商用PCI-E密码卡

用户手册

成都卫士通信息产业股份有限公司

二○一五年三月制

**目录**

[商用PCI-E密码卡 1](#_Toc415498151)

[用户手册 1](#_Toc415498152)

[1 产品概述 1](#_Toc415498153)

[2 设备清单 1](#_Toc415498154)

[3 产品介绍 1](#_Toc415498155)

[3.1主要功能 1](#_Toc415498156)

[3.2密码体制 2](#_Toc415498157)

[3.3设备性能 2](#_Toc415498158)

[3.4环境要求 2](#_Toc415498159)

[3.5物理特性 3](#_Toc415498160)

[3.6电气特性 3](#_Toc415498161)

[4 设备安装 3](#_Toc415498162)

[4.1安装条件 3](#_Toc415498163)

[4.2设备面板示意图 3](#_Toc415498164)

[4.3设备状态灯 4](#_Toc415498165)

[4.4设备按钮 4](#_Toc415498166)

[4.5硬件安装方法 4](#_Toc415498167)

[4.6软件安装方法 5](#_Toc415498168)

[4.6.1Windows /2000 /XP/2003、Win7系统 5](#_Toc415498169)

[4.6.2Linux系统安装/卸载 8](#_Toc415498170)

[5 API 5.2软件系统架构及典型使用过程 8](#_Toc415498171)

[5.1设备访问 9](#_Toc415498172)

[5.2安全管道访问 10](#_Toc415498173)

[5.2.1三段式结构 10](#_Toc415498174)

[5.3密钥访问 14](#_Toc415498175)

[5.3.1对称密钥管理 14](#_Toc415498176)

[5.3.2非对称密钥管理 14](#_Toc415498177)

[6 API 5.2应用参考 15](#_Toc415498178)

[6.1数据类型 15](#_Toc415498179)

[6.2数据结构 15](#_Toc415498180)

[6.2.1 SM\_MECHANISM\_INFO 15](#_Toc415498181)

[6.2.2 SM\_ALGORITHM 18](#_Toc415498182)

[6.2.3 SM\_KEY\_ATTRIBUTE 19](#_Toc415498183)

[6.2.4 SM\_ECC\_PARAMETER 23](#_Toc415498184)

[6.2.5SM\_BLOB\_KEY 24](#_Toc415498185)

[6.2.6SM\_BLOB\_CONTEXT 25](#_Toc415498186)

[6.2.7 SM\_BLOB\_ECCCIPHER 26](#_Toc415498187)

[6.2.8 SM\_DEVICE\_INFO 27](#_Toc415498188)

[6.3接口函数 32](#_Toc415498189)

[6.3.1设备访问接口函数 32](#_Toc415498190)

[6.3.1.1 SM\_GetDeviceNum 33](#_Toc415498191)

[6.3.1.2 SM\_GetErrorString 34](#_Toc415498192)

[6.3.1.3SM\_GetAPIVersion 34](#_Toc415498193)

[6.3.1.4 SM\_GetDeviceType 35](#_Toc415498194)

[6.3.1.5 SM\_OpenDevice 36](#_Toc415498195)

[6.3.1.6 SM\_CloseDevice 38](#_Toc415498196)

[6.3.1.7 SM\_GetMechanismList 39](#_Toc415498197)

[6.3.1.8 SM\_GetMechanismInfo 42](#_Toc415498198)

[6.3.1.9 SM\_GetDeviceIndex 43](#_Toc415498199)

[6.3.1.10 SM\_TestDevice 45](#_Toc415498200)

[6.3.1.11SM\_GetDeviceInfo 47](#_Toc415498201)

[6.3.1.12 SM\_LockMem 48](#_Toc415498202)

[6.3.1.13 SM\_UnlockMem 51](#_Toc415498203)

[6.3.1.14 SM\_WriteNonVolatile 52](#_Toc415498204)

[6.3.1.15 SM\_ReadNonVolatile 54](#_Toc415498205)

[6.3.1.16 SM\_BuildAuthDev 55](#_Toc415498206)

[6.3.1.17 SM\_ChangeUserPin 57](#_Toc415498207)

[6.3.1.18 SM\_BackupAuthDev 58](#_Toc415498208)

[6.3.2安全管道访问接口函数 59](#_Toc415498209)

[6.3.2.1 SM\_OpenSecPipe 61](#_Toc415498210)

[6.3.2.2 SM\_CloseSecPipe 62](#_Toc415498211)

[6.3.2.3 SM\_EncryptInit 63](#_Toc415498212)

[6.3.2.4 SM\_EncryptUpdate 66](#_Toc415498213)

[6.3.2.5 SM\_EncryptFinal 67](#_Toc415498214)

[6.3.2.6 SM\_Encrypt 69](#_Toc415498215)

[6.3.2.7 SM\_DecryptInit 71](#_Toc415498216)

[6.3.2.8 SM\_DecryptUpdate 74](#_Toc415498217)

[6.3.2.9 SM\_DecryptFinal 75](#_Toc415498218)

[6.3.2.10 SM\_Decrypt 77](#_Toc415498219)

[6.3.2.11 SM\_DigestInit 80](#_Toc415498220)

[6.3.2.12 SM\_DigestUpdate 82](#_Toc415498221)

[6.3.2.13 SM\_DigestFinal 83](#_Toc415498222)

[6.3.2.14 SM\_Digest 85](#_Toc415498223)

[6.3.2.15 SM\_ECCSignature 87](#_Toc415498224)

[6.3.2.16 SM\_ECCVerify 90](#_Toc415498225)

[6.3.2.17 SM\_GenRandom 91](#_Toc415498226)

[6.3.2.18 SM\_ECCEncrypt 92](#_Toc415498227)

[6.3.2.19 SM\_ECCDecrypt 95](#_Toc415498228)

[6.3.2.20 SM\_CloseAllSecPipe 96](#_Toc415498229)

[6.3.2.21 SM\_Login 96](#_Toc415498230)

[6.3.2.22 SM\_Logout 98](#_Toc415498231)

[6.3.3密钥访问管理接口函数 99](#_Toc415498232)

[6.3.3.1 SM\_ImportKey 99](#_Toc415498233)

[6.3.3.2 SM\_ExportKey 101](#_Toc415498234)

[6.3.3.2SM\_DestroyKey 103](#_Toc415498235)

[6.3.3.3 SM\_ImportPrivateKey 104](#_Toc415498236)

[6.3.3.4 SM\_ImportPublicKey 107](#_Toc415498237)

[6.3.3.5 SM\_DestroyPrivateKey 108](#_Toc415498238)

[6.3.3.6 SM\_DestroyPublicKey 109](#_Toc415498239)

[6.3.3.8 SM\_GenerateKey 110](#_Toc415498240)

[6.3.3.9 SM\_GenerateKeyPair 113](#_Toc415498241)

[6.3.3.10 SM\_ExportPrivateKey 118](#_Toc415498242)

[6.3.3.11 SM\_ExportPublicKey 121](#_Toc415498243)

[6.3.3.12 SM\_UpdateKeyPair 123](#_Toc415498244)

[6.3.3.13SM\_GetCfgKeyHandle 125](#_Toc415498245)

[6.3.3.14SM\_CloseTokKeyHdl 126](#_Toc415498246)

[6.4 错误代码 127](#_Toc415498247)

# 产品概述

商用PCI-E密码卡（以下简称密码卡）通常嵌入服务器或台式计算机内部，为其宿主设备提供各种密码服务。密码卡可实现数据加/解密、用户身份识别等安全功能。密码卡提供了丰富的安全服务接口供用户业务系统使用，其服务接口遵循GM/T 0018-2012《密码设备应用接口规范》。

密码卡支持Windows2000/XP/2003、Windows7、Linux等操作系统。在套件光盘中附带有功能完善的API软件包及应用测试程序，便于用户根据实际应用进行二次开发。

# 设备清单

一套完整的高速商用PCI-E密码卡包括：

|  |  |  |
| --- | --- | --- |
| **设备名称** | **数量** | **主要功能** |
| 商用PCI-E密码卡 | 1 | 完成数据加/解密、数字签名等功能 |
| 智能USB锁 | 1 | 密码卡安全认证及存储 |
| 光盘 | 1 | 内含常用操作系统驱动程序、API等支持软件以及操作手册。 |

**注：本清单仅提供参考，具体以包装箱中的装箱清单为准。**

# 产品介绍

## 3.1主要功能

密码卡能为系统提供数字签名/验证、数据加密/解密和信息完整性保护，同时也能为访问用户提供身份鉴别功能。其主要功能如下：

1. 基于密钥分层保护的密钥管理；
2. 提供数据加密/解密；
3. 提供数字签名/验证；
4. 提供杂凑运算；
5. 提供消息鉴别码（MAC）的产生和验证；
6. 提供用户身份鉴别；
7. 支持真随机数产生。

## 3.2密码体制

* 完备的身份鉴别方式
* 完整的安全保密体系结构
* 全面支持国家密码管理局批准使用的SM1/SM2/SM3/SM4密码算法
* SM1/SM4算法均支持ECB、CBC以及MAC三种工作模式
* 完善的密钥分层保护机制

## 3.3设备性能

* SM1加解密速率不低于150Mbps
* SM2签名速率不低于1800次/秒
* SM2验证速率不低于1300次/秒
* SM3摘要计算速率不低于270Mbps
* SM4加解密速率不低于250MGbps
* 真随机数产生速率：5Mbps

## 3.4环境要求

* 工作温度：0℃ - 40℃
* 贮存温度：-20℃ - 55℃
* 工作湿度：20%-80%

## 3.5物理特性

* 尺寸153mm×119mm×21.5mm（板卡，屏蔽盖和弯角件）；140mm×92.5mm×10mm（板卡）
* 质量：500g

## 3.6电气特性

* 功耗≤10W

# 设备安装

## 4.1安装条件

安装密码卡前，请确认满足以下条件：

1. 接收设备与装箱清单明细对应且完好无损；
2. 一台具备空闲PCI-E x1以上插槽的台式计算机或服务器；
3. 主机操作系统为密码卡支持的操作系统（Win2000/XP/2003、Win7以及Linux等）；

## 4.2设备面板示意图



图 1加密机面板示意图

A 销毁按钮 B 状态指示灯 C 电源指示灯 D 加密指示灯

E 销毁指示灯 F 智能USB锁接口

## 4.3设备状态灯

密码卡的启动和工作状态，可以通过前面板的LED指示灯查看。LED灯描述如下：

表 1状态灯指示表

|  |  |
| --- | --- |
| **LED灯** | **描述** |
| 电源指示灯 | 长亮表示设备电源已正常工作 |
| 状态指示灯 | 闪烁表示设备正在复位过程中  长亮表示设备已启动进入工作状态  熄灭表示设备在工作中遇到异常指令 |
| 加密指示灯 | 闪烁表示设备正在与计算机通信  熄灭表示设备正处于空闲状态 |
| 销毁指示灯 | 闪烁表示设备正处于毁钥过程  熄灭表示设备未进行毁钥操作 |

## 4.4设备按钮

设备有一个销毁按钮，通过点触销毁按钮并保持2秒以上，设备将进行毁钥操作，同时销毁指示灯将闪烁。

## 4.5硬件安装方法

1. 将密码卡正确插入计算机的空闲PCI-E插槽，并将其固定；
2. 将智能USB锁与密码卡正确连接；
3. 此时即完成密码卡的硬件安装，启动电脑即可在系统中发现

密码卡设备。

## 4.6软件安装方法

商用PCI-E密码卡支持Win2000/XP/2003、Win7以及Linux等操作系统的访问使用。其软件部分安装分为硬件驱动安装和API安装两部分。

系统需求：该软件包适用于Microsoft Windows 2000、Windows XP、Windows2003Windows7以及Linux等操作系统平台下，奔腾166MHz或更快，64 MB内存或更多，16MB磁盘空间。

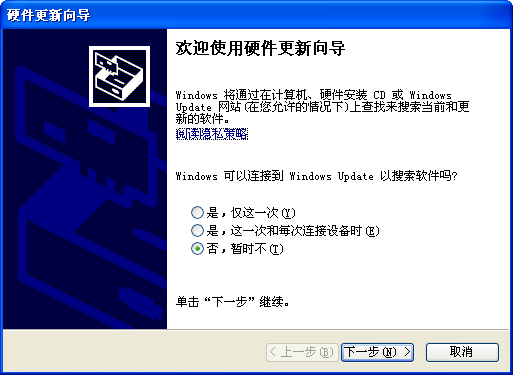
### 4.6.1Windows /2000 /XP/2003、Win7系统

1.硬件驱动程序安装

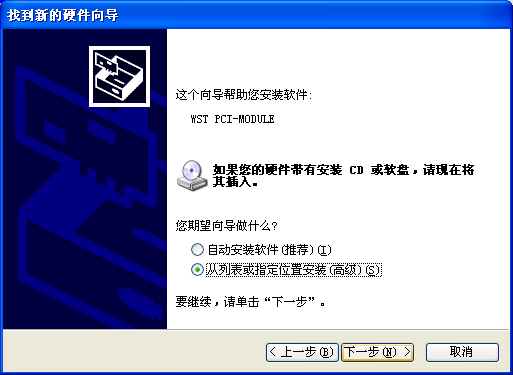
1)启动计算机，请确保在开机前已正确插入PCI-E卡；

2)第一次使用时，可根据操作系统的提示装载驱动程序，驱动程序在光盘中“win32(或win64)/drv”目录下，如果曾经安装过驱动程序，现在希望进行驱动更新，可进入设备管理器中进行更新；

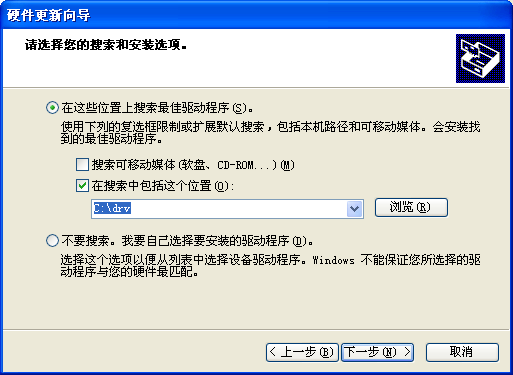
3)如需要卸载该设备硬件驱动程序，请执行操作系统自带的“设备管理器”，找到硬件设备，右键选择“更新驱动程序”，打开“硬件更新向导”，选择“否”，单击“下一步”；



选择从“列表或者指定位置安装（高级）”，单击“下一步”；



单击“{浏览”按钮，选择驱动所在目录，点击“下一步”；



如果安装失败，选择“不要搜索，我要自己选择要安装的驱动程序”，单击“下一步”，选择“从磁盘安装”，找到驱动所在目录，选择驱动的inf文件，打开后选择“下一步”完成安装。



2. API

请自行将光盘中的“win32(或win64)/api”目录复制到用户指定的目录中即可。

### 4.6.2Linux系统安装/卸载

1. 硬件驱动

1)根据自己计算机中使用的Linux内核版本号选择对应的驱动程序目录，这里，以内核2.4.20-8单CPU版本为例；

2)将光盘中“drivers/linux/2-4-20.8up”目录复制到用户指定目录中；

3)在用户指定目录中运行“./installssf”即完成硬件驱动安装操作；

4)如需要卸载该设备硬件驱动程序，请在用户指定目录中运行“rmmod xxx.o”,即完成硬件驱动卸载操作。

注：xxx.o为相应的驱动程序执行文件。

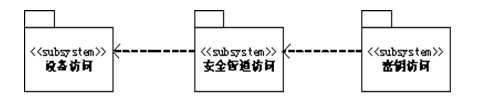
2. API

1)请自行将光盘中的“api/linux”目录复制到用户指定的目录中；

# API 5.2软件系统架构及典型使用过程

安全模块API 5.2软件系统是以动态链接库（DLL、SO）的方式提供给用户使用，任何支持Win32 DLL，Win64 DLL，LINUX的编程软件均可使用该API软件系统。

该软件系统架构由以下几部分组成（如图示）：



用户首先通过[设备访问](Device.htm)获得对安全模块硬件设备的访问权限，读取设备的基本信息。接着，通过[安全管道访问](SecPipe.htm)可实现数据加解密、数据块散列、数据块鉴别等安全功能的应用。而对于对称密钥和非对称密钥的管理则可通过[密钥访问](KeyManage.htm)实现。

相关说明文档：[API 5.2应用参考](APIReference.htm)，[设备访问](Device.htm)，[安全管道访问](SecPipe.htm)，[密钥访问](KeyManage.htm)

## 5.1设备访问

设备访问是整个安全模块API 5.2的基础子系统。用户在使用该API 5.2时，首先必须使用该系统获得对硬件设备的访问权限，读取设备的基本信息。之后，才能够继续后续的[安全管道访问](SecPipe.htm)和[密钥访问](KeyManage.htm)操作。

通常，一台主机最多支持8个安全模块硬件设备。用户在使用[SM\_OpenDevice](SM_OpenDevice.htm)函数打开指定安全模块硬件设备时，通过参数uiDevID与指定的硬件设备相关联。主机中存在的安全模块硬件设备有效数量可通过函数[SM\_GetDeviceNum](SM_GetDeviceNum.htm)获取。

整个API 5.2接口函数调用时，如果返回值为SM\_ERR\_FREE，表示该函数调用成功；否则调用失败，相应的错误信息可调用[SM\_GetErrorString](SM_GetErrorString.htm)函数获取，具体错误代码参阅[API错误代码表](errorcode.htm)。

相关说明文档：[API 5.2软件系统架构及典型使用过程](APISystem.htm)，[安全管道访问](SecPipe.htm)，[密钥访问](KeyManage.htm)，[设备访问接口函数](Device_Fun.htm)

## 5.2安全管道访问

安全管道访问是负责除密钥管理外的安全功能访问操作的子系统。通过访问该子系统，用户可实现数据加解密、数据块摘要、数据签名/验签等安全功能的应用。

三段式结构：API 5.2软件将对称算法加解密运算、数据块摘要运算定义为三段式结构，具体说明请参见[三段式结构](ThreePart.htm)。

相关说明文档：[API 5.2软件系统架构及典型使用过程](APISystem.htm)，[设备访问](Device.htm)，[密钥访问](KeyManage.htm)，[安全管道访问接口函数](SecPipe_Fun.htm)

### 5.2.1三段式结构

本系统将对称算法加解密运算函数、数据块摘要函数设计为三段式结构函数。

三段式结构分别由SM\_xxxInit、SM\_xxxUpdate、SM\_xxxFinal三个接口函数组成。

|  |
| --- |
| **SM\_xxxInit**  **SM\_xxxUpdate（可选择性多次调用）**  **SM\_xxxFinal** |

**A．成员**

**SM\_xxxInit**

该函数用于三段式结构运算初始化操作。在每进行一轮三段式结构运算时，首先调用SM\_xxxInit进行运算初始化。一旦调用此函数成功后，则进行新一轮三段式结构运算。

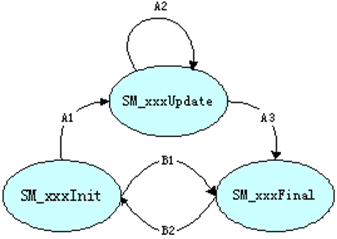
**SM\_xxxUpdate**

该函数用于三段式结构运算中间数据运算操作。这里，当参与三段式运算的数据流较大时（大于1M字节），推荐将数据流根据三段式结构运算分组长度的大小，进行整数倍分块后重复调用该函数，以提高设备的处理效率（图示A流程）；而对于数据流较小的运算（小于1M字节），则可直接调用SM\_xxxFinal函数（函数内部将自动调用SM\_xxxUpdate函数，图示B流程）。

**SM\_xxxFinal**

该函数用于三段式结构运算最后一组数据运算操作，数据块摘要函数将输出结果。该函数当参数pDataOut（SM\_HashFinal函数为*pHashValue*，SM\_MacFinal函数为*pMacValue*）为空时，参数puiDataOutLen（SM\_HashFinal函数为*puiHashLen*，SM\_MacFinal函数为*puiMacValLen*）将返回输出数据所需的长度。这里，推荐在调用SM\_xxxInit函数后，推荐先调用该函数获取输出数据所需的长度后，创建足够的输出数据存储单元后再调用SM\_xxxFinal函数进行运算。

**B．调用流程**



**C．示例**

本示例以mac运算为例，说明三段式结构函数的调用过程。

**S**M**\_DEVICE\_HANDLE hDevice;/\*为打开硬件设备获得的安全模块设备访问句柄\*/**

**SM\_PIPE\_HANDLE hPipe;/\*为打开安全管道获得的安全管道访问句柄 \*/**

**SM\_ALGORITHM stAlgorithm;**

**SM\_BYTE byDataSrc[1024] = {0}; /\*输入数据，假设长度为1024字节\*/**

**PSM\_BYTE pbyDataDst = SM\_NULL; /\*输出结果\*/**

**PSM\_BYTE pIn = SM\_NULL;**

**SM\_WORD i, wRet,wTryNum;**

**SM\_WORD wLoopTime, wLeftLen, wOutLen;**

**SM\_BYTE byMacIV[SM\_ALG34\_IV\_LEN];/\*算法ALG34做MAC运算\*/**

**SM\_BYTE byContext[SMMA\_ALG34\_CTX\_LEN] = {0};**

**//1.OpenDevice...................................**

**wRet = SM\_OpenDevice(..., &hDevice);**

**//2.OpenSecPipe..................................**

**wRet = SM\_OpenSecPipe(hDevice, &hPipe);**

**//3.Login.......... ....... .....................**

**wRet = SM\_Login(hPipe,"00000000",8,&wTryNum);**

**...**

**//4.Set Source Data..............................**

**pIn = byDataSrc; /\*设置数据操作指针\*/**

**for(i=0; i<1024; ++i )**

**byDataSrc [i] = (SM\_BYTE)i;**

**//5.Init.........................................**

**Algorithm.AlgoType = SM\_ALG34\_MAC; /\*采用对称算法ALG34做Mac运算\*/**

**Algorithm.pParameter = byMacIv;**

**Algorithm.ulParameterLen = SM\_ALG34\_IV\_LEN;**

**Algorithm.ulReserve = 0;**

**wRet = SM\_DigestInit(m\_hPipe, SM\_NULL, &Algorithm, &byContext);**

**if(wRet != SM\_ERR\_FREE)**

**goto LB\_OPEREND;**

**//6.Update.......................................**

**wLoopTime = 1024 / SM\_ALG34\_BLOCK\_LEN - 1; /\*取循环轮数\*/**

**for(i=0; i<wLoopTime; ++I)**

**{**

**wRet = SM\_DigestUpdate(m\_hPipe, &byContext, pIn, SM\_ALG34\_BLOCK\_LEN);**

**if(wRet != SM\_ERR\_FREE)**

**goto LB\_OPEREND;**

**pIn += SM\_ALG34\_BLOCK\_LEN;**

**}//for**

**//7.Final........................................**

**wLeftLen = byDataSrc + 1024 - pIn;**

**//提取输出数据长度**

**wRet = SM\_DigestFinal(m\_hPipe, 1024, &byContext, pIn, wLeftLen, SM\_NULL, &wOutLen);**

**if(wRet != SM\_ERR\_FREE)**

**goto LB\_OPEREND;**

**pbyDataDst = (SM\_BYTE \*)malloc(wOutLen);**

**wRet = SM\_DigestFinal(m\_hPipe, 1024, &byContext, pIn, wLeftLen, pbyDataDst, &wOutLen);**

**if(wRet != SM\_ERR\_FREE)**

**{**

**free(pbyDataDst);**

**pbyDataDst = SM\_NULL;**

**goto LB\_OPEREND;**

**}**

**……**

**LB\_OPEREND:**

**if ( pbyDataDst != SM\_NULL )**

**free(pbyDataDst);**

**SM\_Logout(hPipe);  
SM\_CloseSecPipe(hPipe);**

**SM\_CloseDevice(hUSBhandle);**

**hDevice = hPipe = SM\_NULL;**

**if(wRet != SM\_ERR\_FREE)**

**printf("Error: %d, %s", wRet, (SM\_CHAR\*)SM\_GetErrorString(wRet));**

**return;**

**D．备注**

（1）该结构函数通常需成功登录后方可使用。

（2）对于每种三段式结构操作，其对应的三个接口函数现通过数据上下文进行关联，所以允许任一线程将正在进行同一种三段式结构运算(且hPipe相同)的操作中途打断。

（3）调用SM\_xxxInit函数表示新一轮三段式结构运算开始，而调用SM\_xxxFinal函数则表示此轮三段式结构运算结束。一旦调用SM\_xxxFinal函数后，下一轮三段式结构运算则必须重新调用SM\_xxxInit函数。

（4）调用SM\_xxxUpdate函数时，其输入数据长度必须为三段式结构运算分组长度的整数倍，否则调用失败。

## 5.3密钥访问

密钥访问是基于[安全管道访问](SecPipe.htm)子系统，负责设备对称密钥和非对称密钥管理的子系统。通过访问该子系统，用户可实现密钥的创建、导入、导出以及销毁等安全管理应用。

### 5.3.1对称密钥管理

API 5.2软件支持的对称加解密算法为 ALG34, ALG35等，您使用的模块具体支持哪种算法，请参考头文件"sm\_algo.h"说明。使用ECB模式和CBC模式，支持生成相应算法对称密钥的生成、导入、导出和销毁管理。其中，导入、导出操作支持四种保护模式，而密钥生成、存储则在计算机内存中完成。当设备退出访问时，软件自动清除计算机指定内存中的相关数据，TOKEN对象除外。

### 5.3.2非对称密钥管理

API5.2软件支持的非对称加解密算法可通过调用函数[SM\_GetMechanismInfo](SM_GetMechanismInfo.htm)获取相关信息，您使用的模块具体支持模长多少位的公私钥对，请参考头文件"sm\_algo.h"说明。所有私钥的访问操作均必须经过经过用户登录后方可使用，而公钥的访问操作则无需用户登录即可使用。

相关说明文档：[API 5.2软件系统架构及典型使用过程](APISystem.htm)，[设备访问](Device.htm)，[安全管道访问](SecPipe.htm)[密钥访问接口函数](KeyManage_Fun.htm)。

# API 5.2应用参考

安全模块API 5.2的应用参考分以下几部分：[数据类型](DataType.htm)，[数据结构](DataStructure.htm)，[接口函数](Functions.htm)。

相关说明文档：[API 5.2软件系统架构及典型使用过程](APISystem.htm)

## 6.1数据类型

安全模块API 5.2版本支持的数据类型的详细定义请参见头文件" sm\_api\_type.h"。

## 6.2数据结构

安全模块API 5.2版本支持的数据结构定义如下：

|  |  |
| --- | --- |
| 结构体定义 | 结构体说明 |
| SM\_MECHANISM\_INFO | 机制信息 |
| SM\_ALGORITHM | 算法类型 |
| SM\_KEY\_ATTRIBUTE | 密钥属性 |
| SM\_ECC\_PARAMETER | ECC密钥访问属性 |
| SM\_BLOB\_KEY | 密钥BLOB信息 |
| SM\_BLOB\_CONTEXT | 上下文BLOB信息 |

### 6.2.1 SM\_MECHANISM\_INFO

安全模块的机制信息。

typedef struct \_SM\_MECHANISM\_INFO

{

SM\_UINT uiMinBlockSize;

SM\_UINT uiMaxBlockSize;

SM\_UINT uiMinKeySize;

SM\_UINT uiMaxKeySize;

SM\_UINT uiFlags;

}SM\_MECHANISM\_INFO, \*PSM\_MECHANISM\_INFO;

**参数**

***uiMinBlockSize***

[out] 安全模块提供的指定机制的最小分组长度，以字节为单位；对于非对称算法，该参数固定取值为0。

***uiMaxBlockSize***

[out] 安全模块提供的指定机制的最大分组长度，以字节为单位；对于非对称算法，该参数固定取值为0。

***uiMinKeySize***

[out] 安全模块提供的指定机制的最小密钥长度，以字节为单位；对于非对称算法，该参数表示最小模长，以比特为单位。

***uiMaxKeySize***

[out] 安全模块提供的指定机制的最大密钥长度，以字节为单位；对于非对称算法，该参数表示最大模长，以比特为单位。

***uiFlags***

[out] 安全模块提供的机制的类型。取值定义如下：

|  |  |  |
| --- | --- | --- |
| 位标识 | MASK | 说明 |
| SMMF\_ENCRYPT | 0x00000001 | TURE 算法可用于数据加密  FALSE 算法不能用于数据加密 |
| SMMF\_DECRYPT | 0x00000002 | TURE 算法可用于数据解密  FALSE 算法不能用于数据解密 |
| SMMF\_DIGEST | 0x00000004 | TURE 算法可用于数据摘要  FALSE 算法不能用于数据摘要 |
| SMMF\_SIGN | 0x00000008 | TURE 算法可用于签名（包括产生MAC码）  FALSE 算法不能用于签名（包括产生MAC码） |
| SMMF\_VERIFY | 0x00000010 | TURE 算法可用于验证（包括验证MAC码）  FALSE 算法不能用于验证（包括验证MAC码） |
| SMMF\_WRAP | 0x00000020 | TURE 算法可用于保护其他密钥  FALSE 算法不能用于保护其他密钥 |
| SMMF\_UNWRAP | 0x00000040 | TURE 算法可用于恢复其他密钥  FALSE 算法不能用于恢复其他密钥 |

**必要条件**

[头文件及库文件对应表](DllCall.htm)。

### 6.2.2 SM\_ALGORITHM

算法类型。

typedef struct \_SM\_ALGORITHM{

SM\_ALGORITHM\_TYPE AlgoType;

SM\_VOID\_PTR pParameter;

SM\_UINT uiParameterLen;

SM\_UINT uiReserve;

} SM\_ALGORITHM, \*PSM\_ALGORITHM;

**参数**

***AlgoType***

[in] 算法工作模式。

***pParameter***

[in] 算法参数，对应关系见下表。

***uiParameterLen***

[in] 算法参数长度，为pParameter的字节数。

***uiReserve***

[in] 算法保留参数，对应关系见下表。

以上参数与算法（Arithmetic）的关系详见下表定义：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| AlgoType | pParameter | ulParameterLen | ulReserve | 说明 |
| SMM\_ALG34\_ECB | SM\_NULL | 0 | 0 | ECB方式 |
| SMM\_ALG34\_CBC | 指向IV数据块首地址 | SMMA\_ALG34\_IV\_LEN | 0 | CBC方式 |
| SMM\_HASH1\_TZ | SM\_NULL或指向IV数据块首地址 | 0或 SMMA\_HASH1\_TZ\_LEN | 0 | HASH方式 |
| SMM\_ECC\_\* | SM\_NULL | 0 | 算法模长（256） | ECC运算 |

对于MAC和HASH算法，通常参数***pParameter***取值为SM\_NULL，参数***uiParameterLen***取值为0；对于部分指定算法要求指定IV数据，此时才在上述两个参数中指定相应的IV数据特性。

**必要条件**

[头文件及库文件对应表](DllCall.htm)。

### 6.2.3 SM\_KEY\_ATTRIBUTE

密钥属性。

typedef struct \_SM\_KEY\_ATTRIBUTE {

SM\_UINTuiObjectClass;

SM\_KEY\_TYPE KeyType;

SM\_UINTuiKeyLabel;

SM\_BYTE byStartDate[4];

SM\_BYTE byEndDate[4];

SM\_UINTuiFlags;

SM\_VOID\_PTR pParameter;

SM\_UINTuiParameterLen;

} SM\_KEY\_ATTRIBUTE, \*PSM\_KEY\_ATTRIBUTE;

**参数**

***uiObjectClass***

[in] 密钥对象类型。

根据所使用的安全模块实现算法的不同，选择与其算法相对应的密钥对象类型。

|  |  |
| --- | --- |
| KeyType | 说明 |
| SMO\_PUBLIC\_KEY | 0x00000002 |
| SMO\_PRIVATE\_KEY | 0x00000003 |
| SMO\_SECRET\_KEY | 0x00000004 |

***KeyType***

[in] 密钥类型。

根据所使用的安全模块实现算法的不同，选择与其算法相对应的密钥类型,请参考[密钥访问](KeyManage.htm)，每次仅选其一。

|  |  |
| --- | --- |
| KeyType | 说明 |
| SM\_KEY\_ALG\* | 使用ALG\*算法进行运算的对称密钥 |
| SM\_KEY\_ECC\_FP\_PUBLIC | ECC P域的公钥 |
| SM\_KEY\_ECC\_FP\_PRIVATE | ECC P域的私钥 |

***uiKeyLabel***

[in] 用户指定的密钥索引标识。

***byStartDate***

[in] 用户指定的密钥启用日期，日期格式为BCD编码，表示为“YYMD（年年月日）”。例如日期为“2010年6月30日”，则该参数为“0x20 0x10 0x06 0x30”。

***byEndDate***

[in] 用户指定的密钥结束日期，日期格式为BCD编码，表示为“YYMD（年年月日）”。

***uiFlags***

[in] 标明密钥的属性。取值定义如下：

|  |  |
| --- | --- |
| KeyType | 说明 |
| SMKA\_TOKEN | TRUE 是Token对象  FALSE 不是Token对象（缺省值） |
| SMKA\_EXTRACTABLE | TRUE 可输出  FALSE 不可输出（缺省值） |
| SMKA\_MODIFIABLE | TRUE 可修改对象属性  FALSE 不可修改对象属性 |
| SMKA\_ENCRYPT | TURE 密钥可用于数据加密  FALSE 密钥不能用于数据加密 |
| SMKA\_DECRYPT | TURE 密钥可用于数据解密  FALSE 密钥不能用于数据解密 |
| SMKA\_SIGN | TURE 密钥可用于签名（包括产生MAC码）  FALSE 密钥不能用于签名（包括产生MAC码） |
| SMKA\_VERIFY | TURE 密钥可用于验证（包括验证MAC码）  FALSE 密钥不能用于验证（包括验证MAC码） |
| SMKA\_WRAP | TURE 密钥可用于保护其他密钥  FALSE 密钥不能用于保护其他密钥 |
| SMKA\_UNWRAP | TURE 密钥可用于恢复其他密钥  FALSE 密钥不能用于恢复其他密钥 |

在实际使用中，参数***uiFlags***可从以上定义中任选需要的定义，用位或的形式组合在一起。

请注意：Token对象即永久保存对象，掉电也不会丢失，如果未主动调用销毁函数销毁其内容，它将永久占用模块资源，直到调用函数[SM\_ClearAllTokenKeys、SM\_DestroySensitiveInfo](SM_ClearHWKeys.htm)销毁所有密钥。

该版本不支持token密钥的使用。

***pParameter***

[in] 密钥参数，若该参数是ECC密钥属性访问指针，则应该指向[SM\_ECC\_PARAMETER](SM_RSA_PARAMETER.htm)数据结构；否则应该置为SM\_NULL。

***uiParameterLen***

[in] 密钥参数长度，如果采用ECC算法，则uiParameterLen应该设置为结构SM\_ECC\_PARAMETER的有效长度；否则置为0。

**必要条件**

[头文件及库文件对应表](DllCall.htm)。

### 6.2.4 SM\_ECC\_PARAMETER

ECC密钥访问属性。

typedef struct \_SM\_ECC\_PARAMETER

{

SM\_UINT uiModulusBits;

SM\_VOID\_PTR pParameter;

SM\_UINT uiParameterLen;

} SM\_ECC\_PARAMETER, \*PSM\_ECC\_PARAMETER;

**参数**

***uiModulusBits***

[in] ECC模长，以比特为单位。

您可以使用函数[SM\_GetMechanismInfo](SM_GetMechanismInfo.htm)来获取您正在使用的模块所支持的ECC模长，然后对该参数进行设置。当参数***uiMechanismList***取值为SM\_ECC\_\*时，SM\_MECHANISM\_INFO结构中的参数***ulMaxKeySize***表示模块支持的ECC密钥的最大模长，***uiMinKeySize***表示模块支持的ECC密钥的最小模长。

***pParameter***

[in] 保留参数。

***uiParameterLen***

[in] 保留参数。

**必要条件**

[头文件及库文件对应表](DllCall.htm)。

### 6.2.5SM\_BLOB\_KEY

密钥BLOB信息。

typedef struct \_SM\_BLOB\_DATA

{

SM\_UINT uiDataLen;

PSM\_BYTE pbyData;

}SM\_BLOB\_KEY, \*PSM\_BLOB\_KEY;

**参数**

***uiDataLen***

[in] 密钥数据的有效长度，以字节为单位。

***pbyData***

[in] 密钥数据的首地址，该参数必须由用户自行分配缓冲区大小。

该结构体参数取值定义如下：

|  |  |  |
| --- | --- | --- |
| ulDataLen | pbyData | 说明 |
| 密钥句柄（SM\_KEY\_HANDLE）的长度（32位下为4,  64位下为8） | 密钥访问句柄 | 表明此时使用密钥访问句柄 |
| >16 | 密钥有效数据 | 表明此时使用密钥实际有效数据 |

**使用**

该结构体在密钥管理接口函数中作为输入参数使用，具体请参见函数SM\_GenerateKey、SM\_ImportKey、SM\_GenerateKeyPair、SM\_ImportPublicKey和SM\_ImportPrivateKey说明。

**备注**

**必要条件**

[头文件及库文件对应表](DllCall.htm)。

### 6.2.6SM\_BLOB\_CONTEXT

上下文BLOB信息。

typedef struct \_SM\_BLOB\_DATA

{

SM\_UINT uiDataLen;

PSM\_BYTE pbyData;

}SM\_BLOB\_CONTEXT, \*PSM\_BLOB\_CONTEXT;

**参数**

***uiDataLen***

[in] 用户分配数据的有效长度，以字节为单位。缺省必须大于等于4字节。

***pbyData***

[in] 用户分配数据的首地址，该参数必须由用户自行分配缓冲区大小。

该结构体参数取值定义如下：

|  |  |  |
| --- | --- | --- |
| uiDataLen | pbyData | 说明 |
| 4 | 算法工作模式 | 表明此时由硬件内部自行维护上下文数据，用户只需指定使用的算法类型，请参见结构体SM\_ALGORITHM的参数AlgoType的说明 |
| >4 | 算法工作模式和上下文数据 | 表明此时由用户指定算法运算的上下文数据和使用的算法类型，根据不同算法的定义要求，其对应的上下文数据长度有所不同，具体取值请参见头文件"sm\_algo.h" |

**使用**

该结构体在算法运算接口函数中使用。对于函数SM\_xxxInit作为输出参数使用，对于函数SM\_xxxUpdate、SM\_xxxFinal和SM\_xxx均作为输入输出参数使用。

**必要条件**

[头文件及库文件对应表](DllCall.htm)。

### 6.2.7 SM\_BLOB\_ECCCIPHER

ECC密文BLOB信息

typedef struct \_SM\_BLOB\_ECCCIPHERA

{

SM\_UINT uiSessionKeyLen;

SM\_UINT uiCipherDataLen;

SM\_UINT uiCheckDataLen;

PSM\_BYTE pbyData;

}SM\_BLOB\_ECCCIPHER, \*PSM\_BLOB\_ECCCIPHER;

**参数**

***uiSessionKeyLen***

[in/out] 会话密钥的有效长度，以字节为单位。

***uiCipherDataLen***

[in/out] 密文数据的有效长度，以字节为单位，作为ECC解密函数的输入参数时，此处密文的有效长度的范围应在1至1024字节之间。

***uiCheckDatalen***

[in/out]验证数据的有效长度，以字节为单位。

***pbyData***

[in/out] ECC密文数据流，数据流中数据依次是：会话密钥、密文数据和验证数据。

**使用**

该结构体在使用ECC进行加密时作为输出参数使用，使用ECC解密时作为输入参数使用。具体请参见函数SM\_ECCEncrypt、SM\_ECCDecryp说明。

### 6.2.8 SM\_DEVICE\_INFO

**设备信息**

/\*! struct SM\_DEVICE\_INFO. \*/

typedef struct \_SM\_DEVICE\_INFO

{

SM\_RESOURCE\_INFO stDevResourceInfo;

SM\_MANUFCT\_INFO stManufactureInfo;

SM\_UINT uiFlags;

SM\_UINT uiStatus;

} SM\_DEVICE\_INFO, \*PSM\_DEVICE\_INFO;

**参数**

**stDevResourceInfo**

设备资源信息，详细见结构体SM\_RESOURCE\_INFO。

**stManufactureInfo**

设备生产信息，详细见结构体SM\_MANUFCT\_INFO。

**设备生产信息**

typedef struct \_SM\_MANUFACTRUE\_INFO

{

SM\_BYTE byModel[16];

SM\_BYTE byManufacturerID[32];

SM\_BYTE byManufactureDate[4];

SM\_BYTE byBatch[4];

SM\_BYTE bySerial[16];

SM\_BYTE byDateTime[8];

} SM\_MANUFCT\_INFO;

**设备资源信息**

/\*! struct SM\_NVMEM\_INFO. \*/

typedef struct \_SM\_NVMEM\_INFO

{

SM\_UINT uiMaxNVMemSize;

SM\_UINT uiNVMemSectorSize;

} SM\_NVMEM\_INFO;

/\*! struct SM\_ADMEM\_INFO. \*/

typedef struct \_SM\_ADMEM\_INFO

{

SM\_UINT uiMaxAuthDevMem1Size;

SM\_UINT uiMaxAuthDevMem2Size;

} SM\_ADMEM\_INFO;

/\*! struct SM\_RESOURCE\_INFO. \*/

typedef struct \_SM\_RESOURCE\_INFO

{

SM\_UINT uiHPIBufSize;

SM\_WORD wMaxPipeCount;

SM\_WORD wFreePipeCount;

SM\_WORD wMaxSecretKeyCount;

SM\_WORD wFreeSecretKeyCount;

SM\_WORD wMaxPublicKeyCount;

SM\_WORD wFreePublicKeyCount;

SM\_WORD wMaxPrivateKeyCount;

SM\_WORD wFreePrivateKeyCount;

SM\_WORD wMaxSecretKeyTokenCount;

SM\_WORD wFreeSecretKeyTokenCount;

SM\_WORD wMaxPublicKeyTokenCount;

SM\_WORD wFreePublicKeyTokenCount;

SM\_WORD wMaxPrivateKeyTokenCount;

SM\_WORD wFreePrivateKeyTokenCount;

SM\_NVMEM\_INFO stNVMem;

SM\_ADMEM\_INFO stAuthDevMem;

SM\_WORD wMaxPinLen;

SM\_WORD wMinPinLen;

SM\_WORD wMaxSOPinLen;

SM\_WORD wMinSOPinLen;

SM\_WORD wHardwareVersion;

SM\_WORD wFirmwareVersion;

} SM\_RESOURCE\_INFO;

**参数**

|  |  |
| --- | --- |
| *Parameter* | 含义 |
| uiHPIBufSize | 传输buffer尺寸 |
| wMaxPipeCount | 设备可用最多管道数 |
| wFreePipeCount | 设备剩余管道数 |
| wMaxSecreKeyCount | 设备可用最多对称密钥个数 |
| wFreeSecreKeyCount | 设备剩余对称密钥个数 |
| wMaxPublicKeyCount | 设备可用最多公开密钥个数 |
| wFreePublicKeyCount | 设备剩余公开密钥个数 |
| wMaxPrivateKeyCount | 设备可用最多xxx密钥个数 |
| wFreePrivateKeyCount | 设备剩余xxx密钥个数 |
| wMaxSecretKeyTokenCount | 设备可用最多对称密钥Token对象个数(该项目可忽略) |
| wFreeSecretKeyTokenCount | 设备剩余对称密钥Token对象个数(该项目可忽略) |
| wMaxPublicKeyTokenCount | 设备可用最多公开密钥Token对象个数(该项目可忽略) |
| wFreePublicKeyTokenCount | 设备剩余公开密钥Token对象个数(该项目可忽略) |
| wMaxPrivateKeyTokenCount | 设备可用最多xxx密钥Token对象个数(该项目可忽略) |
| wFreePrivateKeyTokenCount | 设备剩余xxx密钥Token对象个数(该项目可忽略) |
| stNVMem | 非易失存储区结构体 |
| stAuthDevMem | 认证设备存储区结构体 |
| wMaxPinLen | 用户口令的最大长度 |
| wMinPinLen | 用户口令的最小长度 |
| wMaxSOPinLen | SO口令的最大长度(该项目可忽略) |
| wMinSOPinLen | SO口令的最小长度(该项目可忽略) |
| wHardwareVersion | 设备硬件版本信息，前两字节为主版本号，后两字节为次版本号 |
| wFirewareVersion | 设备软件（DSP）版本信息，前两字节为主版本号，后两字节为次版本号 |

**必要条件**

[头文件及库文件对应表](DllCall.htm)。

## 6.3接口函数

安全模块API5.2提供的接口函数根据系统架构的划分为以下几部分：[设备访问接口函数](Device_Fun.htm)，[安全管道访问接口函数](SecPipe_Fun.htm)，[密钥访问接口函数](KeyManage_Fun.htm)。

### 6.3.1设备访问接口函数

安全模块API5.2提供的设备访问接口函数如下：

1．设备基本操作函数（11个）

|  |  |  |
| --- | --- | --- |
| 访问接口函数 | 成对使用 | 说明 |
| SM\_GetDeviceNum | \ | 获取有效设备数量 |
| SM\_GetErrorString | \ | 提取错误信息 |
| SM\_GetAPIVersion | \ | 获取API接口库版本信息 |
| SM\_GetDeviceType | \ | 获取安全模块设备类型 |
| SM\_OpenDevice | 要求 | 打开硬件设备 |
| SM\_CloseDevice | 关闭硬件设备 |
| SM\_LockMem | 要求 | 加锁安全存储区 |
| SM\_UnLockMem | 解锁安全存储区 |
| SM\_WriteNonVolatile | \ | 写非易失存储区 |
| SM\_ReadNonVolatile | \ | 读非易失存储区 |
| SM\_BiuldAuthDev | \ | 制作用户的认证介质 |

2．设备访问函数（5个）

|  |  |
| --- | --- |
| 访问接口函数 | 说明 |
| SM\_GetMechanismList | 获取机制类型列表 |
| SM\_GetMechanismInfo | 获取机制信息 |
| SM\_GetDeviceIndex | 提取硬件设备索引号 |
| SM\_TestDevice | 设备自检 |
| SM\_GetDeviceInfo | 读取设备信息 |

#### 6.3.1.1 SM\_GetDeviceNum

该函数用来获取可访问的安全模块个数。

SM\_RV SM\_GetDeviceNum(

PSM\_UINT puiDevNum

);

参数

***puiDevNum***

[out] 返回可访问的安全模块个数。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm)获取错误描述。

**备注**

该函数用来获取可访问的安全模块个数，可在调用SM\_OpenDevice函数前调用。

**示例代码**

**SM\_UINT ulDevNum = 0;**

**SM\_RV uiRet = 0;**

**uiRet = SM\_GetDeviceNum(&ulDevNum);**

**if(uiRet == SM\_ERR\_FREE )**

**{**

**printf("SM\_GetDeviceNum is success,ulDevNum=%d\n",ulDevNum);**

**}**

**else**

**{**

**printf("SM\_GetDeviceNum is error,errcode=0x%x\n",uiRet);**

**exit(1);**

**}**

#### 6.3.1.2 SM\_GetErrorString

该函数将输入的错误码转换为字符串。

**PSM\_CHAR SM\_GetErrorString(**

**SM\_RV uiErrCode,**

**SM\_BOOL bChinese**

**);**

**参数**

***uiErrCode***

[in] 需要进行转化的错误码。

***bChinese***

指定字串的类型，取值定义如下：

|  |  |
| --- | --- |
| bChinese | 说明 |
| TRUE | 以中文形式返回错误字符串 |
| FALSE | 以英文形式返回错误字符串 |

**返回值**

与错误码相对应的字符串。

**示例代码**

**SM\_RV uiErrCode = 0x5;**

**SM\_RV uiRet = 0;**

**uiRet = SM\_GetErrorString(ulErrCode, TRUE);**

#### 6.3.1.3SM\_GetAPIVersion

该函数将获取API接口库的软件版本信息。

**PSM\_CHAR SM\_GetAPIVersion(**

**);**

**返回值**

与API接口库头文件中定义相符的API接口库版本信息字符串，该信息最大长度为20字节。

**示例代码**

**SM\_RV uiRet = 0;**

**uiRet = SM\_GetAPIVersion();**

**printf("APIversion: %s\n\n",uiRet);**

#### 6.3.1.4 SM\_GetDeviceType

该函数将获取设备类型值。

**PSM\_CHAR SM\_GetDeviceType(**

**PSM\_UINT puiDeviceType**

**);**

**参数**

***puiDeviceType***

[out]用于存放安全模块设备类型值,该参数返回值仅当函数返回值为SM\_ERR\_FREE时有效。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm)获取错误描述。

**示例代码**

**SM\_RV uiRet = 0;**

**SM\_UINT uiDeviceType = SM\_NULL;**

**uiRet = SM\_GetDeviceType(&uiDeviceType);**

**if(uiRet == SM\_ERR\_FREE )**

**{**

**printf("SM\_GetDeviceType is success,**

**puiDeviceType =%x\n", uiDeviceType);**

**}**

**else**

**{**

**printf("SM\_GetDeviceType is error,errcode=0x%x\n",uiRet);**

**exit(1);**

**}**

#### 6.3.1.5 SM\_OpenDevice

该函数用来打开安全模块并获取设备访问句柄。

**SM\_RV SM\_OpenDevice(**

**SM\_UINT uiDevID,**

**SM\_BOOL bExclusive,**

**PSM\_DEVICE\_HANDLE phDevice**

**);**

**参数**

***uiDevID***

[in]指定安全模块设备访问索引号，参考函数<SM_GetDeviceNum>，例如第一块为0，第二块为1。

***bExclusive***

[in]指定安全模块访问方式，取值定义如下：

|  |  |
| --- | --- |
| bExclusive | 说明 |
| TRUE | 以独占方式访问安全模块 |
| FALSE | 以共享方式访问安全模块 |

***phDevice***

[out]用于存放安全模块设备访问句柄,该参数返回的句柄仅当函数返回值为SM\_ERR\_FREE时有效。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm)获取错误描述。

**备注**

1.安全模块硬件设备有两种访问方式：独占方式和共享方式。当设备以独占方式打开访问时，该设备只能被唯一地打开，此时仅支持当前进程访问，其它进程无法调用该函数再次打开此设备，否则函数将返回错误值SM\_ERR\_EXCLUSIVE或SM\_ERR\_OCCUPY，直至该进程调用函数[SM\_CloseDevice](SM_CloseDevice.htm)结束访问。当设备以共享方式打开访问时，则可支持多个进程或多个线程同时访问该安全模块硬件设备。

2.该函数需要与函数SM\_CloseDevice一一对应使用，在每个进程(或者线程)调用此函数打开设备后，必须调用函数SM\_CloseDevice执行关闭设备，结束访问操作。

3.参数phDevice不应被用来作为监测函数是否成功的标志，函数失败时内部可能不会将参数phDevice置空(SM\_NULL)，因此，函数调用时应该以函数返回值作为判断依据。

**示例代码**

**SM\_DEVICE\_HANDLEhDevice = SM\_NULL;**

**SM\_RV uiRet = 0;**

**uiRet = SM\_OpenDevice(0,0, &hDevice);**

**if(uiRet == SM\_ERR\_FREE )**

**{**

**printf("SM\_OpenDevice is success\n");**

**SM\_CloseDevice(hDevice);**

**}**

**else**

**{**

**printf("SM\_OpenDevice =0x%x\n",uiRet);**

**exit(1);**

**}**

**相关函数及说明**

关闭安全模块：[SM\_CloseDevice](SM_CloseDevice.htm)

#### 6.3.1.6 SM\_CloseDevice

函数用来关闭安全模块。

**SM\_RV SM\_CloseDevice**

**(**

**SM\_DEVICE\_HANDLE hDevice**

**);**

**参数**

***hDevice***

[in]安全模块设备访问句柄。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm)获取错误描述。

**备注**

1.该函数需要与[SM\_OpenDevice](SM_OpenDevice.htm)一一对应使用。

2.安全模块硬件设备有两种访问方式：独占方式和共享方式。当设备以独占方式打开访问时，其它进程或线程无法调用函数SM\_OpenDevice再次打开此设备，直至该进程或线程调用此函数结束硬件设备访问。当设备以共享方式打开访问时，此函数需要与函数SM\_OpenDevice一一对应使用，每次调用设备对象的引用计数会被减1，当计数为0时，设备才会被释放。

**示例代码**

**SM\_DEVICE\_HANDLEhDevice = SM\_NULL;**

**SM\_RV uiRet = 0;**

**uiRet = SM\_OpenDevice(0,0, &hDevice);**

**if(uiRet == SM\_ERR\_FREE )**

**{**

**printf("SM\_OpenDevice is success\n");**

**SM\_CloseDevice(hDevice);**

**}**

**else**

**{**

**printf("SM\_OpenDevice =0x%x\n",uiRet);**

**exit(1);**

**}**

**uiRet = SM\_CloseDevice(hDevice);**

**if(uiRet == SM\_ERR\_FREE )**

**{**

**printf("SM\_ CloseDevice is success\n");**

**}**

**else**

**{**

**printf("SM\_CloseDevice =0x%x\n",uiRet);**

**exit(1);**

**}**

**相关函数及说明**

打开安全模块：[SM\_OpenDevice](SM_OpenDevice.htm)

#### 6.3.1.7 SM\_GetMechanismList

该函数获取安全模块内部支持的机制类型列表。

**SM\_RV SM\_GetMechanismList**

**(**

**SM\_DEVICE\_HANDLE hDevice,**

**PSM\_UINT puiMechanismList,**

**PSM\_WORD pwMechanismNum**

**);**

**参数**

***hDevice***

[in] 安全模块设备访问句柄。

***puiMechanismList***

[out] 支持的机制信息列表，具体取值请参见下列表

|  |  |
| --- | --- |
| pulMechanismList | 说明 |
| SMM\_SYM1\_TZ\_ECB | 对称密钥进行ECB模式运算 |
| SMM\_SYM1\_TZ\_CBC | 对称密钥进行CBC模式运算 |
| SMM\_HASH1\_TZ | 摘要运算 |
| SMM\_ECC\_\*\_ENC | ECC非对称加密运算 |
| SMM\_ECC\_\*\_DEC | ECC非对称解密运算 |
| SMM\_ECC\_\*\_SIGN | ECC非对称签名运算 |
| SMM\_ECC\_\*\_VERIFY | ECC非对称验签运算 |

***pwMechanismNum***

[in\out] 同时支持的机制数量。

由于函数SM\_GetMechanismList不划分自己的空间，因此应用会经常两次调用SM\_GetMechanismList 。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm) 获取错误描述。

**备注**

应用调用该函数有两种方法：

1.当参数*puiMechanismList*取值为SM\_NULL时，那么该函数将通过参数p*wMechanismNum*获得有效的机制数目。

2.当参数*puiMechanismList*取值不为SM\_NULL时，则参数*pwMechanismNum*必须包含由参数*puiMechanismList*所指缓冲容量的大小(以SM\_UINT为单位)。如果该缓冲容量有足够大的空间容纳机制列表，那么在缓冲中返回该列表，并返回SM\_ERR\_FREE；否则返回错误。在任何一种情况下，参数pwMechanismNum的取值被设置成可容纳机制数的大小。

建议在程序初始化时就调用该函数以了解目前模块对于算法及函数功能的支持情况。

**相关函数及说明**

获取机制信息：[SM\_GetMechanismInfo](SM_GetMechanismInfo.htm)

**示例代码**

**SM\_DEVICE\_HANDLEhDevice = SM\_NULL;**

**SM\_WORD wMechanismNum = 1;**

**SM\_UINT \*puiMechanismList = SM\_NULL;**

**SM\_MECHANISM\_INFO stMech;**

**SM\_RV uiRet = 0;**

**puiMechanismList=(SM\_UINT\*)malloc(sizeof(SM\_UINT)\* wMechanismNum)**

**memset(puiMechanismList, 0, wMechanismNum);**

**memset(stMech,0,sizeof(SM\_MECHANISM\_INFO));**

**uiRet = SM\_OpenDevice(0,0, &hDevice);**

**if(uiRet == SM\_ERR\_FREE )**

**{**

**printf("SM\_OpenDevice is success\n");**

**}**

**else**

**{**

**printf("SM\_OpenDevice =0x%x\n",uiRet);**

**exit(1);**

**}**

**uiRet = SM\_GetMechanismList(hDevice,**

**puiMechanismList, &wMechanismNum);**

**if ( ret != SM\_ERR\_FREE )**

**{**

**printf("[Error]SM\_GetMechanismList(num) = 0x%x\n", ret);**

**exit(1);**

**}**

**else**

**{**

**printf("SM\_GetMechanismList is success");**

**printf("SM\_GetMechanismList(num) = 0x%x,**

**( puiMechanismList)=0x%x\n",**

**wMechanismNum,\* puiMechanismList);**

**}**

**uiRet=SM\_GetMechanismInfo(hDevice, puiMechanismList, &stMech);**

**if ( ret != SM\_ERR\_FREE )**

**{**

**printf("[Error]SM\_GetMechanismInfo[%d] = 0x%x\n", i, ret);**

**exit(1);**

**}**

**else**

**{**

**printf("SM\_GetMechanismInfo is ok!\n");**

**printf("ulMinBlockSize = %d\n", stMech.uiMinBlockSize);**

**printf("ulMaxBlockSize = %d\n", stMech.uiMaxBlockSize);**

**printf("ulMinKeySize = %d\n", stMech.uiMinKeySize);**

**printf("ulMaxKeySize = %d\n", stMech.uiMaxKeySize);**

**printf("ulFlags = 0x%x\n\n", stMech.uiFlags);**

**}**

#### 6.3.1.8 SM\_GetMechanismInfo

该函数获取安全模块内部支持的一个特定机制的信息。

**SM\_RV SM\_GetMechanismInfo**

**(**

**SM\_DEVICE\_HANDLE hDevice,**

**SM\_UINT uiMechanism,**

**PSM\_MECHANISM\_INFO pstMech**

**);**

**参数**

***hDevice***

[in] 安全模块设备访问句柄。

***uiMechanism***

[in] 指定的机制类型，支持的机制信息列表，具体取值请参见下支持的机制信息列表

|  |  |
| --- | --- |
| uiMechanism | 说明 |
| SMM\_SYM1\_TZ\_ECB | 对称密钥进行ECB模式运算 |
| SMM\_SYM1\_TZ\_CBC | 对称密钥进行CBC模式运算 |
| SMM\_HASH1\_TZ | 摘要运算 |
| SMM\_ECC\_\*\_ENC | ECC非对称加密运算 |
| SMM\_ECC\_\*\_DEC | ECC非对称解密运算 |
| SMM\_ECC\_\*\_SIGN | ECC非对称签名运算 |
| SMM\_ECC\_\*\_VERIFY | ECC非对称验签运算 |

***pstMech***

[out] 指向接收机制信息[PSM\_MECHANISM\_INFO](SM_MECHANISM_INFO.htm)结构的单元。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm)获取错误描述。

**备注**

建议在程序初始化时就调用该函数以了解目前模块对于算法及函数功能的支持情况,参考函数SM\_GetMechanismList。

**相关函数及说明**

获取机制类型列表：[SM\_GetMechanismList](SM_GetMechanismList.htm)

**示例代码**

见SM\_GetMechanismList示例代码。

#### 6.3.1.9 SM\_GetDeviceIndex

该函数用来获取安全模块设备访问索引号。

**SM\_RV WINAPI SM\_GetDeviceIndex**

**(**

**SM\_DEVICE\_HANDLE hDevice,**

**PSM\_UINTpuiDeviceIndex**

**);**

**参数**

***hDevice***

[in] 安全模块设备访问句柄。

***puiDeviceIndex***

[out] 用于存放安全模块设备访问索引号,该参数返回的索引号仅当函数返回值为SM\_ERR\_FREE时有效。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm)获取错误描述。

**示例代码**

//假定目前安全模块已经成功打开，hDevice已经获取

**SM\_DEVICE\_HANDLEhDevice = SM\_NULL;**

**SM\_UINTuiDeviceIndex;**

**SM\_RV uiRet = 0;**

**uiRet = SM\_GetDeviceIndex (hDevice, &uiDeviceIndex);**

**if(uiRet == SM\_ERR\_FREE )**

**{**

**printf("SM\_GetDeviceIndex is success\n");**

**}**

**else**

**{**

**printf("SM\_GetDeviceIndex =0x%x\n",uiRet);**

**exit(1);**

**}**

**相关函数及说明**

打开安全模块：[SM\_OpenDevice](SM_OpenDevice.htm)

关闭安全模块：[SM\_CloseDevice](SM_CloseDevice.htm)

#### 6.3.1.10 SM\_TestDevice

函数用来测试模块平台当前状态。

**SM\_RV WINAPI SM\_TestDevice**

**(**

**SM\_DEVICE\_HANDLE hDevice,**

**PSM\_UINT puiResult**

**)**

**参数**

***hDevice***

[in] 模块平台设备访问句柄。

***puiResult***

[out] 指向自检结果的指针。该自检结果仅在函数返回值为SM\_ERR\_FREE时有意义。如自检结果为0x00000000，自检通过；否则，自检失败。自检结果的每一比特对应一个自检项目。比特值为'0'，该项目通过；比特值为'1'，该项目失败。

|  |  |
| --- | --- |
| *pulResult* | *表征位含义* |
| 0x00000000 | 状态正常 |
| 0x00000001 | 物理噪声源检测错 |
| 0x00000002 | SDRAM检测错 |
| 0x00000004 | SSX30E检测错 |
| 0x00000008 | FPGA检测错 |

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则，调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm) 获取错误描述。

|  |  |  |
| --- | --- | --- |
| *错误码* | *错误号* | *含义* |
| SM\_ERR\_FREE | 0x0000 | 模块平台操作成功 |
| SM\_ERR\_MALLOC | 0x0017 | 内存分配失败 |
| SM\_ERR\_UNKNOWN | 0x0016 | 模块平台发生未知错误 |
| SM\_ERR\_TIME\_OUT | 0x0106 | 模块平台操作超时 |
| SM\_ERR\_DEVICE\_HANDLE | 0x0101 | 无效的模块平台设备访问句柄 |

**备注**

１、建议在调用了SM\_OpenDevice函数后立即调用该函数，以保证模块平台处于正常的工作状态。

**示例代码**

**SM\_DEVICE\_HANDLE hDevice;**

**SM\_RV uiRet = SM\_OpenDevice(0, 0, 0, &hDevice);**

**if(SM\_ERR\_FREE == uiRet)**

**{**

**SM\_UINT　ulResult;**

**SM\_WORD wTryNum,**

**uiRet = SM\_TestDevice(hDevice,&ulResult);**

**if(uiRet == SM\_ERR\_FREE )**

**{**

**printf("SM\_TestDevice is success\n");**

**}**

**else**

**{**

**printf("SM\_TestDevice is error\n");**

**SM\_CloseDevice(hDevice);**

**exit(1);**

**}**

**}**

**必要条件**

[头文件及库文件对应表](../接口文档/DllCall.htm)。

**相关函数及说明**

关闭模块平台：[SM\_CloseDevice](SM_CloseDevice.htm)

打开模块平台：[SM\_OpenDevice](SM_OpenDevice.htm)

#### 6.3.1.11SM\_GetDeviceInfo

函数用来获取模块平台设备信息。

**SM\_RV WINAPI SM\_GetDeviceInfo**

**(**

**SM\_DEVICE\_HANDLE hDevice,**

**PSM\_DEVICE\_INFO pDeviceInfo**

**)**

**参数**

***hDevice***

[in] 模块平台设备访问句柄。

***pDeviceInfo***

[out] 指向[SM\_DEVICE\_INFO](SM_DEVICE_INFO.htm)结构的指针,用于存放模块平台信息。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则，调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm) 获取错误描述。

|  |  |  |
| --- | --- | --- |
| *错误码* | *错误号* | *含义* |
| SM\_ERR\_FREE | 0x0000 | 模块平台操作成功 |
| SM\_ERR\_UNKNOWN | 0x0016 | 模块平台发生未知错误 |
| SM\_ERR\_TIME\_OUT | 0x0106 | 模块平台操作超时 |
| SM\_ERR\_DEVICE\_HANDLE | 0x0101 | 无效的模块平台设备访问句柄 |

**备注**

该函数可用于获取模块平台的安全管道、密钥分组长度、安全存储区、非易失存储区的长度、版本、插槽等方面的信息。

**示例代码**

**SM\_DEVICE\_HANDLE hDevice;**

**SM\_DEVICE\_INFO DeviceInfo;**

**SM\_RV uiRet = SM\_GetDeviceInfo(hDevice, &DeviceInfo);**

**if(SM\_ERR\_FREE ==uiRet)**

**{**

**printf("SM\_GetDeviceInfois success\n");**

**}**

**else**

**{**

**printf("SM\_GetDeviceInfo error is Code=0x%x\n",uiRet);**

**exit(1);**

**}**

#### 6.3.1.12 SM\_LockMem

SM\_LockMem函数用来加锁安全存储区(指IC卡或模块平台内的IC卡芯片)与非易失性存储区(指模块平台内部FLASH或电池保护的SRAM)。

SM\_RV SM\_LockMem

(

SM\_DEVICE\_HANDLE hDevice,

)

**参数**

***hDevice***

[in] 模块平台设备访问句柄。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则，调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm) 获取错误描述。

**备注**

该函数需与函数[SM\_UnlockMem](SM_UnlockMem.htm)配对使用。

**示例代码**

//假定目前模块平台已经成功打开,hDevice已经获取。

SM\_DEVICE\_HANDLE hDevice;

SM\_RV ulRet = SM\_LockMem(hDevice);

if (SM\_ERR\_FREE == ulRet)

{

　SM\_UCHAR DataIn[100];

　SM\_ULONG ulDataInLen=32;

　SM\_UCHAR DataOut[100];

　ulRet = SM\_WriteNonVolatile(hDevice ,0, DataIn, ulDataInLen);

　if(ulRet != SM\_ERR\_FREE)

　{

printf("SM\_WriteNonVolatile is error errorCode=0x%x\n",ulRet);

}

else

　{

　　ulRet = SM\_ReadNonVolatile(hDevice, 0, ulDataInLen, DataOut);

　　if(ulRet != SM\_ERR\_FREE)

　　{

　　　printf("SM\_ReadNonVolatile is error errorCode=0x%x\n",ulRet);

　　}

　　else

　　{

　　　if(memcmp(DataIn, DataOut, ulDataInLen))

　　　{

　　　　printf("memcmp is error\n");

　　　}

　　}

　}

　ulRet = SM\_UnlockMem(hDevice, SM\_MEM\_NONVOLATILE);

　if(ulRet != SM\_ERR\_FREE)

　{

　　printf("SM\_UnlockMem is error errorCode=0x%x\n",ulRet);

　}

　printf("SM\_LockMem is success\n");

}

else

{

　printf("SM\_LockMem is error errorCode=0x%x\n",ulRet);

　exit(1);

}

**必要条件**

[头文件及库文件对应表](../接口文档/DllCall.htm)。

**相关函数及说明**

解锁存储区：[SM\_UnlockMem](SM_UnlockMem.htm)

#### 6.3.1.13 SM\_UnlockMem

 SM\_UnlockMem函数用来解锁安全存储区(指IC卡或模块平台内的IC卡芯片)与非易失性存储区(指模块平台内部FLASH或电池保护的SRAM)。

SM\_RV SM\_UnlockMem

(

SM\_DEVICE\_HANDLE hDevice,

)

**参数**

***hDevice***

[in] 模块平台设备访问句柄。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则，调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm) 获取错误描述。

**备注**

该函数需与函数SM\_LockMem配对使用。

**示例代码**

 请参考函数[SM\_LockMem](SM_LockMem.htm)的示例代码。

**必要条件**

[头文件及库文件对应表](../接口文档/DllCall.htm)。

**相关函数及说明**

加锁存储区：[SM\_LockMem](SM_LockMem.htm)

#### 6.3.1.14 SM\_WriteNonVolatile

函数向模块平台内部非易失性存储区(指模块平台内部FLASH或电池保护的SRAM)中写入数据。

SM\_RV WINAPI SM\_WriteNonVolatile

(

SM\_DEVICE\_HANDLE hDevice,

SM\_UINT uiLocation,

SM\_UINT uiDataInLen,

PSM\_BYTE pbyDataIn

)

**参数**

***hDevice***

[in] 模块平台设备访问句柄。

***uiLocation***

[in] 数据写入的绝对地址，从0开始，针对64位PCI平台，该地址必须为4的倍数，针对其他模块，该地址必须为2的倍数。

***uiDataInLen***

[in] 写入数据的字节数，针对64位PCI平台，该长度必须为4的倍数，针对其他模块，该长度必须为2的倍数。

***pbyDataIn***

[in] 待写入数据块的首地址。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则，调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm) 获取错误描述。

**备注**

该函数向模块平台内部FLASH或电池保护的SRAM中写入数据，下电后数据不会丢失。对模块平台内部物理结构不了解的人员不应调用该函数，以免破坏模块平台数据结构。参考[SM\_ReadNonVolatile](SM_ReadNonVolatile.htm)函数的说明。非易失性存储区大小可通过调用函数[SM\_GetDeviceInfo](SM_GetDeviceInfo.htm)获取。

**示例代码**

请参考函数[SM\_LockMem](SM_LockMem.htm)的示例代码。

**必要条件**

[头文件及库文件对应表](../接口文档/DllCall.htm)。

**相关函数及说明**

读非易失性存储区：[SM\_ReadNonVolatile](SM_ReadNonVolatile.htm)

#### 6.3.1.15 SM\_ReadNonVolatile

 函数从模块平台内部非易失性存储区(指模块平台内部FLASH或电池保护的SRAM)中读出数据。

SM\_RV WINAPI SM\_ReadNonVolatile

(

SM\_DEVICE\_HANDLE hDevice,

SM\_UINT uiLocation,

SM\_UINT uiDataOutLen,

PSM\_BYTE pbyDataOut

)

**参数**

***hDevice***

[in] 模块平台设备访问句柄。

***ulLocation***

[in] 数据读取的绝对地址,从0开始，针对64位PCI平台，该地址必须为4的倍数，针对其他模块，该地址必须为2的倍数。

***uiDataOutLen***

[in] 读出数据的字节数,针对64位PCI平台，该长度必须为4的倍数，针对其他模块，该长度必须为2的倍数。

***pbyDataOut***

[out] 读出数据块的首地址。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则，调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm) 获取错误描述。

**备注**

该函数从模块平台内部FLASH或电池保护的SRAM中读出数据。对模块平台内部物理结构不了解的人员不应调用该函数。参考SM\_WriteNonVolatile函数的说明。非易失性存储区大小可通过调用函数[SM\_GetDeviceInfo](SM_GetDeviceInfo.htm)获取。

**示例代码**

请参考函数[SM\_LockMem](SM_LockMem.htm)的示例代码。

**必要条件**

[头文件及库文件对应表](../接口文档/DllCall.htm)。

**相关函数及说明**

写非易失性存储区：[SM\_WriteNonVolatile](SM_WriteNonVolatile.htm)

#### 6.3.1.16 SM\_BuildAuthDev

函数用来制作用户的认证介质。

SM\_RV WINAPI SM\_BuildAuthDev

(

SM\_DEVICE\_HANDLE hDevice,

**PSM\_UCHAR pPin,**

**SM\_UINT uiPinLen,**

**SM\_WORD wKeyNum,**

**PSM\_BYTE pbyVerifyPublicKey,**

**PSM\_WORD pwVerifyPublicKeyLEN,**

**PSM\_BYTE pbyWrapPublicKey,**

**PSM\_WORD pwWrapPublicKeyLEN,**

)

***hDevice***

[in] 模块平台设备访问句柄。

***pPin***

[in] 模块平台初始用户认证介质口令。为'\0'结尾字符串。

***uiPinLen***

[in] 模块平台初始用户认证介质口令长度，该项目固定PinLen = 8。

***wKeyNum***

[in] 产生的公私钥对数量。本项目Number=2（或1）。当Number=2时，模块产生2对公私钥，分别作为“签名公私钥”和“加密公私钥”。当Number=1时，模块产生1对公私钥，既作为“签名公私钥”，也作为“加密公私钥”，但在逻辑上将这1对公私钥作为2对公私钥进行处理，也即分别作为“签名公私钥”和“加密公私钥”，只不过这2对的密钥内容相同而已。

***pbyVerifyPublicKey***

[in] 输出的“验证公钥”，该参数为空时，接口返回“验证公钥”的长度

***pwVerifyPublicKeyLEN***

[in] 输出的“验证公钥”长度

***pbyWrapublicKey***

[in] 输出的“加密公钥”，该参数为空时，接口返回“加密公钥”的长度

***pwWrapPublicKeyLEN***

[in] 输出的“加密公钥”长度

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则，调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm) 获取错误描述。

|  |  |  |
| --- | --- | --- |
| *错误码* | *错误号* | *含义* |
| SM\_ERR\_FREE | 0x0000 | 模块平台操作成功 |
| SM\_ERR\_UNKNOWN | 0x0016 | 模块平台发生未知错误 |
| SM\_ERR\_DEVICE\_HANDLE | 0x003b | 无效的模块平台设备访问句柄 |
| SM\_ERR\_PARAMETER | 0x0103 | 接口参数有误 |
| SM\_ERR\_PIN\_LENGTH | 0x0112 | 口令长度错误 |

**备注**

该函数在制作认证介质时，需要输入缺省的PIN，模块在内部产生2对（或1对）公私钥，将公私钥存储在认证介质上，并将对应公钥输出给调用方。这些公私钥在成功调用了SM\_Login命令后作为设备的配用密钥使用。

#### 6.3.1.17 SM\_ChangeUserPin

该函数更换用户口令

**SM\_RV SM\_ChangeUserPin(**

**SM\_DEVICE\_HANDLE hDevice, /\* in \*/**

**PSM\_BYTE pbyOldPin, /\* in \*/**

**SM\_UINT uiOldPinLen, /\* in \*/**

**PSM\_BYTE pbyNewPin, /\* in \*/**

**SM\_UINT uiNewPinLen, /\* in \*/**

**PSM\_WORD pwTryNum /\* out \*/**

**);**

**参数**

***hDevice***

[in] 模块平台设备访问句柄。

***pbyOldPin***

[in] 旧口令。

***uiOldPinLen***

[in] 旧口令长度

***pbyNewPin***

[in] 新口令。

***uiNewPinLen***

[in] 新口令长度

***pwTryNum***

[in] 剩余可尝试次数

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则，调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm) 获取错误描述。

#### 6.3.1.18 SM\_BackupAuthDev

该函数用于备份认证介质上的配用密钥。

**SM\_RV SM\_BackupAuthDev**

**(**

**SM\_PIPE\_HANDLE hPipe,**

**PSM\_UCHAR pPin,**

**SM\_UINT uiPinLen,**

**)**

**参数**

***hPipe***

[in] 模块平台的安全管道访问句柄。

***pPin***

[in] 模块平台用户认证介质口令。为'\0'结尾字符串。

***uiPinLen***

[in] 模块平台用户认证介质口令长度，该项目固定PinLen = 8。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则，调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm) 获取错误描述。

**备注**

在备份认证介质时，需要调用方插上备用的认证介质，并输入该认证介质的缺省PIN。在成功调用[SM\_Login](SM_Login.htm)函数登录模块平台以后该函数的调用才有效。

### 6.3.2安全管道访问接口函数

安全管道访问是负责除密钥管理外的安全功能访问操作的子系统。通过访问该子系统，用户可实现数据加解密、数据块摘要等安全功能的应用。

安全模块API5.2提供的安全管道访问接口函数如下：

1．设备基本操作函数（2个）

|  |  |  |
| --- | --- | --- |
| 访问接口函数 | 成对使用 | 说明 |
| [SM\_OpenSecPipe](SM_OpenSecPipe.htm) | 要求 | 打开安全管道 |
| [SM\_CloseSecPipe](SM_CloseSecPipe.htm) | 关闭安全管道 |

2．对称运算函数（8个）

|  |  |  |  |
| --- | --- | --- | --- |
| 访问接口函数 | 用户登录 | 三段式 | 说明 |
|  | \ | 数据对称  加密运算 | 加密初始化 |
|  | \ | 加密数据 |
| [SM\_EncryptFinal](SM_EncryptFinal.htm) | \ | 加密结束 |
|  | \ | 数据对称  解密运算 | 解密初始化 |
| [SM\_DecryptUpdate](SM_DecryptUpdate.htm) | \ | 解密数据 |
|  | \ | 解密结束 |
| [SM\_Encrypt](SM_Encrypt.htm) | \ | \ | 数据块加密运算 |
| [SM\_Decrypt](SM_Decrypt.htm) | \ | \ | 数据块解密运算 |

3．摘要运算函数（4个）

|  |  |  |  |
| --- | --- | --- | --- |
| 访问接口函数 | 用户登录 | 三段式 | 说明 |
| SM\_DigestInit | \ | 数据  摘要运算 | 摘要运算初始化 |
| SM\_DigestUpdate | \ | 数据摘要运算 |
| SM\_DigestFinal | \ | 摘要运算结束 |
| SM\_Digest | \ | \ | 数据块摘要运算 |

4．非对称运算函数（4个）

|  |  |  |
| --- | --- | --- |
| 访问接口函数 | 用户登录 | 说明 |
| SM\_ECCSignature | \ | ECC私有密钥签名运算 |
| SM\_ECCVerify | \ | ECC公开密钥验签运算 |
| SM\_ECCDecrypt | \ | ECC私有密钥解密运算 |
| SM\_ECCEncrypt | \ | ECC公开密钥加密运算 |

5．其他功能函数（1个）

|  |  |  |
| --- | --- | --- |
| 访问接口函数 | 用户登录 | 说明 |
| [SM\_GenRandom](SM_GenRandom.htm) | \ | 随机数生成 |

相关说明文档：，，[设备访问接口函数](Device_Fun.htm)，[密钥访问接口函数](KeyManage_Fun.htm)

#### 6.3.2.1 SM\_OpenSecPipe

函数用来打开安全模块的安全管道。

**SM\_RV SM\_OpenSecPipe**

**(**

**SM\_DEVICE\_HANDLE hDevice,**

**PSM\_PIPE\_HANDLE phPipe**

**);**

**参数**

***hDevice***

[in] 安全模块设备访问句柄。

***phPipe***

[out] 用于存放安全模块的安全管道访问句柄,该参数返回的句柄仅当函数返回值为SM\_ERR\_FREE时有效。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm)获取错误描述。

**备注**

调用进程在安全模块内部分配资源。多个共享进程（或线程）可以独立地调用该函数，得到各自的操作句柄phPipe。该命令成功返回后，如果某个进程（或线程）不再使用安全管道，它必须调用函数[SM\_CloseSecPipe](SM_CloseSecPipe.htm)命令关闭安全管道，释放phPipe。否则，会造成安全模块内部资源无效占用，参考函数SM\_CloseSecPipe的说明。

**示例代码**

**//假定目前安全模块已经成功打开，hDevice已经获取**

**SM\_DEVICE\_HANDLEhDevice = SM\_NULL;**

**SM\_PIPE\_HANDLE hPipe = SM\_NULL;**

**SM\_RV uiRet = 0;**

**uiRet = SM\_OpenSecPipe(hDevice, &hPipe);**

**if(uiRet == SM\_ERR\_FREE)**

**{**

**printf("SM\_OpenSecPipe is success\n");**

**SM\_CloseSecPipe(hDevice,hPipe);**

**}**

**else**

**{**

**printf("SM\_OpenSecPipe =0x%x\n",uiRet);**

**exit(1);**

**}**

**相关函数及说明**

关闭安全管道：[SM\_CloseSecPipe](SM_CloseSecPipe.htm)

#### 6.3.2.2 SM\_CloseSecPipe

函数用来关闭安全模块的安全管道。

**SM\_RV SM\_CloseSecPipe**

**(**

**SM\_DEVICE\_HANDLE hDevice**

**SM\_PIPE\_HANDLE hPipe**

**);**

**参数**

***hDevice***

[in] 安全模块设备访问句柄。

***hPipe***

[in] 安全模块的安全管道访问句柄。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm)获取错误描述。

**备注**

该函数释放安全模块内部对应于hPipe的占用资源。参考函数[SM\_OpenSecPipe](SM_OpenSecPipe.htm)的说明。

**示例代码**

请参考函数[SM\_OpenSecPipe](SM_OpenSecPipe.htm)的示例代码。

**相关函数及说明**

打开安全管道：[SM\_OpenSecPipe](SM_OpenSecPipe.htm)

#### 6.3.2.3 SM\_EncryptInit

该函数执行加密初始化操作。

**SM\_RV SM\_EncryptInit**

**(**

**SM\_PIPE\_HANDLE hPipe,**

**PSM\_BLOB\_KEY pstKey,**

**PSM\_ALGORITHM pstAlgo,**

**);**

**参数**

***hPipe***

[in] 安全模块的安全管道访问句柄。

***pstKey***

[in] 密钥BLOB信息，具体设置请参考结构体SM\_BLOB\_KEY。

***pstAlgo***

[in] 对称算法类型，具体参数设置方式请参考结构体[SM\_ALGORITHM](SM_ALGORITHM.htm)说明。此处指算法工作模式为CBC。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm)获取错误描述。

**备注**

该函数用参数pstKey所标识的密钥来初始化加密过程。该函数和函数[SM\_EncryptUpdate](SM_EncryptUpdate.htm)、[SM\_EncryptFinal](SM_EncryptFinal.htm)一起构成了加密数据的三段式结构。对于大量数据来说，可以首先调用该函数，然后多次调用函数SM\_EncryptUpdate，最后调用一次函数SM\_EncryptFinal完成对数据块的加密；对于少量数据来说，可以首先调用该函数，然后调用一次函数SM\_EncryptFinal完成对数据块的加密。

请参考函数SM\_EncryptUpdate和SM\_EncryptFinal的说明。

**示例代码**

**SM\_DEVICE\_HANDLEhDevice = SM\_NULL;**

**SM\_PIPE\_HANDLE hPipe = SM\_NULL;**

**SM\_RV uiRet = 0;**

**...**

**//假设安全模块已经成功打开，句柄hDevice、hPipe已经获取**

**//同时密钥数据byKey已经产生，使用ALG34\_CBC算法**

**SM\_ALGORITHM Algo;**

**SM\_UINT uiOutLen;**

**SM\_BYTEbyInBuf[100], byOutBuf[100];**

**SM\_BYTEbyContext[SMMA\_ALG34\_CTX\_LEN], byIV[SMMA\_ALG34\_IV\_LEN];**

**SM\_BLOB\_KEY stKey;**

**memset(byInBuf,0x39,100);**

**Algo.AlgoType = SMM\_ALG34\_CBC;**

**Algo.pParameter = byIV;**

**Algo.ulParameterLen = SMMA\_ALG34\_IV\_LEN;**

**Algo.ulReserve = 0;**

**stKey.ulDataLen = SMMA\_ALG34\_KEY\_LEN;**

**stKey.pbyData = &byKey；**

**uiRet = SM\_EncryptInit(hPipe, &stKey, &Algo);**

**if (uiRet != SM\_ERR\_FREE )**

**{**

**printf("SM\_EncryptInit =0x%x\n",uiRet);**

**exit(1);**

**}**

**printf("SM\_EncryptInit is success\n");**

**uiRet = SM\_EncryptUpdate(hPipe,**

**byInBuf,32, byOutBuf, &uiOutLen);**

**if (uiRet != SM\_ERR\_FREE )**

**{**

**printf("SM\_EncryptUpdate =0x%x\n", uiRet);**

**exit(1);**

**}**

**printf("SM\_EncryptUpdate is success\n");**

**uiRet = SM\_EncryptFinal(hPipe, FALSE,**

**byInBuf+32, 10, byOutBuf+32, &uiOutLen);**

**if (uiRet != SM\_ERR\_FREE )**

**{**

**printf("SM\_EncryptFinal =0x%x\n", uiRet);**

**exit(1);**

**}**

**printf("SM\_EncryptFinal is success\n");**

**相关函数及说明**

加密中间数据块：[SM\_EncryptUpdate](SM_EncryptUpdate.htm)

加密最后一块数据：[SM\_EncryptFinal](SM_EncryptFinal.htm)

#### 6.3.2.4 SM\_EncryptUpdate

该函数加密中间数据块。

**SM\_RV SM\_EncryptUpdate**

**(**

**SM\_PIPE\_HANDLE hPipe,**

**PSM\_BYTE pbyDataIn,**

**SM\_UINT uiDataInLen,**

**PSM\_BYTE pbyDataOut,**

**PSM\_UINT puiDataOutLen**

**);**

**参数**

***hPipe***

[in] 安全模块的安全管道访问句柄。

***pbyDataIn***

[in] 明文数据块的首地址。

***uiDataInLen***

[in] 明文数据块的字节数，必须为算法分组长度的整数倍。算法的分组长度可通过调用函数[SM\_GetMechanismInfo](SM_GetMechanismInfo.htm)获取。

***pbyDataOut***

[out] 密文数据块的首地址。

***puiDataOutLen***

[out] 指向返回的密文数据块字节数的指针。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm)获取错误描述。

**备注**

该函数将从参数pbyDataIn所指地址开始的连续ulDataInLen字节的明文数据加密，结果放在从参数pbyDataOut所指地址开始的连续单元中。密文数据的长度与明文数据长度相同。参数pbyDataIn和pbyDataOut可以指向同一地址。可以连续调用该函数对连续的数据块进行加密。

请参考函数SM\_EncryptInit和SM\_EncryptFinal的说明。

**示例代码**

请参考函数[SM\_EncryptInit](SM_EncryptInit.htm)的示例代码。

**相关函数及说明**

加密初始化：[SM\_EncryptInit](SM_EncryptInit.htm)

加密最后一块数据：[SM\_EncryptFinal](SM_EncryptFinal.htm)

#### 6.3.2.5 SM\_EncryptFinal

该函数加密最后一个数据块。

**SM\_RV SM\_EncryptFinal**

**(**

**SM\_PIPE\_HANDLE hPipe,**

**SM\_BOOL bPad,**

**PSM\_BYTE pbyDataIn,**

**SM\_UINT uiDataInLen,**

**PSM\_BYTE pbyDataOut,**

**PSM\_UINT puiDataOutLen**

**);**

**参数**

***hPipe***

[in] 安全模块的安全管道访问句柄。

***bPad***

[in] 加密后的数据是否对明文数据进行填充。取值定义如下：

|  |  |
| --- | --- |
| bPad | 说明 |
| TRUE | 允许对明文数据进行填充，加密后的数据比明文数据长,但密文长度最多比明文长一个加密分组 |
| FALSE | 不允许对明文数据进行填充，加密后的数据长度和明文数据长度相同 |

***pbyDataIn***

[in] 明文数据块的首地址。

***uiDataInLen***

[in] 明文数据块的字节数，可以为任意值。

***pbyDataOut***

[out] 密文数据块的首地址。

***puiDataOutLen***

[out] 指向返回的密文数据块字节数的指针。

当参数*pbyDataOut*为SM\_NULL时，函数将在参数*puiDataOutLen*中返回参数*pbyDataOut*需要分配的缓冲区大小。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm)获取错误描述。

**备注**

该函数将从参数pbyDataIn所指地址开始的连续ulDataInLen字节的明文数据加密，结果放在从参数pbyDataOut所指地址开始的连续单元中。参数pbyDataIn和pbyDataOut可以指向同一地址。该函数将根据参数bPad的值决定是否进行数据填充。该函数执行后，下一轮加密操作如需更换密钥，则必须重新调用函数SM\_EncryptInit；如继续使用上一轮的密钥，则可直接调用函数SM\_EncryptUpdate。

请参考函数SM\_EncryptInit和SM\_EncryptUpdate的说明。

**示例代码**

请参考函数[SM\_EncryptInit](SM_EncryptInit.htm)的示例代码。

**相关函数及说明**

加密初始化：[SM\_EncryptInit](SM_EncryptInit.htm)

加密中间数据块：[SM\_EncryptUpdate](SM_EncryptUpdate.htm)

#### 6.3.2.6 SM\_Encrypt

该函数加密一个任意大小的数据块。

**SM\_RV SM\_Encrypt**

**(**

**SM\_PIPE\_HANDLE hPipe,**

**PSM\_BLOB\_KEY pstKey,**

**PSM\_ALGORITHM pstAlgo,**

**SM\_BOOL bPad,**

**PSM\_BYTEpbyDataIn,**

**SM\_UINTuiDataInLen,**

**PSM\_BYTEpbyDataOut,**

**PSM\_UINTpuiDataOutLen**

**);**

**参数**

***hPipe***

[in] 安全模块的安全管道访问句柄。

***pstKey***

[in] 密钥BOLB信息，具体定义请参考结构体SM\_BLOB\_KEY。

***pstAlgo***

[in] 对称算法类型，具体参数设置方式请参考[SM\_ALGORITHM](SM_ALGORITHM.htm)说明此处指算法工作模式为CBC。

***bPad***

[in] 加密后的数据是否对明文数据进行填充。取值定义如下：

|  |  |
| --- | --- |
| bPad | 说明 |
| TRUE | 允许对明文数据进行填充，加密后的数据比明文数据长,但密文长度最多比明文长一个加密分组 |
| FALSE | 不允许对明文数据进行填充，加密后的数据长度和明文数据长度相同 |

***PbyDataIn***

[in] 明文数据块的首地址。

***uiDataInLen***

[in] 明文数据块的字节数，可以为任意值。

pbyDataOut

[out] 密文数据块的首地址。

***puiDataOutLen***

[out] 指向返回的密文数据块字节数的指针。

当参数pbyDataOut为SM\_NULL时，函数将在参数puiDataOutLen中返回参数pbyDataOut需要分配的缓冲区大小。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm)获取错误描述。

**备注**

该函数将从参数pbyDataIn所指地址开始的连续pulDataOutLen字节的明文数据加密，结果放在从参数pbyDataOut所指地址开始的连续单元中。参数pbyDataIn和pbyDataOut可以指向同一地址。该函数将根据参数bPad的值决定是否进行数据填充。

**示例代码**

**SM\_DEVICE\_HANDLEhDevice = SM\_NULL;**

**SM\_PIPE\_HANDLE hPipe = SM\_NULL;**

**SM\_RV uiRet = 0;**

**...**

**//假设安全模块已经成功打开，句柄hDevice、hPipe已经获取**

**//同时密钥数据byKey已经产生，使用ALG34\_CBC算法**

**SM\_ALGORITHM Algo;**

**SM\_UINTuiOutLen;**

**SM\_BYTE byInBuf[100], byOutBuf[100];**

**SM\_BYTE byContext[SMMA\_ALG34\_CTX\_LEN], byIV[SMMA\_ALG34\_IV\_LEN];**

**SM\_BLOB\_KEY stKey;**

**memset(byInBuf,0x39,100);**

**Algo.AlgoType = SMM\_ALG34\_CBC;**

**Algo.pParameter = byIV;**

**Algo.ulParameterLen = SMMA\_ALG34\_IV\_LEN;**

**Algo. ulReserve = 0;**

**stKey.ulDataLen = SMMA\_ALG34\_KEY\_LEN;**

**stKey.pbyData = &byKey；**

**uiRet = SM\_Encrypt (hPipe, &stKey, &Algo, FALSE,**

**byInBuf, 100, byOutBuf, &uiOutLen);**

**if (uiRet != SM\_ERR\_FREE )**

**{**

**printf("SM\_Encrypt =0x%x\n",uiRet);**

**exit(1);**

**}**

**printf("SM\_Encrypt is success\n");**

#### 6.3.2.7 SM\_DecryptInit

该函数执行解密初始化操作。

**SM\_RV SM\_DecryptInit**

**(**

**SM\_PIPE\_HANDLE hPipe,**

**PSM\_BLOB\_KEY pstKey,**

**PSM\_ALGORITHM pstAlgo,**

**);**

**参数**

***hPipe***

[in] 安全模块的安全管道访问句柄。

***pstKey***

[in] 密钥BLOB信息，具体设置请参考结构体SM\_BLOB\_KEY。

***pAlgo***

[in] 对称算法类型，具体参数设置方式请参考结构体[SM\_ALGORITHM](SM_ALGORITHM.htm)说明。此处指算法工作模式为CBC。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm)获取错误描述。

**备注**

该函数用参数pstKey所标识的密钥来初始化解密过程。该函数和函数SM\_DecryptUpdate、SM\_DecryptFinal一起构成了解密数据块的三段式结构。对于大量数据来说，可以首先调用该函数，然后多次调用函数SM\_DecryptUpdate，最后调用一次函数SM\_DecryptFinal完成对数据块的解密；对于少量数据来说，可以首先调用该函数，然后调用一次函数SM\_DecryptFinal完成对数据块的解密。

请参考函数SM\_DecryptUpdate和SM\_DecryptFinal的说明。

**示例代码**

**SM\_DEVICE\_HANDLEhDevice = SM\_NULL;**

**SM\_PIPE\_HANDLE hPipe = SM\_NULL;**

**SM\_RV uiRet = 0;**

**...**

**//假设安全模块已经成功打开，句柄hDevice、hPipe已经获取**

**//同时密钥hKey已经产生，使用ALG34\_CBC算法**

**SM\_ALGORITHM Algo;**

**SM\_UINTuiOutLen;**

**SM\_BYTE byInBuf[100], byOutBuf[100];**

**SM\_BYTE byContext[SMMA\_ALG34\_CTX\_LEN], byIV[SMMA\_ALG34\_IV\_LEN];**

**SM\_BLOB\_KEY stKey;**

**Algo.AlgoType = SMM\_ALG34\_CBC;**

**Algo.pParameter = byIV;**

**Algo.ulParameterLen = SMMA\_ALG34\_IV\_LEN;**

**Algo. ulReserve = 0;**

**stKey.ulDataLen = sizeof(SM\_HANDLE);**

**stKey.pbyData = &hKey；**

**uiRet = SM\_DecryptInit(hPipe, &stKey, &Algo);**

**if (uiRet != SM\_ERR\_FREE )**

**{**

**printf("SM\_DecryptInit =0x%x\n",uiRet);**

**exit(1);**

**}**

**printf("SM\_DecryptInit is success\n");**

**uiRet = SM\_DecryptUpdate(hPipe,**

**byInBuf,32, byOutBuf, &uiOutLen);**

**if (uiRet != SM\_ERR\_FREE )**

**{**

**printf("SM\_DecryptUpdate =0x%x\n", uiRet);**

**exit(1);**

**}**

**printf("SM\_DecryptUpdate is success\n");**

**uiRet = SM\_DecryptFinal(hPipe, FALSE,**

**byInBuf+32, 10, byOutBuf+32, &uiOutLen);**

**if (uiRet != SM\_ERR\_FREE )**

**{**

**printf("SM\_DecryptFinal =0x%x\n", uiRet);**

**exit(1);**

**}**

**printf("SM\_DecryptFinal is success\n");**

**相关函数及说明**

解密中间数据块：[SM\_DecryptUpdate](SM_DecryptUpdate.htm)

解密最后一块数据：[SM\_DecryptFinal](SM_DecryptFinal.htm)

#### 6.3.2.8 SM\_DecryptUpdate

该函数解密中间数据块。

**SM\_RV SM\_DecryptUpdate**

**(**

**SM\_PIPE\_HANDLEhPipe,**

**PSM\_BYTEpbyDataIn,**

**SM\_UINT uiDataInLen,**

**PSM\_BYTEpbyDataOut,**

**PSM\_UINT puiDataOutLen**

**);**

**参数**

***hPipe***

[in] 安全模块的安全管道访问句柄。

***pbyDataIn***

[in] 密文数据块的首地址。

***ulDataInLen***

[in] 密文数据块的字节数，必须为算法分组长度的整数倍。算法的分组长度可通过调用函数[SM\_GetMechanismInfo](SM_GetMechanismInfo.htm)获取。

***pbyDataOut***

[out] 明文数据块的首地址。

***puiDataOutLen***

[out] 指向返回的明文数据块字节数的指针。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm)获取错误描述。

**备注**

该函数将从参数pbyDataIn所指地址开始的连续ulDataInLen字节的密文数据解密，结果放在从参数pbyDataOut所指地址开始的连续单元中。参数pbyDataIn和pbyDataOut可以指向同一地址。可以连续调用该函数对连续的数据块进行解密。

请参考函数SM\_DecryptInit和SM\_DecryptFinal的说明。

**示例代码**

请参考函数[SM\_DecryptInit](SM_DecryptInit.htm)的示例代码。

**相关函数及说明**

解密初始化：[SM\_DecryptInit](SM_DecryptInit.htm)

解密最后一块数据：[SM\_DecryptFinal](SM_DecryptFinal.htm)

#### 6.3.2.9 SM\_DecryptFinal

该函数解密最后一个数据块。

**SM\_RV SM\_DecryptFinal**

**(**

**SM\_PIPE\_HANDLEhPipe,**

**SM\_BOOL bPad,**

**PSM\_BYTEpbyDataIn,**

**SM\_UINTuiDataInLen,**

**PSM\_BYTEpbyDataOut,**

**PSM\_UINTpuiDataOutLen**

**);**

**参数**

***hPipe***

[in] 安全模块的安全管道访问句柄。

***bPad***

[in] 加密后的数据是否对明文数据进行填充。取值定义如下：

|  |  |
| --- | --- |
| bPad | 说明 |
| TRUE | 允许对明文数据进行填充，加密后的数据比明文数据长,但密文长度最多比明文长一个加密分组 |
| FALSE | 不允许对明文数据进行填充，加密后的数据长度和明文数据长度相同 |

***pDataIn***

[in] 密文数据块的首地址。

***uiDataInLen***

[in] 密文数据块的字节数。如参数*bPad*为TRUE，该参数必须为算法分组长度的整数倍；如否则该参数可以为任意值。

***pDataOut***

[out] 明文数据块的首地址。

***puiDataOutLen***

[out] 指向返回的明文数据块字节数的指针。

当参数pbyDataOut为SM\_NULL时，函数将在参数pulDataOutLen中返回参数pbyDataOut需要分配的缓冲区大小。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm)获取错误描述。

**备注**

该函数将从参数pDataIn所指地址开始的连续ulDataInLen字节的明文数据解密，结果放在从参数pDataOut所指地址开始的连续单元中。参数pDataIn和pDataOut可以指向同一地址。该函数执行后，下一轮解密操作如需更换密钥，则必须重新调用函数SM*\_*DecryptInit；如继续使用上一轮的密钥，则可直接调用函数SM\_DecryptUpdate。

请参考函数SM\_DecryptInit和SM\_DecryptUpdate的说明。

**示例代码**

请参考函数[SM\_DecryptInit](SM_DecryptInit.htm)的示例代码。

**相关函数及说明**

解密初始化：[SM\_DecryptInit](SM_DecryptInit.htm)

解密中间数据块：[SM\_DecryptUpdate](SM_DecryptUpdate.htm)

#### 6.3.2.10 SM\_Decrypt

该函数解密一个任意大小的数据块。

**SM\_RV SM\_Decrypt**

**(**

**SM\_PIPE\_HANDLEhPipe,**

**PSM\_BLOB\_KEY pstKey,**

**PSM\_ALGORITHM pstAlgo,**

**SM\_BOOL bPad,**

**PSM\_BYTEpbyDataIn,**

**SM\_UINTuiDataInLen,**

**PSM\_BYTEpbyDataOut,**

**PSM\_UINTpuiDataOutLen**

**);**

**参数**

***hPipe***

[in] 安全模块的安全管道访问句柄。

***pstKey***

[in] 密钥BOLB信息，具体定义请参考结构体SM\_BLOB\_KEY。

***pstAlgo***

[in] 对称算法类型，具体参数设置方式请参考[SM\_ALGORITHM](SM_ALGORITHM.htm)说明此处指算法工作模式为CBC。

***bPad***

[in] 加密后的数据是否对明文数据进行填充。取值定义如下：

|  |  |
| --- | --- |
| bPad | 说明 |
| TRUE | 允许对明文数据进行填充，加密后的数据比明文数据长,但密文长度最多比明文长一个加密分组 |
| FALSE | 不允许对明文数据进行填充，加密后的数据长度和明文数据长度相同 |

***pbyDataIn***

[in] 密文数据块的首地址。

***uiDataInLen***

[in] 密文数据块的字节数，可以为任意值。

***pbyDataOut***

[out] 明文数据块的首地址。

***puiDataOutLen***

[out] 指向返回的明文数据块字节数的指针。

当参数pbyDataOut为SM\_NULL时，函数将在参数pulDataOutLen中返回参数pbyDataOut需要分配的缓冲区大小。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm)获取错误描述。

**备注**

该函数将从参数pbyDataIn所指地址开始的连续ulDataInLen字节的密文数据解密，结果放在从参数pbyDataOut所指地址开始的连续单元中。参数pbyDataIn和pbyDataOut可以指向同一地址。

**示例代码**

**SM\_DEVICE\_HANDLEhDevice = SM\_NULL;**

**SM\_PIPE\_HANDLE hPipe = SM\_NULL;**

**SM\_RV uiRet = 0;**

**...**

**//假设安全模块已经成功打开，句柄hDevice、hPipe已经获取**

**//同时密钥数据byKey已经产生，使用ALG34\_CBC算法**

**SM\_ALGORITHM Algo;**

**SM\_UINTuiOutLen;**

**SM\_BYTE byInBuf[100], byOutBuf[100];**

**SM\_BYTE byContext[SMMA\_ALG34\_CTX\_LEN], byIV[SMMA\_ALG34\_IV\_LEN];**

**SM\_BLOB\_KEY stKey;**

**Algo.AlgoType = SMM\_ALG34\_CBC;**

**Algo.pParameter = byIV;**

**Algo.ulParameterLen = SMMA\_ALG34\_IV\_LEN;**

**Algo. ulReserve = 0;**

**stKey.ulDataLen = SMMA\_ALG34\_KEY\_LEN;**

**stKey.pbyData = &byKey；**

**uiRet = SM\_Decrypt(hPipe, &stKey, &Algo, FALSE,**

**byInBuf, 100, byOutBuf, &uiOutLen);**

**if (uiRet != SM\_ERR\_FREE )**

**{**

**printf("SM\_Decrypt =0x%x\n",uiRet);**

**exit(1);**

**}**

**printf("SM\_Decrypt is success\n");**

#### 6.3.2.11 SM\_DigestInit

该函数执行对数据块摘要的初始化操作。

**SM\_RV SM\_DigestInit**

**(**

**SM\_PIPE\_HANDLE hPipe,**

**PSM\_BLOB\_KEY pstKey,**

**PSM\_ALGORITHM pstAlgo,**

**);**

**参数**

***hPipe***

[in] 安全模块的安全管道访问句柄。

***pstKey***

[in] 密钥的BLOB信息，具体定义请参考结构体SM\_BLOB\_KEY。

当该参数为SM\_NULL时，表示进行Hash运算，pAlgo指出工作模式采用Hash算法；否则表示进行Mac运算，pAlgo指出工作模式采用Mac算法。

***pstAlgo***

[in] 算法类型，具体参数设置方式请参考[SM\_ALGORITHM](SM_ALGORITHM.htm)说明，此处指算法工作模式采用Hash或Mac算法模式。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm)获取错误描述。

**备注**

该函数用参数pstKey所标识的密钥和参数pAlgo所指定的摘要算法来初始化摘要过程。该函数和函数[SM\_DigestUpdate](mk:@MSITStore:D:\\API业务小组\\项目文档\\项目文档\\帮助文档-API5.0\\api5.0help.chm::/SM_HashUpdate.htm)、[SM\_DigestFinal](mk:@MSITStore:D:\\API业务小组\\项目文档\\项目文档\\帮助文档-API5.0\\api5.0help.chm::/SM_HashFinal.htm)一起构成了摘要数据块的三段式结构。对于大量数据来说，可以首先调用该函数，然后多次调用函数[SM\_DigestUpdate](mk:@MSITStore:D:\\API业务小组\\项目文档\\项目文档\\帮助文档-API5.0\\api5.0help.chm::/SM_HashUpdate.htm)，最后调用一次函数[SM\_DigestFinal](mk:@MSITStore:D:\\API业务小组\\项目文档\\项目文档\\帮助文档-API5.0\\api5.0help.chm::/SM_HashFinal.htm)完成对数据块的摘要；对于少量数据来说，可以首先调用该函数，然后调用一次函数[SM\_DigestFinal](mk:@MSITStore:D:\\API业务小组\\项目文档\\项目文档\\帮助文档-API5.0\\api5.0help.chm::/SM_HashFinal.htm)完成对数据块的摘要。

请参考函数[SM\_DigestUpdate](mk:@MSITStore:D:\\API业务小组\\项目文档\\项目文档\\帮助文档-API5.0\\api5.0help.chm::/SM_HashUpdate.htm)和[SM\_DigestFinal](mk:@MSITStore:D:\\API业务小组\\项目文档\\项目文档\\帮助文档-API5.0\\api5.0help.chm::/SM_HashFinal.htm)的说明。

**示例代码**

**SM\_DEVICE\_HANDLEhDevice = SM\_NULL;**

**SM\_PIPE\_HANDLE hPipe = SM\_NULL;**

**SM\_RV uiRet = 0;**

**...**

**//假设安全模块已经成功打开，句柄hDevice、hPipe已经获取**

**//同时密钥hKey已经产生，使用ALG34\_MAC算法**

**SM\_ALGORITHM Algo;**

**SM\_UINTuiOutLen;**

**SM\_BYTE byInBuf[100], byOutBuf[SMMA\_ALG34\_MAC\_VALUE\_LEN];**

**SM\_BYTE byContext[SMMA\_ALG34\_CTX\_LEN], byIV[SMMA\_ALG34\_IV\_LEN];**

**SM\_BLOB\_KEY stKey;**

**SM\_BLOB\_CONTEXT stContext;**

**Algo.AlgoType = SMM\_ALG34\_MAC;**

**Algo.pParameter = byIV;**

**Algo.ulParameterLen = SMMA\_ALG34\_IV\_LEN;**

**Algo. ulReserve = 0;**

**stKey.ulDataLen = sizeof(SM\_HANDLE);**

**stKey.pbyData = &hKey；**

**uiRet = SM\_DigestInit(hPipe, &stKey, &Algo);**

**if (uiRet != SM\_ERR\_FREE )**

**{**

**printf("SM\_DigestInit =0x%x\n",uiRet);**

**exit(1);**

**}**

**printf("SM\_DigestInit is success\n");**

**uiRet = SM\_DigestUpdate(hPipe,**

**byInBuf,32);**

**if (uiRet != SM\_ERR\_FREE )**

**{**

**printf("SM\_DigestUpdate =0x%x\n", uiRet);**

**exit(1);**

**}**

**printf("SM\_DigestUpdate is success\n");**

**uiRet = SM\_DigestFinal (hPipe,byInBuf+32,**

**10, byOutBuf, &uiOutLen);**

**if (uiRet != SM\_ERR\_FREE )**

**{**

**printf("SM\_DigestFinal =0x%x\n", uiRet);**

**exit(1);**

**}**

**printf("SM\_DigestFinal is success\n");**

**相关函数及说明**

对最后一块数据进行摘要运算：[SM\_DigestFinal](SM_HashFinal.htm)

对中间数据块进行摘要运算：[SM\_DigestUpdate](SM_HashUpdate.htm)

#### 6.3.2.12 SM\_DigestUpdate

该函数对中间数据块进行摘要运算。

**SM\_RV SM\_DigestUpdate**

**(**

**SM\_PIPE\_HANDLEhPipe,**

**PSM\_BYTEpbyDataIn,**

**SM\_UINTuiDataInLen**

**);**

**参数**

***hPipe***

[in] 安全模块的安全管道访问句柄。

***pbyDataIn***

[in] 数据块的首地址。

***uiDataInLen***

[in] 数据块的字节数，必须为摘要分组长度的整数倍。算法的分组长度可通过调用函数[SM\_GetMechanismInfo](SM_GetMechanismInfo.htm)获取。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm)获取错误描述。

**备注**

该函数将从参数pbyDataIn所指地址开始的连续pulDataOutLen字节的数据进行摘要运算，运算结果放在从参数pbyDataOut所指地址开始的连续单元中。该函数与函数[SM\_DigestInit](mk:@MSITStore:D:\\API业务小组\\项目文档\\项目文档\\帮助文档-API5.0\\api5.0help.chm::/SM_HashInit.htm)和[SM\_DigestFinal](mk:@MSITStore:D:\\API业务小组\\项目文档\\项目文档\\帮助文档-API5.0\\api5.0help.chm::/SM_HashFinal.htm)构成了Digest三段式函数，用户可连续调用该函数对连续的数据块进行摘要运算操作。

请参考函数[SM\_DigestInit](mk:@MSITStore:D:\\API业务小组\\项目文档\\项目文档\\帮助文档-API5.0\\api5.0help.chm::/SM_HashInit.htm)和[SM\_DigestFinal](mk:@MSITStore:D:\\API业务小组\\项目文档\\项目文档\\帮助文档-API5.0\\api5.0help.chm::/SM_HashFinal.htm)的说明。

**示例代码**

请参考函数[SM\_DigesthInit](SM_HashInit.htm)的示例代码。

**相关函数及说明**

散列初始化：[SM\_DigestInit](SM_HashInit.htm)

散列最后一块数据：[SM\_DigestFinal](SM_HashFinal.htm)

#### 6.3.2.13 SM\_DigestFinal

该函数对最后一个数据块进行摘要运算。

**SM\_RV SM\_DigestFinal**

**(**

**SM\_PIPE\_HANDLEhPipe,**

**PSM\_BYTE pbyDataIn,**

**SM\_UINTuiDataInLen,**

**PSM\_BYTEpbyDigestValue,**

**PSM\_UINTpuiDigestValLen**

**);**

**参数**

***hPipe***

[in] 安全模块的安全管道访问句柄。

***pbyDataIn***

[in] 输入数据首地址。

***uiDataInLen***

[in] 数据块的字节数，可以为任意值。

***pbyDigestValue***

[out] 摘要值的首地址。

***puiDigestValLen***

[out] 指向返回的摘要值字节数的指针。

当参数pbyDigestValue为SM\_NULL时，函数将在参数pulDigestValLen中返回参数pbyDigestValue需要分配的缓冲区大小。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm)获取错误描述。

**备注**

该函数将从参数pbyDataIn所指地址开始的连续ulDataInLen字节的数据进行数据摘要运算（Hash或Mac）。

请参考函数SM\_DigestInit和SM\_DigestFinal的说明。

**示例代码**

请参考函数[SM\_DigestInit](SM_HashInit.htm)的示例代码。

**相关函数及说明**

摘要初始化：[SM\_DigestInit](SM_HashInit.htm)

摘要最后一块数据：[SM\_DigestFinal](SM_HashFinal.htm)

#### 6.3.2.14 SM\_Digest

该函数执行对数据块摘要的操作。

**SM\_RV SM\_Digest**

**(**

**SM\_PIPE\_HANDLE hPipe,**

**PSM\_BLOB\_KEY pstKey,**

**PSM\_ALGORITHM pstAlgo,**

**PSM\_BYTE pbyDataIn,**

**SM\_UINT uiDataInLen,**

**PSM\_BYTE pbyDigestValue,**

**PSM\_UINT puiDigestValLen**

**);**

**参数**

***hPipe***

[in] 安全模块的安全管道访问句柄。

***pstKey***

[in] 密钥的BLOB信息，具体定义请参考结构体SM\_BLOB\_KEY。

当该参数为SM\_NULL时，表示进行Hash运算，pAlgo指出工作模式采用Hash算法；否则表示进行Mac运算，pAlgo指出工作模式采用Mac算法。

***pstAlgo***

[in] 算法类型，具体参数设置方式请参考[SM\_ALGORITHM](SM_ALGORITHM.htm)说明，此处指算法工作模式采用Hash或Mac算法模式。

***pbyDataIn***

[in] 输入数据首地址。

***uiDataInLen***

[in] 数据块的字节数，可以为任意值。

***pbyDigestValue***

[out] 摘要值的首地址。

***puiDigestValLen***

[out] 指向返回的摘要值字节数的指针。

当参数pbyDigestValue为SM\_NULL时，函数将在参数pulDigestValLen中返回参数pbyDigestValue需要分配的缓冲区大小。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm)获取错误描述。

**备注**

该函数将从参数pbyDataIn所指地址开始的连续pulDataOutLen字节的数据进行摘要运算，运算结果放在从参数pbyDataOut所指地址开始的连续单元中。

**示例代码**

**SM\_DEVICE\_HANDLEhDevice = SM\_NULL;**

**SM\_PIPE\_HANDLE hPipe = SM\_NULL;**

**SM\_RV uiRet = 0;**

**...**

**//假设安全模块已经成功打开，句柄hDevice、hPipe已经获取**

**//同时密钥hKey已经产生，使用ALG34\_MAC算法**

**SM\_ALGORITHM Algo;**

**SM\_UINTuiOutLen;**

**SM\_BYTE byInBuf[100], byOutBuf[SMMA\_ALG34\_MAC\_VALUE\_LEN];**

**SM\_BYTE byContext[SMMA\_ALG34\_CTX\_LEN];**

**SM\_BYTE byIV[SMMA\_ALG34\_IV\_LEN];**

**SM\_BLOB\_KEY stKey;**

**SM\_BLOB\_CONTEXT stContext;**

**Algo.AlgoType = SMM\_ALG34\_MAC;**

**Algo.pParameter = byIV;**

**Algo.ulParameterLen = SMMA\_ALG34\_IV\_LEN;**

**Algo. ulReserve = 0;**

**stKey.ulDataLen = sizeof(SM\_HANDLE);**

**stKey.pbyData = &hKey;**

**uiRet = SM\_Digest(hPipe, &stKey, &Algo,**

**byInBuf, 100, byOutBuf, &uiOutLen);**

**if (uiRet != SM\_ERR\_FREE )**

**{**

**printf("SM\_Digest =0x%x\n",uiRet);**

**exit(1);**

**}**

**printf("SM\_Digest is success\n");**

#### 6.3.2.15 SM\_ECCSignature

该函数使用ECC私钥对数据进行签名运算。

**SM\_RV SM\_ECCSignature**

**(**

**SM\_PIPE\_HANDLE hPipe**

**PSM\_BLOB\_KEY pstPriKey**

**PSM\_ALGORITHM pstAlgo**

**PSM\_BYTE pbyDataIn**

**SM\_UINTuiDataInLen**

**PSM\_BYTEpbyDataSign**

**PSM\_UINTpuiDataSignLen**

**};**

**参数**

***hPipe***

[in] 安全模块的安全管道访问句柄。

***pstPriKey***

[in] 密钥BLOB信息，具体定义请参考结构体SM\_BLOB\_KEY。

***pAlgo***

[in] 非对称算法类型，具体参数设置方式请参考[SM\_ALGORITHM](SM_ALGORITHM.htm)说明，此处指算法工作模式采用ECC。

***pbyDataIn***

[in] 输入数据块的首地址。

***uiDataInLen***

[in] 数据块有效长度必须与相应的算法向匹配。

|  |  |
| --- | --- |
| 算法类型 | 说明 |
| SMM\_ECC\_FP\_SIGN | 进行公钥验签运算，要求明文数据有效长度为指定模长的块长度，取值如下：  >=SMMA\_ECC\_FP\_TZ\_SIG\_MIN\_LEN  <=SMMA\_ECC\_FP\_TZ\_SIG\_MAX\_LEN |

***pbyDataSign***

[out] 输出数据的首地址。

***puiDataSignLen***

[out] 指向返回的数据块字节数的指针。

当参数pbyDataSign为SM\_NULL时，函数将在参数puiDataSignLen中返回参数pDataSign需要分配的缓冲区大小。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm)获取错误描述。

**备注**

无。

**示例代码**

**SM\_PIPE\_HANDLE hPipe = SM\_NULL;**

**PSM\_BYTE pbyRandom = SM\_NULL, pbyDataSign = SM\_NULL;**

**SM\_BYTE \*pDataIn = SM\_NULL;**

**SM\_UINT uiInLen = 32, ulDataSignLen = 64;**

**SM\_RV uiRet = 0;**

**SM\_BLOB\_KEY stPubKey, stPrvKey;**

**SM\_ALGORITHM stAlgoAttr;**

**/假设安全模块已经成功打开， hPipe已经获取**

**stPubKey.uiDataLen = sizeof(SM\_KEY\_HANDLE);**

**stPubKey.pbyData = (SM\_BYTE\*)&g\_stHandle.hPK;**

**stPrvKey.uiDataLen = sizeof(SM\_KEY\_HANDLE);**

**stPrvKey.pbyData = (SM\_BYTE\*)&g\_stHandle.hSK;**

**memset(&stAlgoAttr, 0, sizeof(SM\_ALGORITHM));**

**stAlgoAttr.AlgoType=SMM\_ECC\_FP\_VERIFY;**

**stAlgoAttr.uiReserve =SMMA\_ECC\_FP\_256\_MODULUS\_BITS;**

**memset(pDataIn,0x5a,uiInLen);**

**uiRet = SM\_ECCSignature(hPipe, &stPrvKey, &stAlgoAttr,pDataIn,**

**uiInLen, pbyDataSign, & ulDataSignLen);**

**if(uiRet == SM\_ERR\_FREE)**

**{**

**Printf("SM\_ECCSignature is success\n");**

**}**

**else**

**{**

**printf("SM\_GenRandomerror is =0x%x\n",uiRet);**

**exit(1);**

**}**

**stAlgoAttr.AlgoType=SMM\_ECC\_FP\_VERIFY;**

**uiRet =SM\_ECCVerify(hPipe, &stPubKey, &stAlgoAttr,pDataIn,**

**uiInLen, pbyDataSign, ulDataSignLen);**

**if(uiRet == SM\_ERR\_FREE)**

**{**

**Printf("SM\_ECCVerify is success\n");**

**}**

**else**

**{**

**printf("SM\_ECCVerify error is =0x%x\n",uiRet);**

**exit(1);**

**}**

#### 6.3.2.16 SM\_ECCVerify

该函数使用ECC公开密钥对输入数据进行验证签名运算。

**SM\_RV SM\_Verify**

**(**

**SM\_PIPE\_HANDLE hPipe,**

**PSM\_BLOB\_KEY pstPubKey;**

**PSM\_ALGORITHM pstAlgo,**

**PSM\_BYTE pbyDataIn,**

**SM\_UINTuiDataInLen,**

**PSM\_BYTE pbyDataSign**

**SM\_UINTuiDataSignLen**

**);**

**参数**

***hPipe***

[in] 安全模块的安全管道访问句柄。

***pstPubKey***

[in] 密钥BLOB信息，具体定义请参考结构体SM\_BLOB\_KEY。

***pAlgo***

[in] 非对称算法类型，具体参数设置方式请参考[SM\_ALGORITHM](SM_ALGORITHM.htm)说明，此处指算法工作模式采用ECC。

***pbyDataIn***

[in] 指向待验证签名的明文数据的首地址。

***uiDataInLen***

[in] 待验证签名的明文数据有效长度，该长度必须与相应的算法向匹配。这里定义如下：

|  |  |
| --- | --- |
| 算法类型 | 说明 |
| SMM\_ECC\_FP\_VERIFY | 进行公钥验签运算，要求明文数据有效长度为指定模长的块长度，取值如下：  >=SMMA\_ECC\_FP\_TZ\_SIG\_MIN\_LEN  <=SMMA\_ECC\_FP\_TZ\_SIG\_MAX\_LEN |

***pbyDataSign***

[in] 指向签名结果数据块的首地址。

***uiDataSignLen***

[in] 签名结果数据块的有效长度。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm)获取错误描述。

**示例代码**

见**SM\_ECCSignature**接口示例代码。

#### 6.3.2.17 SM\_GenRandom

该函数从安全模块上获取一定长度随机数。

**SM\_RV SM\_GenRandom**

**(**

**SM\_PIPE\_HANDLE hPipe,**

**SM\_WORD wRndNum,**

**PSM\_BYTEpbyRandom,**

**SM\_UINTuiRandomLen**

**);**

**参数**

***hPipe***

[in] 安全模块的安全管道访问句柄。

***wRndNum***

[in] 指定随机数单元编号，该项目无效。

***pbyRandom***

[out] 用于存放获取的随机数的首地址。

***uiRandomLen***

[in] 待获取的随机数的字节数，以字节为单位。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm)获取错误描述。

**备注**

该函数将启动安全模块内部的物理噪声源，并将产生的随机数输出。

**示例代码**

**SM\_PIPE\_HANDLE hPipe = SM\_NULL;**

**PSM\_BYTE pbyRandom = SM\_NULL;**

**SM\_UINT uiRandomLen = 64;**

**SM\_RV uiRet = 0;**

**/假设安全模块已经成功打开， hPipe已经获取**

**uiRet = SM\_GenRandom(hPipe,0, pbyRandom, uiRandomLen);**

**if(uiRet == SM\_ERR\_FREE)**

**{**

**printf("SM\_GenRandom is success\n");**

**}**

**else**

**{**

**Printf("SM\_GenRandomerror is =0x%x\n",uiRet);**

**Exit(1);**

**}**

#### 6.3.2.18 SM\_ECCEncrypt

该函数使用用户指定ECC公开密钥句柄对输入数据进行ECC加密运算操作。

**SM\_RV WINAPI SM\_ECCEncrypt(**

**SM\_PIPE\_HANDLE hPipe,**

**PSM\_BLOB\_KEY pstPubKey,**

**PSM\_ALGORITHM pstAlgo,**

**PSM\_BYTE pbyDataIn,**

**SM\_UINT uiDataInLen,**

**PSM\_BLOB\_ECCCIPHER pstEccCipher**

**)**

**参数**

***hPipe***

[in] 模块平台的安全管道访问句柄。

***pstPubKey***

[in] 公开密钥句柄。具体定义请参考结构体SM\_BLOB\_KEY。

***pstAlgo***

[in] ECC算法类型，具体参数设置方式请参考[SM\_ALGORITHM](SM_ALGORITHM.htm)

***pbyDataIn***

[in] 输入明文数据的首地址。

***uiDataInLen***

[in] 输入明文数据的有效长度，以字节为单位。明文的有效长度的范围应在1至1024字节之间。

***pstEccCipher***

[out] 用于存放输出密文数据的结构体首地址，具体定义请参考结构体**SM\_BLOB\_ECCCIPHER**

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则，调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm) 获取错误描述。

**示例代码**

**SM\_PIPE\_HANDLE hPipe = SM\_NULL;**

**PSM\_BYTE pbyDataIn = SM\_NULL, pbyDataOut = SM\_NULL;**

**SM\_UINT uiDataInLen = 32, uiOutLen = 32;**

**SW\_TEST\_HANDLE g\_stHandle;**

**SM\_BLOB\_KEY stPubKey, stPrvKey;**

**SM\_ALGORITHM stAlgoAttr;**

**SM\_BLOB\_ECCCIPHER stEccCipher;**

**SM\_RV uiRet = 0;**

**//假设安全模块已经成功打开， hPipe, g\_stHandle.hSK与g\_stHandle.Hpk**

**已经获取**

**//ECC算法**

**stPubKey.uiDataLen = sizeof(SM\_KEY\_HANDLE);**

**stPubKey.pbyData = (SM\_BYTE\*)&g\_stHandle.hPK;**

**stPrvKey.uiDataLen = sizeof(SM\_KEY\_HANDLE);**

**stPrvKey.pbyData = (SM\_BYTE\*)&g\_stHandle.hSK;**

**memset(&stAlgoAttr, 0, sizeof(SM\_ALGORITHM));**

**stAlgoAttr.AlgoType = 273;**

**stAlgoAttr.uiReserve = 256;**

**memset(&stCipher, 0, sizeof(SM\_BLOB\_ECCCIPHER));**

**memset(pbyDataIn,0x5a, uiDataInLen);**

**uiRet = SM\_ECCEncrypt(hPipe, &stPubKey, &stAlgoAttr,**

**pbyDataIn, uiInLen, & stEccCipher);**

**if(uiRet == SM\_ERR\_FREE)**

**{**

**Printf("SM\_ECCEncrypt is success\n");**

**}**

**else**

**{**

**printf("SM\_ECCEncrypt error is =0x%x\n",uiRet);**

**exit(1);**

**}**

**stAlgoAttr.AlgoType = 274;**

**memset(pbyDataOut ,0x0 uiDataInLen);**

**uiRet = SM\_ECCDecrypt(hPipe, &stPrvKey, &stAlgoAttr,**

**&stCipher, pbyDataOut, &uiOutLen);**

**if(uiRet == SM\_ERR\_FREE)**

**{**

**Printf("SM\_ECCDecrypt is success\n");**

**}**

**else**

**{**

**printf("SM\_ECCDecrypt error is =0x%x\n",uiRet);**

**exit(1);**

**}**

#### 6.3.2.19 SM\_ECCDecrypt

该函数清除模块平台内部维护的非对称商用算法公开密钥

**SM\_ECCDecrypt(**

**SM\_PIPE\_HANDLE hPipe,**

**PSM\_BLOB\_KEY pstPriKey,**

**PSM\_ALGORITHM pstAlgo,**

**PSM\_BLOB\_ECCCIPHER pstEccCipher,**

**PSM\_BYTE pbyDataOut,**

**PSM\_UINT puiDataOutLen**

**)**

**参数**

***hPipe***

[in] 模块平台的安全管道访问句柄。

***pstPriKey***

[in] 私有密钥句柄。具体定义请参考结构体SM\_BLOB\_KEY。

***pstAlgo***

[in] ECC算法类型，具体参数设置方式请参考[SM\_ALGORITHM](SM_ALGORITHM.htm)

***pstEccCipher***

[in] 用于存放输入密文数据的结构体首地址，具体定义请参考结构体**SM\_BLOB\_ECCCIPHER**

***pbyDataOut***

[out] 输出明文数据的首地址。

***puiDataOutLen***

[out] 输出明文数据的有效长度，以字节为单位。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则，调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm) 获取错误描述。

**示例代码**

见**SM\_ECCDecrypt**接口示例代码。

#### 6.3.2.20 SM\_CloseAllSecPipe

该函数用来关闭所有安全管道句柄。

**SM\_RV SM\_CloseAllSecPipe**

**(**

**SM\_DEVICE\_HANDLE hDevice //in**

**);**

**参数**

***hDevice***

[in] 密码机设备访问句柄。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm) 获取错误描述。

#### 6.3.2.21 SM\_Login

该函数使用用户认证介质口令登录模块平台。

**SM\_RV SM\_Login**

**(**

**SM\_PIPE\_HANDLE hPipe,**

**PSM\_UCHAR pPin,**

**SM\_UINT uiPinLen,**

**PSM\_WORD pwTryNum**

**)**

**参数**

***hPipe***

[in] 模块平台的安全管道访问句柄。

***pPin***

[in] 模块平台用户认证介质口令。为'\0'结尾字符串。

***uiPinLen***

[in] 模块平台用户认证介质口令长度，该项目固定PinLen = 8。

***pwTryNum***

[out] 用于存放剩于的用户认证介质口令尝试次数。当且仅当函数返回值为SM\_ERR\_USER\_PASSWORD时该参数返回的数值有效。

|  |  |
| --- | --- |
| *pwTryNum* | *含义* |
| 0x0000 | 用户认证介质口令锁死 |
| 0x0001-0x000E | 用户认证介质口令剩余可尝试次数 |

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则，调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm) 获取错误描述。

**备注**

该函数调用成功之前，所有非对称密钥对等敏感数据均处于加密状态，禁止进行私钥的导入\导出以及私钥加解密运算等操作；该函数调用成功后，上述数据即可在模块内部脱密，并允许私钥的导入\导出以及私钥加解密运算等操作。

2.在同一进程内，当一个安全管道调用该函数成功后，则其余所有安全管道均自动进入登录状态，无需再次调用该函数；而当一个安全管道成功调用[SM\_Logout](SM_Logout.htm)退出登录后，则其余所有安全管道均自动退出登录状态。

#### 6.3.2.22 SM\_Logout

 函数用来退出登录状态。

**SM\_RV SM\_Logout**

**(**

**SM\_PIPE\_HANDLE hPipe**

**)**

**参数**

***hPipe***

[in] 模块平台的安全管道访问句柄。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则，调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm) 获取错误描述。

### 6.3.3密钥访问管理接口函数

#### 6.3.3.1 SM\_ImportKey

该函数向模块平台内部导入对称商用算法所需的对称密钥WK。

**SM\_RV SM\_ImportKey**

**(**

**SM\_PIPE\_HANDLE hPipe,**

**PSM\_BYTE pbyKey,**

**SM\_WORD wKeyLen,**

**SM\_KEY\_HANDLE hKEK,**

**PSM\_ALGORITHM pstKEKAlgo,**

**PSM\_KEY\_ATTRIBUTE pstKeyAttr,**

**PSM\_KEY\_HANDLE phKey**

**}**

**参数**

***hPipe***

[in] 模块平台的安全管道访问句柄。

***pbyKey***

[in] 待导入的对称密钥数据WK的首地址。

***wKeyLen***

[in] 待导入的对称密钥数据WK的长度，明文WK的长度可参照函数[SM\_GetMechanismInfo](SM_GetMechanismInfo.htm),被对称商用算法保护(加密)的WK的长度为保护商用算法分组长度的整数倍，同时被保护后的WK的长度应该大于明文WK的长度，被公开密钥保护的WK的长度由公开密钥的模长决定,如：模长为1024，WK=128BYTES。

***hKEK***

[in] 保护(解密)WK的密钥句柄，依据pKEKAlgo的不同，可为非对称密钥句柄、主密钥句柄、对称密钥句柄，或为NULL即WK未被保护。

***pstKEKAlgo***

[in] 指定WK的保护(解密)模式，可以采用对称商用算法ECB模式解密、对称商用算法CBC模式解密、非对称密钥解密三种方式，具体参数设置方式请参考数据结构[SM\_ALGORITHM](SM_ALGORITHM.htm),如果hKEK置为NULL，则该参数必须置为NULL。

***pstKeyAttr***

[in] 指定导入的对称密钥WK的属性，具体参数设置方式请参考数据结构[SM\_KEY\_ATTRIBUTE](SM_KEY_ATTRIBUTE.htm)，注意：SM\_KEY\_ATTRIBUTE中的参数KeyType只能设为SM\_KEY\_ALG\*。

***phKey***

[out] 用于存放对称密钥句柄的地址。该参数返回的句柄仅当函数返回值为SM\_ERR\_FREE时有效。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则，调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm) 获取错误描述。

**示例代码**

**SM\_DEVICE\_HANDLEhDevice = SM\_NULL;**

**SM\_PIPE\_HANDLE hPipe = SM\_NULL;**

**SW\_TEST\_HANDLE g\_stHandle;**

**SM\_BYTE \*pbyKeyData = NULL;**

**SM\_UINT uiKeyLen = 16;**

**SM\_KEY\_ATTRIBUTE stKeyAttr;**

**SM\_RV uiRet = 0;**

**SM\_UINT i = 0, iNo = 0;**

**SM\_RV uiRet = 0;**

**...**

**//假设安全模块已经成功打开，句柄hDevice、hPipe已经获取**

**//同时密钥数据byKey已经产生，使用ALG34算法**

**memset(pstKeyAttr, 0, sizeof(SM\_KEY\_ATTRIBUTE));**

**pstKeyAttr->uiObjectClass = SMO\_SECRET\_KEY;**

**pstKeyAttr->KeyType = 42;**

**pstKeyAttr->pParameter = SM\_NULL;**

**pstKeyAttr->uiParameterLen = 0;**

**pstKeyAttr->uiKeyLabel = 1;**

**pstKeyAttr->uiFlags=SMKA\_EXTRACTABLE |SMKA\_ENCRYPT | SMKA\_DECRYPT;**

**uiRet=SM\_ImportKey(hDevice, pbyKeyData, (SM\_WORD)uiKeyLen,**

**NULL, NULL, &stKeyAttr, &g\_stHandle.hWK);**

**if(uiRet == SM\_ERR\_FREE)**

**{**

**printf("SM\_ImportKey is success\n");**

**}**

**else**

**{**

**printf("SM\_ImportKey error is =0x%x\n",uiRet);**

**exit(1);**

**}**

**uiRet=SM\_DestroyKey(hPipe ,g\_stHandle.hWK);**

**if(uiRet == SM\_ERR\_FREE)**

**{**

**printf("SM\_DestroyKey is success\n");**

**}**

**else**

**{**

**printf("SM\_DestroyKey error is =0x%x\n",uiRet);**

**exit(1);**

**}**

#### 6.3.3.2 SM\_ExportKey

该函数从模块平台内部导出对称商用算法所需的对称密钥。

**SM\_RV WINAPI SM\_ExportKey**

**(**

**SM\_PIPE\_HANDLE hPipe,**

**SM\_KEY\_HANDLE hKey,**

**SM\_KEY\_HANDLE hKEK,**

**PSM\_ALGORITHM pstKEKAlgo,**

**PSM\_BYTE pbyKey,**

**PSM\_WORD pwKeyLen**

**}**

**参数**

***hPipe***

[in] 模块平台的安全管道访问句柄。

***hKey***

[in] 模块平台的对称密钥句柄，为[SM\_GenerateKey](SM_GenerateKey.htm)或[SM\_ImportKey](SM_ImportKey.htm)函数的返回值。

***hKEK***

[in] 保护(加密)WK的密钥句柄,依据pKEKAlgo的不同，可为公开密钥句柄、主密钥句柄、对称密钥句柄，或为NULL即不保护。

***pstKEKAlgo***

[in] 指定WK的保护(加密)模式，可以采用对称商用算法ECB模式加密、对称商用算法CBC模式加密、公开密钥加密三种方式，具体参数设置方式请参考数据结构[SM\_ALGORITHM](SM_ALGORITHM.htm),如果hKEK置为NULL，则该参数必须置为NULL。

***pbyKey***

[out] 对称密钥WK的首地址。该参数返回值仅当函数返回值为SM\_ERR\_FREE时有效。

***pwKeyLen***

[out] 对称密钥WK的长度指针。该参数返回值仅当函数返回值为SM\_ERR\_FREE时有效。

当**pbyKey**为NULL时，函数将在参数pwKeyLen中返回参数**pbyKey**需要分配的缓冲区大小。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则，调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm) 获取错误描述。

**备注**

该函数将指定的模块平台内部维护的WK输出。参考SM\_GenerateKey、SM\_ImportKey和SM\_DestroyKey函数的说明。

在成功调用[SM\_Login](SM_Login.htm)函数登录模块平台以后该函数的调用才有效。

#### 6.3.3.2SM\_DestroyKey

该函数清除模块平台内部维护的对称商用算法所需的密钥WK。

**SM\_RV WINAPI SM\_DestroyKey**

**(**

**SM\_PIPE\_HANDLE hPipe,**

**SM\_KEY\_HANDLE hKey**

**)**

**参数**

***hPipe***

[in] 模块平台的安全管道访问句柄。

***hKey***

[in] 密钥句柄。为[SM\_GenerateKey](SM_GenerateKey.htm)或[SM\_ImportKey](SM_ImportKey.htm)函数的返回值。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则，调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm) 获取错误描述。

**备注**

该函数清除指定的模块平台内部维护的对称工作密钥WK明文。参考SM\_GenerateKey、SM\_ImportKey和SM\_ExportKey函数的说明。

**示例代码**

见接口SM\_ImportKey接口示例代码。

#### 6.3.3.3 SM\_ImportPrivateKey

该函数向模块平台内部导入非对称商用算法私有密钥。

**SM\_RV WINAPI SM\_ImportPrivateKey**

**(**

**SM\_PIPE\_HANDLE hPipe,**

**PSM\_BYTE pbyPrivateKey,**

**SM\_WORD wPriKeyLen,**

**SM\_KEY\_HANDLE hKEK,**

**PSM\_ALGORITHM pstKEKAlgo,**

**PSM\_KEY\_ATTRIBUTE pstPriKeyAttr,**

**PSM\_KEY\_HANDLE phPrivateKey**

**)**

**参数**

***hPipe***

[in] 模块平台的安全管道访问句柄。

***pbyPrivateKey***

[in] 私有密钥数据的首地址。

***wPriKeyLen***

[in] 私有密钥数据的长度。

***hKEK***

[in] 保护商用算法类型，密钥类型只能为对称商用算法，具体参数设置方式请参考SM\_ALGORITHM说明,如果保护商用算法密钥访问句柄置为NULL，则该参数也置为NULL。

***pstKEKAlgo***

[in] 指定WK的保护(解密)模式，可以采用对称商用算法ECB模式解密、对称商用算法CBC模式解密、私有密钥解密三种方式，具体参数设置方式请参考数据结构[SM\_ALGORITHM](SM_ALGORITHM.htm),如果hKEK置为NULL，则该参数必须置为NULL。

***pstPriKeyAttr***

[in] 指定私有密钥的属性，具体参数设置方式请参考数据结构[SM\_KEY\_ATTRIBUTE](SM_KEY_ATTRIBUTE.htm)。

***phPrivateKey***

[out] 用于存放私有密钥句柄的地址。该参数返回的句柄仅当函数返回值为SM\_ERR\_FREE时有效。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则，调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm) 获取错误描述。

**备注**

该函数将从外部得到的私有密钥置入模块平台，供加解密过程使用。对于不再使用的密钥应及时调用SM\_DestroyPrivateKey函数，以便释放模块平台内部占用的资源。参考SM\_GenerateKeyPair、SM\_ImportPublicKey、SM\_ExportPublicKey、SM\_ExportPrivateKey、SM\_DestroyPublicKey和SM\_DestroyPrivateKey函数的说明。

在成功调用[SM\_Login](SM_Login.htm)函数登录模块平台以后该函数的调用才有效。

**示例代码**

**SM\_DEVICE\_HANDLEhDevice = SM\_NULL;**

**SM\_PIPE\_HANDLE hPipe = SM\_NULL;**

**SW\_TEST\_HANDLE g\_stHandle;**

**SM\_BYTE \*pbyPrvKeyData = NULL， \*pbyPubKeyData = NULL;**

**PSM\_BYTE pbyPrvKeyData = SM\_NULL, pbyPubKeyData = SM\_NULL;**

**SM\_UINT uiPrvKeyLen = 32，uiPubKeyLen = 64;**

**PSM\_KEY\_ATTRIBUTE pstPrvKeyAttr，pstPubKeyAttr;**

**SM\_RV uiRet = 0;**

**...**

**/**

**//假定目前模块平台已打开，设备访问句柄hDevice和安全管道访问句柄hPipe打开**

**SM\_UINT i = 0, iNo = 0;**

**SM\_RV uiRet = 0;**

**pstPrvKeyAttr->uiObjectClass = SMO\_PRIVATE\_KEY;**

**pstPrvKeyAttr->KeyType = 6；**

**pstPrvKeyAttr->uiKeyLabel = 1;**

**memcpy(pstPrvKeyAttr->byStartDate, byDate, sizeof(SM\_BYTE) \* 4);**

**memcpy(pstPrvKeyAttr->byEndDate, byDate, sizeof(SM\_BYTE) \* 4);**

**//ECC**

**pstPubKeyAttr->pParameter = pstECCPara;**

**pstPubKeyAttr->uiParameterLen = sizeof(SM\_ECC\_PARAMETER);**

**pstPrvKeyAttr->pParameter= pstECCPara;**

**pstPrvKeyAttr->uiParameterLen = sizeof(SM\_ECC\_PARAMETER);**

**pstECCPara->uiModulusBits = SMMA\_ECC\_FP\_256\_MODULUS\_BITS；**

**//导入私钥**

**uiRet=SM\_ImportPrivateKey(hPipe,pbyPrvKeyData,**

**(SM\_WORD)uiPrvKeyLen, SM\_NULL, SM\_NULL, pstPrvKeyAttr,**

**&g\_stHandle.hSK);**

**if(uiRet == SM\_ERR\_FREE)**

**{**

**printf("SM\_ImportPrivateKey is success\n");**

**}**

**else**

**{**

**printf("SM\_ImportPrivateKeyerror is =0x%x\n",uiRet);**

**exit(1);**

**}**

**//导入公钥**

**uiRet = SM\_ImportPublicKey(hPipe, pbyPubKeyData,**

**(SM\_WORD)uiPubKeyLen, pstPubKeyAttr, &g\_stHandle.hPK);**

**if(uiRet == SM\_ERR\_FREE)**

**{**

**printf("SM\_ImportPublicKey is success\n");**

**}**

**else**

**{**

**printf("SM\_ImportPublicKey error is =0x%x\n",uiRet);**

**exit(1);**

**}**

**//销毁私公钥**

**uiRet = SM\_DestroyPrivateKey(hPipe, g\_stHandle.hSK);**

**if(uiRet == SM\_ERR\_FREE)**

**{**

**printf("SM\_DestroyPrivateKey is success\n");**

**}**

**else**

**{**

**printf("SM\_DestroyPrivateKey error is =0x%x\n",uiRet);**

**exit(1);**

**}**

**//销毁公钥**

**uiRet = SM\_DestroyPublicKey(hPipe, g\_stHandle.hPK);**

**if(uiRet == SM\_ERR\_FREE)**

**{**

**printf("SM\_DestroyPublicKey is success\n");**

**}**

**else**

**{**

**printf("SM\_DestroyPublicKey error is =0x%x\n",uiRet);**

**exit(1);**

**}**

#### 6.3.3.4 SM\_ImportPublicKey

该函数向模块平台内部导入非对称商用算法公开密钥。

**SM\_RV WINAPI SM\_ImportPublicKey**

**(**

**SM\_PIPE\_HANDLE hPipe,**

**PSM\_BYTE pbyPublicKey,**

**SM\_WORD wPubKeyLen,**

**PSM\_KEY\_ATTRIBUTE pstPubKeyAttr,**

**PSM\_KEY\_HANDLE phPublicKey**

**)**

**参数**

***hPipe***

[in] 模块平台的安全管道访问句柄。

***pbyPublicKey***

[in] 待导入的公开密钥数据的首地址。

***wPubKeyLen***

[in] 待导入的公开密钥数据的长度。

***pstPubKeyAttr***

[in] 定公开密钥属性，具体参数设置方式请参考[SM\_KEY\_ATTRIBUTE](SM_KEY_ATTRIBUTE.htm)说明。

***phPublicKey***

[out] 公开密钥句柄。该参数返回的句柄仅当函数返回值为SM\_ERR\_FREE时有效。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则，调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm) 获取错误描述。

**备注**

该函数将从外部得到的公开密钥置入模块平台，供加解密过程使用。对于不再使用的密钥应及时调用SM\_DestroyPublicKey函数，以便释放模块平台内部占用的资源。参考SM\_GenerateKeyPair、SM\_ImportPrivateKey、SM\_ExportPublicKey、SM\_ExportPrivateKey、SM\_DestroyPublicKey和SM\_DestroyPrivateKey函数的说明。

**示例代码**

见**SM\_ImportPrivateKey**接口示例代码。

#### 6.3.3.5 SM\_DestroyPrivateKey

该函数清除模块平台内部维护的非对称商用算法私有密钥。

**SM\_RV SM\_DestroyPrivateKey**

**(**

**SM\_PIPE\_HANDLE hPipe,**

**SM\_KEY\_HANDLE hPrivateKey**

**)**

**参数**

***hPipe***

[in] 模块平台的安全管道访问句柄。

***hPrivateKey***

[in] 私有密钥句柄。为[SM\_GenerateKeyPair](SM_GenerateKeyPair.htm)或[SM\_ImportPrivateKey](SM_ImportPrivateKey.htm)函数的返回值。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则，调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm) 获取错误描述。

**备注**

该函数清除指定的模块平台内部维护的私有密钥。参考SM\_GenerateKeyPair、SM\_ImportPrivateKey、SM\_ImportPublicKey、SM\_ExportPrivateKey、SM\_ExportPublicKey和SM\_DestroyPublicKey函数的说明。

**示例代码**

见**SM\_ImportPrivateKey**接口示例代码。

#### 6.3.3.6 SM\_DestroyPublicKey

该函数清除模块平台内部维护的非对称商用算法公开密钥

**SM\_RV SM\_DestroyPublicKey**

**(**

**SM\_PIPE\_HANDLE hPipe,**

**SM\_KEY\_HANDLE hPublicKey**

**)**

**参数**

***hPipe***

[in] 模块平台的安全管道访问句柄。

***hPublicKey***

[in] 公开密钥句柄。为[SM\_GenerateKeyPair](SM_GenerateKeyPair.htm)或[SM\_ImportPublicKey](SM_ImportPublicKey.htm)函数的返回值。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则，调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm) 获取错误描述。

**备注**

该函数清除指定的模块平台内部维护的公开密钥。参考SM\_GenerateKeyPair、SM\_ImportPublicKey、SM\_ImportPrivateKey、SM\_ExportPublicKey、SM\_ExportPrivateKey和SM\_DestroyPrivateKey函数的说明。

**示例代码**

见**SM\_ImportPrivateKey**接口示例代码。

#### 6.3.3.8 SM\_GenerateKey

该函数在模块平台内部产生对称商用算法所需的对称密钥WK。

**SM\_RV WINAPI SM\_GenerateKey**

**(**

**SM\_PIPE\_HANDLE hPipe,**

**PSM\_KEY\_ATTRIBUTE pKeyAttr,**

**PSM\_KEY\_HANDLE phKey**

**)**

**参数**

***hPipe***

[in] 模块平台的安全管道访问句柄。

***pKeyAttr***

[in] 指定对称密钥WK属性，具体参数设置方式请参考数据结构[SM\_KEY\_ATTRIBUTE](SM_KEY_ATTRIBUTE.htm)。

***phKey***

[out] 用于存放对称密钥句柄的地址。该参数返回的句柄仅当函数返回值为SM\_ERR\_FREE时有效。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则，调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm) 获取错误描述。

|  |  |  |
| --- | --- | --- |
| *错误码* | *错误号* | *含义* |
| SM\_ERR\_FREE | 0x0000 | 模块平台操作成功 |
| SM\_ERR\_RESOURCE | 0x0003 | 模块平台内部资源不足 |
| SM\_ERR\_PIPE\_HANDLE | 0x0005 | 模块平台安全管道句柄错 |
| SM\_ERR\_KEY\_TYPE | 0x0010 | 无效的密钥类型 |
| SM\_ERR\_UNKNOWN | 0x0016 | 模块平台发生未知错误 |
| SM\_ERR\_MALLOC | 0x0017 | 内存分配失败 |
| SM\_ERR\_PARAMETER | 0x0030 | 参数不匹配 |

**备注**

该函数将启动模块平台内部的物理噪声源，随机产生对称商用算法所需的WK，并存放在模块平台内部，供加密过程使用。对于不再使用的密钥应及时调用SM\_DestroyKey函数，以便释放模块平台内部占用的资源。参考SM\_ImportKey、SM\_ExportKey和SM\_DestroyKey函数的说明。

**示例代码**

//假定目前模块平台的安全管道访问句柄hPipe已获得,并且模块平台已经登陆。

SM\_PIPE\_HANDLE hPipe;

SM\_KEY\_HANDLE hKey;

SM\_KEY\_ATTRIBUTE KeyAttr;

KeyAttr.KeyType = SM\_KEY\_ALG34;

KeyAttr.ulFlags = SM\_KA\_PLAIN\_EXTRACTABLE;

KeyAttr.pParameter = NULL;

KeyAttr.ulParameterLen = 0;

SM\_RV ulRet = SM\_GenerateKey(hPipe, &KeyAttr, &hKey);

if (SM\_ERR\_FREE == ulRet)

{

　ulRet = SM\_DestroyKey(hPipe, hKey);

　printf("SM\_GenerateKey is success\n");

}

Else

{

　printf("SM\_GenerateKey is error errorCode=0x%x\n", ulRet);

　exit(1);

}

**必要条件**

[头文件及库文件对应表](../接口文档/DllCall.htm)。

**相关函数及说明**

导出对称密钥：[SM\_ExportKey](SM_ExportKey.htm)

销毁对称密钥：[SM\_DestroyKey](SM_DestroyKey.htm)

#### 6.3.3.9 SM\_GenerateKeyPair

该函数在模块平台内部生成非对称密钥对。

**SM\_RV WINAPI SM\_GenerateKeyPair**

**(**

**SM\_PIPE\_HANDLE hPipe,**

**PSM\_KEY\_ATTRIBUTE pPubKeyAttr,**

**PSM\_KEY\_HANDLE phPublicKey,**

**PSM\_KEY\_ATTRIBUTE pPriKeyAttr,**

**PSM\_KEY\_HANDLE phPrivateKey**

**)**

**参数**

***hPipe***

[in] 模块平台的安全管道访问句柄。

***pPubKeyAttr***

[in] 指向SM\_KEY\_ATTRIBUTE结构的指针，指定公钥属性，具体参数设置方式请参考数据结构[SM\_KEY\_ATTRIBUTE](SM_KEY_ATTRIBUTE.htm)。

***phPublicKey***

[out] 公开密钥句柄。该参数返回的句柄仅当函数返回值为SM\_ERR\_FREE时有效。

***pPriKeyAttr***

[in] 指向SM\_KEY\_ATTRIBUTE结构的指针，指定私有密钥属性，具体参数设置方式请参考数据结构[SM\_KEY\_ATTRIBUTE](SM_KEY_ATTRIBUTE.htm)。

***phPrivateKey***

[out] 私有密钥句柄。该参数返回的句柄仅当函数返回值为SM\_ERR\_FREE时有效。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则，调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm) 获取错误描述。

|  |  |  |
| --- | --- | --- |
| *错误码* | *错误号* | *含义* |
| SM\_ERR\_FREE | 0x0000 | 模块平台操作成功 |
| SM\_ERR\_RESOURCE | 0x0003 | 模块平台内部资源不足 |
| SM\_ERR\_PIPE\_HANDLE | 0x0005 | 模块平台安全管道句柄错 |
| SM\_ERR\_KEY\_TYPE | 0x0010 | 无效的密钥类型 |
| SM\_ERR\_FUNC\_SUPORT | 0x0015 | 不支持该调用 |
| SM\_ERR\_UNKNOWN | 0x0016 | 模块平台发生未知错误 |
| SM\_ERR\_MALLOC | 0x0017 | 内存分配失败 |
| SM\_ERR\_PARAMETER | 0x0103 | 参数不匹配 |

**备注**

该函数产生非对称商用算法密钥对，并存放在模块平台内部。对于不再使用的非对称商用算法密钥应及时调用SM\_DestroyPublicKey和SM\_DestroyPrivateKey函数，以便释放模块平台内部占用的资源。参考SM\_ImportPublicKey、SM\_ImportPrivateKey、SM\_ExportPublicKey、SM\_ExportPrivateKey、SM\_DestroyPublicKey和SM\_DestroyPrivateKey函数的说明。

**示例代码**

//假定目前模块平台的安全管道访问句柄hPipe已获得。

SM\_PIPE\_HANDLE hPipe;

SM\_KEY\_HANDLE hPublicKey,hPrivateKey,hPublicKey1,hPrivateKey1;

SM\_KEY\_ATTRIBUTE PubKeyAttr,PriKeyAttr;

SM\_ULONG ulPubKeylen,ulPriKeylen;

SM\_MECHANISM\_INFO Mech;

SM\_GetMechanismInfo(hDevice,SM\_RSA,&Mech);

auto\_ptr<unsigned char> ptrDataIn(new unsigned char[Mech.ulMaxKeySize]), ptrDataOut(new unsigned char[Mech.ulMaxKeySize]);

unsigned char\* PubKeyBuf = ptrDataIn.get();

unsigned char\* PriKeyBuf = ptrDataOut.get();

SM\_RSA\_PARAMETER RsaPara;

RsaPara.ulModulusBits = 1024;

RsaPara.ulCodeType = SM\_RSA\_REF;

PubKeyAttr.KeyType = SM\_KEY\_RSA\_PUBLIC;

PubKeyAttr.ulFlags = SM\_KA\_PLAIN\_EXTRACTABLE|SM\_KA\_MODIFIABLE;

PubKeyAttr.pParameter = &RsaPara;

PubKeyAttr.ulParameterLen = sizeof(SM\_RSA\_PARAMETER);

//

PriKeyAttr.KeyType = SM\_KEY\_RSA\_PRIVATE;

PriKeyAttr.ulFlags = SM\_KA\_PLAIN\_EXTRACTABLE|SM\_KA\_MODIFIABLE;

PriKeyAttr.pParameter = &RsaPara;

PriKeyAttr.ulParameterLen = sizeof(SM\_RSA\_PARAMETER);

//

SM\_RV ulRet = SM\_GenerateKeyPair(hPipe,&PubKeyAttr,&hPublicKey,&PriKeyAttr,&hPrivateKey);

if(SM\_ERR\_FREE == ulRet)

{

   printf("SM\_GenerateKeyPair is success\n");

   ulRet = SM\_ExportPublicKey(hPipe,hPublicKey,&PubKeyAttr,PubKeyBuf,&ulPubKeylen);

   if(SM\_ERR\_FREE != ulRet)

   {

     printf("SM\_ExportPublicKey is error errorCode=0x%x\n",ulRet);

   }

   printf("SM\_ExportPublicKey is success\n");

   ulRet = SM\_ExportPrivateKey(hPipe,hPrivateKey,NULL,NULL,&PriKeyAttr,PriKeyBuf,&ulPriKeylen);

   if(SM\_ERR\_FREE != ulRet)

   {

     printf("SM\_ExportPrivateKey is error errorCode=0x%x\n",ulRet);

   }

   printf("SM\_ExportPrivateKey is success\n");

   ulRet = SM\_ImportPublicKey(hPipe,PubKeyBuf,ulPubKeylen,&PubKeyAttr,&hPublicKey1);

   if(SM\_ERR\_FREE != ulRet)

   {

     printf("SM\_ImportPublicKey is error errorCode=0x%x\n",ulRet);

   }

printf("SM\_ImportPublicKey is success\n");

ulRet = SM\_ImportPrivateKey(hPipe,PriKeyBuf,ulPriKeylen,NULL,NULL,&PriKeyAttr,&hPrivateKey1);

if(SM\_ERR\_FREE != ulRet)

{

  printf("SM\_ImportPrivateKey is error errorCode=0x%x\n",ulRet);

}

printf("SM\_ImportPrivateKey is success\n");

ulRet = SM\_DestroyPublicKey(hPipe,hPublicKey);

ulRet = SM\_DestroyPrivateKey(hPipe,hPrivateKey);

ulRet = SM\_DestroyPublicKey(hPipe,hPublicKey1);

ulRet = SM\_DestroyPrivateKey(hPipe,hPrivateKey1);

}

Else

{

  printf("SM\_GenerateKeyPair is error errorCode=0x%x\n",ulRet);

  exit(1);

}

**必要条件**

[头文件及库文件对应表](../接口文档/DllCall.htm)。

**相关函数及说明**

导出公开密钥：[SM\_ExportPublicKey](SM_ExportPublicKey.htm)

导出私有密钥：[SM\_ExportPrivateKey](SM_ExportPrivateKey.htm)

销毁公开密钥：[SM\_DestroyPublicKey](SM_DestroyPublicKey.htm)

销毁私有密钥：[SM\_DestroyPrivateKey](SM_DestroyPrivateKey.htm)

#### 6.3.3.10 SM\_ExportPrivateKey

该函数从模块平台内部导出非对称商用算法私有密钥。

**SM\_RV WINAPI SM\_ExportPrivateKey**

**(**

**SM\_PIPE\_HANDLE hPipe,**

**SM\_KEY\_HANDLE hPrivateKey,**

**SM\_KEY\_HANDLE hKEK,**

**PSM\_ALGORITHM pKEKAlgo,**

**PSM\_UCHAR pPrivateKey,**

**PSM\_UINTpuiPriKeyLen**

**)**

**参数**

***hPipe***

[in] 模块平台的安全管道访问句柄。

***hPrivateKey***

[in] 模块平台的私有密钥句柄，为[SM\_GenerateKeyPair](SM_GenerateKey.htm)或[SM\_ImportPrivateKey](SM_ImportPrivateKey.htm)函数的返回值。

***hKEK***

[in] 保护商用算法密钥访问句柄,依据所采用保护商用算法的不同，可为主密钥句柄、对称密钥句柄，或为NULL即不保护。

***pKEKAlgo***

[in] 保护商用算法类型，商用算法类型只能为对称商用算法,具体参数设置方式请参考[SM\_ALGORITHM](SM_ALGORITHM.htm)说明,如果hKEK置为NULL，则该参数必须置为NULL。

***pPrivateKey***

[out] 私有密钥的首地址。该参数返回值仅当函数返回值为SM\_ERR\_FREE时有效。

***puiPriKeyLen***

[out] 指向私有密钥的长度指针。该参数返回值仅当函数返回值为SM\_ERR\_FREE时有效。

当pPrivateKey为NULL时，函数将在参数pulPriKeyLen中返回参数pPrivateKey需要分配的缓冲区大小。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则，调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm) 获取错误描述。

|  |  |  |
| --- | --- | --- |
| *错误码* | *错误号* | *含义* |
| SM\_ERR\_FREE | 0x0000 | 模块平台操作成功 |
| SM\_ERR\_RESOURCE | 0x0003 | 模块平台内部资源不足 |
| SM\_ERR\_PIPE\_HANDLE | 0x0005 | 模块平台安全管道句柄错 |
| SM\_ERR\_KEY\_HANDLE | 0x0006 | 无效的密钥句柄 |
| SM\_ERR\_NOT\_LOGIN | 0x000d | 模块平台没有登录 |
| SM\_ERR\_MECH\_TYPE | 0x000f | 无效的商用算法类型 |
| SM\_ERR\_KEY\_TYPE | 0x0010 | 无效的密钥类型 |
| SM\_ERR\_UNKNOWN | 0x0016 | 模块平台发生未知错误 |
| SM\_ERR\_MALLOC | 0x0017 | 内存分配失败 |
| SM\_ERR\_PARAMETER | 0x0030 | 参数不匹配 |
| SM\_ERR\_RSA\_ENCODE | 0x0037 | 编解码错误 |
| SM\_ERR\_TIME\_OUT | 0x003a | 模块平台操作超时 |

**备注**

该函数将指定的模块平台内部维护的私有密钥输出。参考SM\_GenerateKeyPair、SM\_ImportPublicKey、SM\_ImportPrivateKey、SM\_ExportPublicKey、SM\_DestroyPublicKey和SM\_DestroyPrivateKey函数的说明。

在成功调用[SM\_Login](SM_Login.htm)函数登录模块平台以后该函数的调用才有效。

**示例代码**

请参考函数[SM\_GenerateKeyPair](SM_GenerateKeyPair.htm)的示例代码。

**必要条件**

[头文件及库文件对应表](../接口文档/DllCall.htm)。

**相关函数及说明**

产生非对称密钥对：[SM\_GenerateKeyPair](SM_GenerateKeyPair.htm)

导入私有密钥：　　[SM\_ImportPrivateKey](SM_ImportPrivateKey.htm)

销毁私有密钥：　　[SM\_DestroyPrivateKey](SM_DestroyPrivateKey.htm)

#### 6.3.3.11 SM\_ExportPublicKey

该函数从模块平台内部导出非对称商用算法公开密钥。

**SM\_RV WINAPI SM\_ExportPublicKey**

**(**

**SM\_PIPE\_HANDLE hPipe,**

**SM\_KEY\_HANDLE hPublicKey,**

**PSM\_UCHAR pPublicKey,**

**PSM\_ULONG pulPubKeyLen**

**)**

**参数**

***hPipe***

[in] 模块平台的安全管道访问句柄。

***hPublicKey***

[in] 模块平台的公开密钥句柄，为[SM\_GenerateKeyPair](SM_GenerateKeyPair.htm)或[SM\_ImportPublicKey](SM_ImportPublicKey.htm)函数的返回值。

***pPublicKey***

[out] 公开密钥的首地址。该参数返回值仅当函数返回值为SM\_ERR\_FREE时有效。

***pulPubKeyLen***

[out] 指向公开密钥的长度指针。该参数返回值仅当函数返回值为SM\_ERR\_FREE时有效。

当pPublicKey为NULL时，函数将在参数pulPubKeyLen中返回参数pPublicKey需要分配的缓冲区大小。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则，调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm) 获取错误描述。

|  |  |  |
| --- | --- | --- |
| *错误码* | *错误号* | *含义* |
| SM\_ERR\_FREE | 0x0000 | 模块平台操作成功 |
| SM\_ERR\_RESOURCE | 0x0003 | 模块平台内部资源不足 |
| SM\_ERR\_PIPE\_HANDLE | 0x0005 | 模块平台安全管道句柄错 |
| SM\_ERR\_KEY\_HANDLE | 0x0006 | 无效的密钥句柄 |
| SM\_ERR\_MECH\_TYPE | 0x000f | 无效的商用算法类型 |
| SM\_ERR\_KEY\_TYPE | 0x0010 | 无效的密钥类型 |
| SM\_ERR\_UNKNOWN | 0x0016 | 模块平台发生未知错误 |
| SM\_ERR\_MALLOC | 0x0017 | 内存分配失败 |
| SM\_ERR\_PARAMETER | 0x0030 | 参数不匹配 |
| SM\_ERR\_RSA\_ENCODE | 0x0037 | 编解码错误 |
| SM\_ERR\_TIME\_OUT | 0x003a | 模块平台操作超时 |

**备注**

该函数将指定的模块平台内部维护的公开密钥输出。参考SM\_GenerateKeyPair、SM\_ExportPrivateKey、SM\_ImportPrivateKey、SM\_ImportPublicKey、SM\_DestroyPublicKey和SM\_DestroyPrivateKey函数的说明。

在成功调用[SM\_Login](SM_Login.htm)函数登录模块平台以后该函数的调用才有效。

**示例代码**

请参考函数[SM\_GenerateKeyPair](SM_GenerateKeyPair.htm)的示例代码。

**必要条件**

[头文件及库文件对应表](../接口文档/DllCall.htm)。

**相关函数及说明**

产生非对称密钥对：[SM\_GenerateKeyPair](SM_GenerateKeyPair.htm)

导入公开密钥：　　[SM\_ImportPublicKey](SM_ImportPublicKey.htm)

销毁公开密钥：　　[SM\_DestroyPublicKey](SM_DestroyPublicKey.htm)

#### 6.3.3.12 SM\_UpdateKeyPair

该函数用于更换认证介质上的配用公私钥。

**SM\_RV WINAPI SM\_UpdateKeyPair**

**(**

**SM\_PIPE\_HANDLE hPipe,**

**PSM\_BLOB\_KEYpstPublicKey,**

**PSM\_BLOB\_KEY pstPrivateKey,**

**SM\_WORD wKeyUse,**

**PSM\_UCHAR pPin,**

**SM\_UINT uiPinLen,**

**)**

**参数**

***hPipe***

[in] 模块平台的安全管道访问句柄。

***pstPublicKey***

[in] 更换的公钥BLOB结构，参见结构体SM\_BLOB\_KEY。当该参数与pstPrivateKey参数赋为NULL时，接口内部默认重新产生公私钥对。当结构体成员pbyData可以为公钥数据或者公钥句柄，相应的长度成员uiDataLen为公钥数据长，或者句柄长，模块内部配用公钥将更换为用户指定公钥；

***pstPrivateKey***

[in] 更换的私钥BLOB结构,参见结构体SM\_BLOB\_KEY。当该参数与pstPublicKey参数赋为NULL时，接口内部默认重新产生公私钥对。当结构体成员pbyData可以为私钥数据或者私钥句柄，相应的长度成员uiDataLen为私钥数据长，或者句柄长，模块内部配用私钥将更换为用户指定私钥；

***wKeyUse***

[in] 该参数指明更换的公私钥用于ECC加解密或者ECC签名验签。取值如下：

SMKF\_UPDATE\_KEY\_PAIR\_SIGN 0表示用于签名验签。

SMKF\_UPDATE\_KEY\_PAIR\_WRAP 1表示用于加解密；

***pPin***

[in] 模块平台用户口令。为'\0'结尾字符串。

***uiPinLen***

[in] 模块平台用户口令长度，该项目固定PinLen = 8。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则，调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm) 获取错误描述。

**备注**

该函数该用于更换认证介质上的配用公私钥，用户在更换认证介质上的配用公私钥时，需要调用方插上配套的认证介质，并输入该认证介质的PIN。调用SM\_GetCfgKeyHandle获取配用公私钥句柄。用户需保证输入的公私钥数据或句柄是成对的。

在成功调用[SM\_Login](SM_Login.htm)函数登录模块平台以后该函数的调用才有效。

#### 6.3.3.13SM\_GetCfgKeyHandle

该函数用于获取认证介质上的配用公私钥。

**SM\_RV SM\_GetCfgKeyHandle(**

**SM\_PIPE\_HANDLE hPipe, /\* in \*/**

**PSM\_BLOB\_KEY pstKey, /\* in \*/**

**PSM\_KEY\_HANDLE phKey /\* out \*/**

**);**

**参数**

***hPipe***

[in] 模块平台的安全管道访问句柄。

***pstKey***

[in] 配用密钥BLOB结构，参见结构体SM\_BLOB\_KEY。

成员pbyData可取值：

SMCK\_ECC\_VERIFY\_PUBLIC 0x0001

SMCK\_ECC\_SIGN\_PRIVATE 0x0002

SMCK\_ECC\_ENC\_PUBLIC 0x0003

SMCK\_ECC\_DEC\_PRIVATE 0x0004

成员uiDataLen取值sizeof（SM\_UINT）;

***phKey***

[in] 获得的配用密钥句柄。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则，调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm) 获取错误描述。

**备注**

该函数该用于获取认证介质上的配用密钥，当不再使用密钥句柄时，应当调用SM\_CloseTokKeyHdl关闭句柄。

在成功调用[SM\_Login](SM_Login.htm)函数登录模块平台以后该函数的调用才有效。

#### 6.3.3.14SM\_CloseTokKeyHdl

该函数用于关闭认证介质上的配用公私钥句柄。

**SM\_RV SM\_CloseTokKeyHdl(**

**SM\_PIPE\_HANDLE hPipe, /\* in \*/**

**SM\_KEY\_HANDLE hKey /\* in \*/**

**);**

**参数**

***hPipe***

[in] 模块平台的安全管道访问句柄。

***phKey***

[in] 待关闭的配用密钥句柄。

**返回值**

如返回值为SM\_ERR\_FREE，则调用成功；否则，调用失败。如果函数执行失败，可以使用函数[SM\_GetErrorString](../接口文档/SM_GetErrorString.htm) 获取错误描述。

**备注**

该函数该关闭获取认证介质上的配用密钥，见SM\_GetCfgKeyHandle。

在成功调用[SM\_Login](SM_Login.htm)函数登录模块平台以后该函数的调用才有效

## 6.4 错误代码

安全模块API 5.2版本支持的错误代码的详细定义请参见头文件"sm\_api.h"。