出的数据密文长度 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

void\* hKeyHdl = NULL;

unsigned char iv[] = "1122334455667788";

unsigned char dataPlain[1024 \* 10 + 1] = { 0 };

long long dataPlainLen = sizeof(dataPlain) - 1;

unsigned char dataEnc[1024 \* 10 + 1] = { 0 };

unsigned int dataEncLen = sizeof(dataEnc);

unsigned int algID = SGD\_SM4\_ECB;

for (int i = 0; i < dataPlainLen; i++)

dataPlain[i] = 'a';

//获取会话密钥句柄

//对称加密

int rt = SDF\_Encrypt(

hSess, hKeyHdl,

algID, iv,

dataPlain, dataPlainLen,

dataEnc, &dataEncLen);

if (rt)

{

printf("\n[algID = %s] SDF\_Encrypt failed %d | 0x%08x\n", GetAlgStr(algID), rt, rt);

return;

}

else

{

printf("\n[algID = %s] SDF\_Encrypt success\n", GetAlgStr(algID));

//printf("encData: %s\n", Bin2String(dataEnc, dataEncLen, true).data());

}

### 数据解密运算

* **功能介绍**

该接口实现密码设备使用指定的密钥句柄和IV对数据进行对称解密运算的功能。此函数不对数据进行填充处理，输入的数据必须是指定算法分组长度的整数倍。

* **接口定义**

int SDF\_Decrypt(

void\* hSessionHandle,

void\* hKeyHandle,

unsigned int uiAlgID,

unsigned char\* pucIV,

unsigned char\* pucEncData,

unsigned int uiEncDataLength,

unsigned char\* pucData,

unsigned int\* puiDataLength);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| hKeyHandle | void \* | 是 | 输入，指定的密钥句柄，若需要使用加密机内部密钥，可使用如下方式  如使用1号索引密钥，则int idx = 1; 传入 &idx 即可 |
| uiAlgID | unsigned int | 是 | 输入，算法标识，指定的对称解密算法  SGD\_SM1\_ECB - SM1算法ECB加密模式  SGD\_SM1\_CBC - SM1算法CBC加密模式  SGD\_SM1\_CFB - SM1算法CFB加密模式  SGD\_SM1\_OFB - SM1算法OFB加密模式  SGD\_SSF33\_ECB - SSF33算法ECB加密模式  SGD\_SSF33\_CBC - SSF33算法CBC加密模式  SGD\_SSF33\_CFB - SSF33算法CFB加密模式  SGD\_SSF33\_OFB - SSF33算法OFB加密模式  SGD\_SM4\_ECB - SM4算法ECB加密模式  SGD\_SM4\_CBC - SM4算法CBC加密模式  SGD\_SM4\_CFB - SM4算法CFB加密模式  SGD\_SM4\_OFB - SM4算法OFB加密模式  SGD\_ZUC\_EEA3 - ZUC祖冲之机密性算法128-EEA3算法 |
| pucIV | unsigned char \* | 是 | 输入，缓冲区指针，用于存放输入和返回的IV数据 |
| pucEncData | unsigned char \* | 是 | 输入，缓冲区指针，用于存放输入的数据密文 |
| uiEncDataLength | unsigned int | 是 | 输入，输入的数据密文长度 |
| pucData | unsigned char \* | 是 | 输出，缓冲区指针，用于存放输出的数据明文 |
| puiDataLength | unsigned int \* | 是 | 输出，输出的数据明文长度 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

void\* hKeyHdl = NULL;

unsigned char iv[] = "1122334455667788";

unsigned char dataEnc[1024 \* 10 + 1] = { 0 };

unsigned int dataEncLen = sizeof(dataEnc);

unsigned int algID = SGD\_SM4\_ECB;

unsigned char dataOut[BUF \* 10 + 1] = { 0 };

unsigned int dataOutLen = sizeof(dataOut);

//获取会话密钥句柄

//对称加密

//对称解密

int rt = SDF\_Decrypt(

hSess, hKeyHdl,

algID, iv,

dataEnc, dataEncLen,

dataOut, &dataOutLen);

if (rt)

{

printf("\n[algID = %s] SDF\_Decrypt failed %d | 0x%08x\n", GetAlgStr(algID), rt, rt);

return;

}

else

{

printf("\n[algID = %s] SDF\_Decrypt success\n", GetAlgStr(algID));

}

### 数据MAC运算

* **功能介绍**

该接口实现密码设备使用指定的密钥句柄和IV对数据进行对称MAC计算的功能。此函数不对数据进行填充处理，输入的数据必须是指定算法分组长度的整数倍。

* **接口定义**

int SDF\_CalculateMAC(

void\* hSessionHandle,

void\* hKeyHandle,

unsigned int uiAlgID,

unsigned char\* pucIV,

unsigned char\* pucData,

unsigned int uiDataLength,

unsigned char\* pucMAC,

unsigned int\* puiMACLength);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| hKeyHandle | void \* | 是 | 输入，指定的密钥句柄，若需要使用加密机内部密钥，可使用如下方式  如使用1号索引密钥，则int idx = 1; 传入 &idx 即可 |
| uiAlgID | unsigned int | 是 | 输入，算法标识，指定的对称加密算法  SGD\_SM1\_MAC - SM1算法MAC计算  SGD\_SSF33\_MAC - SSF33算法MAC计算  SGD\_SM4\_MAC - SM4算法MAC计算  SGD\_ZUC\_EEA3 - ZUC祖冲之机密性算法128-EEA3算法 |
| pucIV | unsigned char \* | 是 | 输入，缓冲区指针，用于存放输入和返回的IV数据 |
| pucData | unsigned char \* | 是 | 输入，缓冲区指针，用于存放输入的数据密文 |
| uiDataLength | unsigned int | 是 | 输入，输入的数据密文长度 |
| pucMAC | unsigned char \* | 是 | 输出，缓冲区指针，用于存放输出的MAC值 |
| puiMACLength | unsigned int \* | 是 | 输出，输出的MAC值长度 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

void\* hKeyHdl = NULL;

unsigned char iv[] = "1122334455667788";

unsigned char dataPlain[1024 \* 10 + 1] = { 0 };

unsigned int dataPlainLen = sizeof(dataPlain) - 1;

unsigned char mac[128] = { 0 };

unsigned int macLen = sizeof(mac);

unsigned int algMacID = SGD\_SM4\_MAC

for (int i = 0; i < dataPlainLen; i++)

dataPlain[i] = 'a';

//获取会话密钥句柄

//计算mac

int rt = SDF\_CalculateMAC(

hSess, hKeyHdl,

algMacID, iv,

dataPlain, dataPlainLen,

mac, &macLen);

if (rt)

{

printf("\n[algMacID = %s] SDF\_CalculateMAC failed %d | 0x%08x\n"

, GetAlgStr(algMacID), rt, rt);

rt = SDF\_DestroyKey(hSess, hKeyHdl);

if (rt)

{

printf("\nSDF\_DestroyKey failed %d | 0x%08x\n", rt, rt);

return;

}

printf("\nSDF\_DestroyKey success\n");

hKeyHdl = NULL;

return;

}

else

{

printf("\n[algMacID = %s] SDF\_CalculateMAC success\n", GetAlgStr(algMacID));

printf("mac: %s\n", Bin2String(mac, macLen, true).data());

rt = SDF\_DestroyKey(hSess, hKeyHdl);

if (rt)

{

printf("\nSDF\_DestroyKey failed %d | 0x%08x\n", rt, rt);

return;

}

printf("\nSDF\_DestroyKey success\n");

hKeyHdl = NULL;

}

## 摘要算法

### 摘要运算初始化

* **功能介绍**

该接口实现密码设备摘要算法初始化操作，返回摘要句柄的功能。此函数是三步式数据杂凑运算第一步。uiIDLength非零且uiAlgID为SGD\_SM3时，函数执行SM2的预处理1操作。计算过程见GM/T 0009。

* **接口定义**

int SDF\_HashInit(

void\* hSessionHandle,

unsigned int uiAlgID,

ECCrefPublicKey\* pucPublicKey,

unsigned char\* pucID,

unsigned int uiIDLength);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| uiAlgID | unsigned int | 是 | 输入，指定杂凑算法标识  SGD\_SM3 - SM3杂凑算法  SGD\_SHA1 - SHA\_1杂凑算法  SGD\_SHA256 - SHA\_256杂凑算法 |
| pucPublicKey | ECCrefPublicKey \* | 否 | 输入，签名者公钥。  仅当uiAlgID位SGD\_SM3时有效 |
| pucID | unsigned char \* | 否 | 输入，签名者的ID值。  仅当uiAlgID位SGD\_SM3时有效 |
| uiIDLength | unsigned int \* | 否 | 输入，签名者ID长度。  仅当uiAlgID位SGD\_SM3时有效 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

ECCrefPublicKey pubKey = { 0 };

unsigned char ID[1024] = { 0 };

unsigned int IDLen = 0;

unsigned char data[1024 \* 10 + 1] = { 0 };

unsigned int dataLen = sizeof(data) - 1;

unsigned int algID = SGD\_SM3；

for (int i = 0; i < dataLen; i++)

data[i] = 'a';

int rt = SDF\_HashInit(hSess, algID, &pubKey, ID, IDLen);

if (rt)

{

printf("\n[algID = %s] SDF\_HashInit failed %d | 0x%08x\n"

, GetAlgStr(algID), rt, rt);

return;

}

else

printf("\n[algID = %s] SDF\_HashInit success\n", GetAlgStr(algID));

### 摘要Update运算

* **功能介绍**

该接口实现密码设备摘要算法update操作的功能。此函数是三步式数据杂凑运算第二步，对输入的明文进行杂凑运算。

* **接口定义**

int SDF\_HashUpdate(

void\* hSessionHandle,

unsigned char\* pucData,

unsigned int uiDataLength);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| pucData | unsigned char \* | 是 | 输入，缓冲区指针，用于存放输入的数据明文 |
| uiDataLength | unsigned int | 是 | 输入，输入的数据明文长度 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

unsigned char data[1024 \* 10 + 1] = { 0 };

unsigned int dataLen = sizeof(data) - 1;

for (int i = 0; i < dataLen; i++)

data[i] = 'a';

int rt = 0；

for (int i = 0; i < 3; i++)

{

rt = SDF\_HashUpdate(hSess, data, dataLen);

if (rt)

{

printf("\n[algID = %s] SDF\_HashUpdate failed %d | 0x%08x\n"

, GetAlgStr(algID), rt, rt);

return;

}

else

printf("\n[algID = %s] SDF\_HashUpdate success\n"

, GetAlgStr(algID));

}

### 摘要运算结束

* **功能介绍**

该接口实现密码设备杂凑运算结束返回杂凑数据并清除中间数据的功能。此函数是三步式数据杂凑运算第二步，杂凑运算结束返回杂凑数据并清除中间数据。

* **接口定义**

int SDF\_HashFinal(

void\* hSessionHandle,

unsigned char\* pucHash,

unsigned int\* puiHashLength);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| pucHash | unsigned char \* | 是 | 输出，缓冲区指针，用于存放输出的杂凑数据 |
| puiHashLength | unsigned int \* | 是 | 输出，返回的杂凑数据长度 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

unsigned char hash[1024] = { 0 };

unsigned int hashLen = sizeof(hash);

int rt = SDF\_HashFinal(hSess, hash, &hashLen);

if (rt)

{

printf("\n[algID = %s] SDF\_HashFinal failed %d | 0x%08x\n"

, GetAlgStr(algID), rt, rt);

return;

}

else

{

printf("\n[algID = %s] SDF\_HashFinal success\n"

, GetAlgStr(algID));

printf("\nhash: %s\n", Bin2String(hash, hashLen, true).data());

}

## 文件操作

### 创建用户数据文件

* **功能介绍**

该接口实现在密码设备内部创建用于存储用户数据的文件的功能。

* **接口定义**

int SDF\_CreateFile(

void\* hSessionHandle,

unsigned char\* pucFileName,

unsigned int uiNameLen,

unsigned int uiFileSize);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| pucFileName | unsigned char \* | 是 | 输入，缓冲区指针，用于存放输入的文件名，最大长度128字节 |
| uiNameLen | unsigned int | 是 | 输入，文件名长度 |
| uiFileSize | unsigned int | 是 | 输入，文件所占存储空间的长度 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

unsigned char fileName[] = "test1507";

unsigned int fileNameLen = strlen((char\*)fileName);

unsigned int fileSize = 8192;

int rt = SDF\_CreateFile(hSess, fileName, fileNameLen, fileSize);

if (rt)

{

printf("\n[fileSize = %d] SDF\_CreateFile failed %d | 0x%08x\n"

, fileSize, rt, rt);

return;

}

else

printf("\n[fileSize = %d] SDF\_CreateFile success\n", fileSize);

### 读取用户数据文件

* **功能介绍**

该接口实现读取在密码设备内部存储用户数据的文件的内容的功能。

* **接口定义**

int SDF\_ReadFile(

void\* hSessionHandle,

unsigned char\* pucFileName,

unsigned int uiNameLen,

unsigned int uiOffset,

unsigned int\* puiFileLength,

unsigned char\* pucBuffer);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| pucFileName | unsigned char \* | 是 | 输入，缓冲区指针，用于存放输入的文件名，最大长度128字节 |
| uiNameLen | unsigned int | 是 | 输入，文件名长度 |
| uiOffset | unsigned int | 是 | 输入，指定读取文件时的偏移值 |
| puiFileLength | unsigned int \* | 是 | 输入/输出，入参时指定读取文件内容的长度；  出参时返回实际读取文件内容的长度 |
| pucBuffer | unsigned char \* | 是 | 输出，缓冲区指针，用于存放读取的文件数据 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

unsigned char fileName[] = "test1507";

unsigned int fileNameLen = strlen((char\*)fileName);

unsigned char dataOut[1024 \* 8 + 1] = { 0 };

unsigned int dataOutLen = 1024\*4;

//打开或创建文件

//读文件

rt = SDF\_ReadFile(hSess, fileName, fileNameLen, 0, &dataOutLen, dataOut);

if (rt)

{

printf("\n[dataLen = %d] SDF\_WriteFile failed %d | 0x%08x\n"

, dataOutLen, rt, rt);

return;

}

else

{

printf("\n[dataOutLen = %d] SDF\_ReadFile success\n", dataOutLen);

printf("\n[dataOutLen = %d] dataOut : %s\n", dataOutLen, dataOut);

}

### 写入用户数据文件

* **功能介绍**

该接口实现向密码设备内部存储用户数据的文件中写入内容的文件的功能。

* **接口定义**

int SDF\_WriteFile(

void\* hSessionHandle,

unsigned char\* pucFileName,

unsigned int uiNameLen,

unsigned int uiOffset,

unsigned int uiFileLength,

unsigned char\* pucBuffer);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| pucFileName | unsigned char \* | 是 | 输入，缓冲区指针，用于存放输入的文件名，最大长度128字节 |
| uiNameLen | unsigned int | 是 | 输入，文件名长度 |
| uiOffset | unsigned int | 是 | 输入，指定写入文件时的偏移值 |
| uiFileLength | unsigned int | 是 | 输入，指定写入文件内容的长度 |
| pucBuffer | unsigned char \* | 是 | 输入，缓冲区指针，用于存放输入的写文件数据 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

unsigned char fileName[] = "test1507";

unsigned int fileNameLen = strlen((char\*)fileName);

unsigned char data[1024 \* 4 + 1] = { 0 };

unsigned int dataLen = sizeof(data) - 1;

for (int i = 0; i < dataLen; i++)

data[i] = 'a';

//打开或创建文件

//写文件

int rt = SDF\_WriteFile(

hSess, fileName,

fileNameLen, 0,

dataLen, data);

if (rt)

{

printf("\n[dataLen = %d] SDF\_WriteFile failed %d | 0x%08x\n"

, dataLen, rt, rt);

return;

}

else

printf("\n[dataLen = %d] SDF\_WriteFile success\n", dataLen);

### 删除用户数据文件

* **功能介绍**

该接口实现在密码设备内部删除用于存储用户数据的文件的功能。

* **接口定义**

int SDF\_DeleteFile(

void\* hSessionHandle,

unsigned char\* pucFileName,

unsigned int uiNameLen);

* **参数描述**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **是否必需** | **描述** |
| hSessionHandle | void \* | 是 | 输入，与密码设备已建立的会话句柄 |
| pucFileName | unsigned char \* | 是 | 输入，缓冲区指针，用于存放输入的文件名，最大长度128字节 |
| uiNameLen | unsigned int | 是 | 输入，文件名长度 |

* **返回参数**

成功返回0，失败返回非0。

* **接口示例**

unsigned char fileName[] = "test1507";

unsigned int fileNameLen = strlen((char\*)fileName);

//打开或创建文件

//删除文件

int rt = SDF\_DeleteFile(hSess, fileName, fileNameLen);

if (rt)

{

printf("SDF\_DeleteFile %s failed %d | 0x%08x\n"

, fileName, rt, rt);

}

else

printf("SDF\_DeleteFile %s success\n", fileName);

# 错误码

## SDF标准错误码

|  |  |  |
| --- | --- | --- |
| 错误代码标识 | | |
| 宏描述 | 预定义值 | 说明 |
| #define SDR\_OK | 0X0 | 操作成功 |
| #define SDR\_BASE | 0X01000000 | 错误码基础值 |
| #define SDR\_UNKNOWERR | SDR\_BASE + 0X00000001 | 未知错误 |
| #define SDR\_NOTSUPPORT | SDR\_BASE + 0X00000002 | 不支持的接口调用 |
| #define SDR\_COMMFAIL | SDR\_BASE + 0X00000003 | 与设备通讯失败 |
| #define SDR\_HARDFAIL | SDR\_BASE + 0X00000004 | 运算模块无响应 |
| #define SDR\_OPENDEVICE | SDR\_BASE + 0X00000005 | 打开设备失败 |
| #define SDR\_OPENSESSION | SDR\_BASE + 0X00000006 | 创建会话失败 |
| #define SDR\_PARDENY | SDR\_BASE + 0X00000007 | 无私钥使用权限 |
| #define SDR\_KEYNOTEXIST | SDR\_BASE + 0X00000008 | 不存在的密钥调用 |
| #define SDR\_ALGNOTSUPPORT | SDR\_BASE + 0X00000009 | 不支持的算法调用 |
| #define SDR\_ALGMODNOTSUPPORT | SDR\_BASE + 0X0000000A | 不支持的算法模式调用 |
| #define SDR\_PKOPERR | SDR\_BASE + 0X0000000B | 公钥运算失败 |
| #define SDR\_SKOPERR | SDR\_BASE + 0X0000000C | 私钥运算失败 |
| #define SDR\_SIGNERR | SDR\_BASE + 0X0000000D | 签名运算失败 |
| #define SDR\_VERIFYERR | SDR\_BASE + 0X0000000E | 验证签名失败 |
| #define SDR\_SYMOPERR | SDR\_BASE + 0X0000000F | 对称算法运算失败 |
| #define SDR\_STEPERR | SDR\_BASE + 0X00000010 | 多步运算步骤错误 |
| #define SDR\_FILESIZEERR | SDR\_BASE + 0X00000011 | 文件长度超出限制 |
| #define SDR\_FILENOEXIST | SDR\_BASE + 0X00000012 | 指定的文件不存在 |
| #define SDR\_FILEOFSERR | SDR\_BASE + 0X00000013 | 文件起始位置错误 |
| #define SDR\_KEYTYPEERR | SDR\_BASE + 0X00000014 | 密钥类型错误 |
| #define SDR\_KEYERR | SDR\_BASE + 0X00000015 | 密钥错误 |
| #define SDR\_ENCDATAERR | SDR\_BASE + 0X00000016 | ECC加密数据错误 |
| #define SDR\_RANDERR | SDR\_BASE + 0X00000017 | 随机数产生失败 |
| #define SDR\_PRKRERR | SDR\_BASE + 0X00000018 | 私钥使用权限获取失败 |
| #define SDR\_MACERR | SDR\_BASE + 0X00000019 | MAC运算失败 |
| #define SDR\_FILEEXISTS | SDR\_BASE + 0X0000001A | 指定文件已存在 |
| #define SDR\_FILEWERR | SDR\_BASE + 0X0000001B | 文件写入失败 |
| #define SDR\_NOBUFFER | SDR\_BASE + 0X0000001C | 存储空间不足 |
| #define SDR\_INARGERR | SDR\_BASE + 0X0000001D | 输入参数错误 |
| #define SDR\_OUTARGERR | SDR\_BASE + 0X0000001E | 输出参数错误 |