ICS 35.030
CCS L 80

GM

中华人民共和国密码行业标准

GM/T 0019—2023
代替 GM/T 0019—2012

通用密码服务接口规范

Universal cryptography service interface specification

2023-12-04 发布　　2024-06-01 实施

国家密码管理局　发布

目　次

前　言

本文件按照 GB/T 1.1—2020《标准化工作导则　第 1 部分：标准化文件的结构和起草规则》的规定起草。

本文件代替 GM/T 0019—2012《通用密码服务接口规范》，与 GM/T 0019—2012 相比，除结构调整和编辑性改动外，主要技术变化如下：

a) 删除了“用户证书列表”中的“数据长度(字节)”列(见 2012 年版的表 2)“密钥容器列表”中的“数据长度(字节)”列(见 2012 年版的表 3)；

b) 删除了“证书中 DN 的结构(见表 4)”内容(见 2012 年版的 5.2.4)；

c) 删除了“通用密码服务在公钥密码应用技术体系框架内的位置”图；(见 2012 年版的图 1)；

d) 增加了“解析 PKCS #7 格式的签名数据”接口(见 7.4.16)；

e) 增加了“验证方法”(见第 8 章)；

f) 增加了“SM9 密码算法应用接口”(见附录 A)；

g) 增加了“证书被吊销的原因”错误码 (见表 B.1)；

h) 删除了“SAR_CertRevokedErr”的宏描述、预定义值、说明 (见 2012 年版的附录 A)。

请注意本文件的某些内容可能涉及专利。本文本的发布机构不承担识别专利的责任。

本文件由密码行业标准化技术委员会提出并归口。

本文件起草单位：北京数字认证股份有限公司、格尔软件股份有限公司、北京海泰方圆科技股份有限公司、无锡江南信息安全工程技术中心、上海数字证书认证中心有限公司、中电科网络安全科技股份有限公司、山东得安信息技术有限公司、北京国脉信安科技有限公司、中国电子技术标准化研究院。

本文件主要起草人：赵松、王银平、刘平、李述胜、郑强、谭武征、蒋红宇、柳增寿、王玉林、王中武、马洪富、高志权、孔凡玉、袁峰、上官晓丽、蔡一鸣、黄晶晶。

本文件及其所代替文件的历次版本发布情况为：

——2012 年首次发布为 GM/T 0019—2012；

——本次为第一次修订。

通用密码服务接口规范

1 范围

本文件规定了通用密码服务接口的数据结构、接口描述、函数定义要求，描述了相应验证方法。

本文件适用于公钥密码应用技术体系下密码应用服务的开发，密码应用支撑平台的研制及检测，密码设备应用系统的开发。

2 规范性引用文件

下列文件中的内容通过文中的规范性引用而构成本文件必不可少的条款。其中，注日期的引用文件，仅该日期对应的版本适用于本文件；不注日期的引用文件，其最新版本（包括所有的修改单）适用于本文件。

GB/T 25069 信息安全技术 术语

GB/T 41389 信息安全技术 SM9 密码算法使用规范

GM/T 0003.1 信息安全技术 SM2 椭圆曲线公钥密码算法 第 1 部分：总则

GM/T 0006 密码应用标识规范

GM/T 0009 SM2 密码算法使用规范

GM/T 0010 SM2 密码算法加密签名消息语法规范

GM/T 0015 数字证书格式

GM/T 0016 智能密码钥匙应用接口规范

GM/T 0018 密码设备应用接口规范

GM/T 0094 公钥密码应用技术体系框架规范

GM/Z 4001 密码术语

3 术语和定义

GB/T 25069、GM/Z 4001 界定的以及下列术语和定义适用于本文件。

3.1

容器 container

密码设备中用于保存密钥所划分的唯一性存储空间。

4 缩略语

下列缩略语适用于本文件。

API：应用程序接口/应用接口（Application Program Interface）

CA：证书认证机构（Certification Authority）

CN：通用名（Common Name）

CRL：证书撤销列表（Certificate Revocation List）

DER：可区分编码规则（Distinguished Encoding Rules）

DN：可识别名(Distinguished Name)
LDAP：轻量级目录访问协议(Lightweight Directory Access Protocol)
OCSP：在线证书状态协议(Online Certificate Status Protocol)
OID：对象标识符(Object Identifier)
PKCS：公钥密码标准(the Public-Key Cryptography Standard)

5 算法标识和数据结构

5.1 算法标识与常量定义

本文件所使用常量定义、各类算法标识和证书解析标识的具体定义应符合 GM/T 0006 的规定。

5.2 密码服务接口数据结构定义和说明

5.2.1 常量定义

数据常量标识的定义见表 1。

表 1 常量定义

常量名	取值	描述
SGD_MAX_COUNT	64	枚举出的对象数量最大值
SGD_MAX_NAME_SIZE	256	证书某项信息的字符串长度最大值

5.2.2 用户证书列表

用户证书列表见表 2。

表 2 用户证书列表

字段名称	含义
certCount	证书总数
Certificate	DER 编码的数字证书
certificateLen	数字证书的长度
containerName	容器名称
containerNameLen	容器名称的长度
keyUsage	密钥用途

实际数据结构定义：

```
typedef struct SGD_USR_CERT_ENUMLIST_ {
  unsigned int certCount;
    unsigned char * certificate[SGD_MAX_COUNT];
    unsigned int certificateLen[SGD_MAX_COUNT];
    unsigned char * containerName[SGD_MAX_COUNT];
    unsigned int containerNameLen[SGD_MAX_COUNT];
    unsigned int keyUsage[SGD_MAX_COUNT];
```

} SGD_USR_CERT_ENUMLIST;

5.2.3 密钥容器信息列表

密钥容器信息列表见表3。

表3 密钥容器信息列表

字段名称	含义
keyPairCount	密钥容器信息总数
containerName	容器名称
containerNameLen	容器名称的长度
keyUsage	密钥用途,1:加密;2:签名;3:密钥交换
keyType	密钥类型,1:SM2;2:RSA1024;3:RSA2048;4:RSA3072;5:RSA4096; 6:SM9(SM9密码算法定义应符合附录A)

实际数据结构定义:

```
typedef struct SGD_ KEYCONTAINERINFO_ENUMLIST_ {
  unsigned int keyPairCount;
    unsigned char * containerName[SGD_MAX_COUNT];
    unsigned int containerNameLen[SGD_MAX_COUNT];
    unsigned int keyUsage[SGD_MAX_COUNT];
  unsigned int keyType[SGD_MAX_COUNT];
} SGD_ KEYCONTAINERINFO_ENUMLIST;
```

6 密码服务接口

6.1 通用密码服务接口在公钥密码基础设施应用技术体系框架中的位置

通用密码服务通过统一的密码服务接口,向典型密码服务层和应用层提供证书解析、证书认证、信息的机密性、完整性和不可否认性通用密码服务,将上层应用的密码服务请求转化为具体的基础密码操作请求,通过统一的密码设备应用接口调用相应的密码设备实现具体的密码运算和密钥操作。

通用密码服务在公钥密码基础设施应用技术体系框架内的位置应符合GM/T 0094的规定。

6.2 密码服务接口组成和功能说明

6.2.1 概述

通用密码服务接口由以下部分组成:

a) 环境类函数;

b) 证书类函数;

c) 密码运算类函数;

d) 消息类函数。

本文件中的相关接口,均采用C语言形式定义,使用其他语言开发本文件中的接口均可参考C语言接口的定义。

6.2.2 环境类函数

环境类函数负责创建和管理程序空间中所需的各种资源、信号，并确保程序空间在应用程序运行期间不会被非法访问，造成信息泄漏。环境类函数负责完成与密码设备的安全连接，确保后续的操作是在安全的程序空间中进行。环境类函数还负责在用户与密码设备之间创建和管理应用接口句柄。可创建两种类型的应用接口句柄，一类是普通用户，该类型的应用接口句柄标识该用户是普通用户，只能访问密码设备中属于自己的信息和数据；另一类是管理员，该类型的应用接口句柄标识该用户是管理员，可以管理普通用户的应用接口句柄。

应用程序在使用通用密码服务接口时，首先要调用初始化环境函数(SAF_Initialize)创建和初始化安全的程序空间，完成与密码设备连接和初始化工作。应用程序在调用任何密码服务函数前应确保运行环境安全，可以采用白名单机制或对接入设备进行认证的方式，进行任何私钥运算之前应首先调用用户登录函数(SAF_Login)，创建应用接口句柄。当不再调用密码服务函数时，应调用注销登录函数(SAF_Logout)注销安全访问令牌，防止密码设备被非法访问。在终止应用程序之前，应调用清除环境函数(SAF_Finalize)，中止与密码设备的连接，销毁所创建的安全的程序空间，防止由于内存残留所带来的安全风险。

6.2.3 证书类函数

证书类函数设置各类数字证书到应用接口会话环境中、验证用户证书和获取数字证书或CRL，提供包括证书获取、CRL获取、CA根证书设置、用户证书验证和用户证书信息获取一系列具体函数。应用程序通过调用证书类函数，实现基于数字证书的身份认证，从证书中获取有关信息，实现授权管理、访问控制安全机制。本文件中涉及的数字证书格式应符合GM/T 0015。

6.2.4 密码运算类函数

密码运算类函数负责具体与密码设备交互实现具体的密码运算，并将密码运算后的结果返回给应用程序，是应用程序实现数据保密性、完整性和不可抵赖性安全机制的基础。

密码运算类函数提供包括base64编解码、随机数生成、密码杂凑以及各种对称和非对称密码运算。密码服务函数支持定长数据和不定长数据的密码运算，对于定长数据可以直接调用相关函数进行处理；对于不定长数据，应先创建相应的密码运算对象，然后调用相应的函数对数据进行持续处理。当数据处理完时，要调用相应的函数表示数据处理完，最后要调用相应函数销毁相应的密码运算对象。

6.2.5 消息类函数

消息类函数的功能是将数据按照数字信封格式进行封装，实现应用系统互联互通和信息共享，使用RSA算法时符合PKCS #7，使用国密算法时符合GM/T 0010。

消息类函数提供了数据编解码、签名数据编解码、数字信封编解码功能，数据编解码格式应符合PKCS #7(RSA算法)或GM/T 0010(SM2算法)、签名数据编解码格式应符合PKCS #7(RSA算法)或GM/T 0010(SM2算法)和PKCS #7(RSA算法)，数字信封编解码格式应符合GM/T 0010(SM2算法)，能够非常方便应用程序实现身份认证、保密性、完整性和不可否认性安全措施。

7 密码服务接口函数定义

7.1 环境类函数

7.1.1 概述

环境类函数负责创建和管理程序空间中所需的各种资源、信号，并确保程序空间在应用程序运行期

间不会被非法访问，造成信息泄漏，包括以下具体函数：

a) 初始化环境：SAF_Initialize；
b) 清除环境：SAF_Finalize；
c) 获取接口版本信息：SAF_GetVersion；
d) 用户登录：SAF_Login ；
e) 修改口令：SAF_ChangePin；
f) 注销登录：SAF_Logout。

各函数返回值应符合附录 B 中的错误代码定义。

7.1.2 初始化环境

原型： int SAF_Initialize(void * * phAppHandle,unsigned char * pucCfgFilePath)；

描述： 初始化密码服务程序空间。

参数： phAppHandle[in/out] 输入并返回应用接口句柄

pucCfgFilePath[in] 配置文件全路径名，配置信息自定义，建议包括：密码设备类型、密码设备动态库名称、设备的配置文件、应用策略

返回值： 0 成功

非 0 失败，返回错误代码

7.1.3 清除环境

原型： int SAF_Finalize(void * hAppHandle)；

描述： 清除应用程序空间。

参数： hAppHandle[in] 应用接口句柄

返回值： 0 成功

非 0 失败，返回错误代码

7.1.4 获取接口版本信息

原型： int SAF_GetVersion(unsigned int * puiVersion)；

描述： 取接口对应标准的版本号。

参数： puiVersion [out] 版本号

返回值： 0 成功

非 0 失败，返回错误代码

备注： 版本号的格式为 0xAABBCCCC，其中 AA 为主版本号，BB 为次版本号，CCCC 为修改号。

7.1.5 用户登录

原型：

```
int SAF_Login (
        void * hAppHandle,
        unsigned int uiUsrType,
        unsigned char * pucContainerName,
        unsigned int uiContainerNameLen,
        unsigned char * pucPin,
        unsigned int uiPinLen,
        unsigned int * puiRemainCount);
```

描述：用户登录密码设备，建立安全令牌。

参数：

hAppHandle[in]	应用接口句柄
uiUsrType[in]	用户类型，当为 0 时表示管理员登录 为 1 时表示用户登录
pucContainerName [in]	容器名(或多个参数的拼接值，多个参数时用 # # # 拼接)或密钥检索号
uiContainerNameLen[in]	容器名或密钥检索号的长度
pucPin[in]	设备口令
uiPinLen[in]	设备口令长度
puiRemainCount[out]	口令剩余重试次数

返回值：

0	成功
非 0	失败，返回错误代码

备注：在服务器端使用加密机或加密卡时，当设备符合 GM/T 0016 规范时，pucContainerName 用于标识指定的容器名或多个参数的拼接值，当设备遵循 GM/T 0018 规范时，pucContainerName 用于标识密码设备内部的密钥位置。

7.1.6 修改口令

原型：

```
int SAF _ChangePin (
        void * hAppHandle,
        unsigned int uiUsrType,
        unsigned char * pucContainerName,
        unsigned int uiContainerNameLen,
        unsigned char * pucOldPin,
        unsigned int uiOldPinLen,
        unsigned char * pucNewPin,
        unsigned int uiNewPinLen,
        unsigned int * puiRemainCount);
```

描述：修改设备口令，本函数仅适用于符合 GM/T 0016 规范的客户端密码设备。

参数：	hAppHandle[in]	应用接口句柄
	uiUsrType[in]	用户类型，当为 0 时表示管理员登录 为 1 时表示用户登录
	pucContainerName [in]	容器名或密钥检索号
	uiContainerNameLen[in]	容器名或密钥检索号的长度
	pucOldPin[in]	设备当前口令
	uiOldPinLen[in]	设备当前口令长度
	pucNewPin[in]	设备新口令
	uiNewPinLen[in]	设备新口令长度
	puiRemainCount[out]	口令剩余重试次数
返回值：	0	成功
	非 0	失败，返回错误代码

7.1.7 注销登录

原型：

```
int SAF_Logout(
        void * hAppHandle,
        unsigned int uiUsrType)
```

描述：设备注销登录

参数：	hAppHandle[in]	应用接口句柄
	uiUsrType [in]	用户类型，当为 0 时表示管理员登录 为 1 时表示用户登录
返回值：	0	成功
	非 0	失败，返回错误代码

7.2 证书类函数

7.2.1 概述

证书类函数设置各类数字证书到应用接口会话环境中、验证用户证书和获取数字证书或 CRL，提供包括证书获取、CRL 获取、CA 根证书设置、用户证书验证和用户证书信息获取等一系列具体函数，包括以下具体函数：

a) 添加根 CA 证书：SAF_AddTrustedRootCaCertificate；
b) 获取根 CA 证书个数：SAF_GetRootCaCertificateCount；
c) 获取根 CA 证书：SAF_GetRootCaCertificate；
d) 删除根 CA 证书：SAF_RemoveRootCaCertificate；
e) 添加 CA 证书：SAF_AddCaCertificate；
f) 获取 CA 证书个数：SAF_GetCaCertificateCount；
g) 获取 CA 证书：SAF_GetCaCertificate；

h) 删除 CA 证书：SAF_RemoveCaCertificate；
i) 添加 CRL：SAF_AddCrl；
j) 验证用户证书：SAF_VerifyCertificate；
k) 根据 CRL 文件获取用户证书注销状态：SAF_VerifyCertificateByCrl；
l) 根据 OCSP 获取证书状态：SAF_GetCertificateStateByOCSP；
m) 通过 LDAP 方式获取证书：SAF_GetCertificateFromLdap；
n) 通过 LDAP 方式获取证书对应的 CRL：SAF_GetCrlFromLdap；
o) 取证书信息：SAF_GetCertificateInfo；
p) 取证书扩展信息：SAF_GetExtTypeInfo；
q) 列举用户证书：SAF_EnumCertificates；
r) 列举用户的密钥容器信息：SAF_EnumKeyContainerInfo；
s) 释放列举用户证书的内存：SAF_EnumCertificatesFree；
t) 释放列举密钥容器信息的内存：SAF_EnumkeyContainerInfoFree。

各函数返回值应符合附录 B 中的错误代码定义。

7.2.2 添加根 CA 证书

原型：
```
int SAF_AddTrustedRootCaCertificate(
    void * hAppHandle,
    unsigned char * pucCertificate,
    unsigned int uiCertificateLen,
    unsigned int uiIndex);
```

描述：添加信任的 CA 根证书。

参数：

hAppHandle[in]	应用接口句柄
pucCertificate[in]	DER 编码的证书
uiCertificateLen[in]	证书长度
uiIndex[out]	CA 根证书索引

返回值：

0	成功
非 0	失败，返回错误代码

备注：本函数仅限于管理员使用。

7.2.3 获取根 CA 证书个数

原型：
```
int SAF_GetRootCaCertificateCount(
    void * hAppHandle,
    unsigned int * puiCount);
```

描述：获取信任根 CA 证书的个数。

参数：

hAppHandle[in]	应用接口句柄
puiCount[out]	信任根 CA 证书个数

返回值： 0 成功

非 0 失败，返回错误代码

7.2.4 获取根 CA 证书

原型：
```
int SAF_GetRootCaCertificate(
    void * hAppHandle,
    unsigned int uiIndex,
    unsigned char * pucCertificate,
    unsigned int * puiCertificateLen);
```

描述： 获取指定位置 CA 根证书。

参数： hAppHandle[in] 应用接口句柄

uiIndex[in] 根证书索引

pucCertificate[out] DER 编码的证书

puiCertificateLen[in,out] 证书长度

返回值： 0 成功

非 0 失败，返回错误代码

7.2.5 删除根 CA 证书

原型：
```
int SAF_RemoveRootCaCertificate(
    void * hAppHandle,
    unsigned int uiIndex);
```

描述： 删除指定位置 CA 根证书。

参数： hAppHandle[in] 应用接口句柄

uiIndex[in] 根证书索引

返回值： 0 成功

非 0 失败，返回错误代码

备注： 本函数仅限于管理员使用。

7.2.6 添加 CA 证书

原型：
```
int SAF_AddCaCertificate(
    void * hAppHandle,
    unsigned char * pucCertificate,
    unsigned int uiCertificateLen,
    unsigned int uiIndex);.
```

描述：　添加 CA 证书。

参数：　hAppHandle[in]　应用接口句柄

pucCertificate[in]　DER 编码的证书

uiCertificateLen[in]　证书长度

uiIndex[out]　CA 证书索引

返回值：　0　成功

非 0　失败，返回错误代码

备注：　本函数仅限于管理员使用。

7.2.7　获取 CA 证书个数

原型：

```
int SAF_GetCaCertificateCount(
    void * hAppHandle,
    unsigned int * puiCount);
```

描述：　获取信任 CA 证书的个数。

参数：　hAppHandle[in]　应用接口句柄

puiCount[out]　信任 CA 证书个数

返回值：　0　成功

非 0　失败，返回错误代码

7.2.8　获取 CA 证书

原型：

```
int SAF_GetCaCertificate(
    void * hAppHandle,
    unsigned int uiIndex,
    unsigned char * pucCertificate,
    unsigned int * puiCertificateLen);
```

描述：　获取指定位置 CA 证书。

参数：　hAppHandle[in]　应用接口句柄

uiIndex[in]　CA 证书索引

pucCertificate[out]　DER 编码的证书

puiCertificateLen[in,out]　证书长度

返回值：　0　成功

非 0　失败，返回错误代码

7.2.9 删除 CA 证书

原型：
```
int SAF_RemoveCaCertificate(
    void * hAppHandle,
    unsigned int uiIndex);
```

描述：删除指定位置 CA 证书。

参数：

hAppHandle[in]	应用接口句柄
uiIndex[in]	证书位置索引

返回值：

0	成功
非 0	失败，返回错误代码

备注：本函数仅限于管理员使用。

7.2.10 添加 CRL

原型：
```
int SAF_AddCrl(
    void * hAppHandle,
    unsigned char * pucDerCrl,
    unsigned int uiDerCrlLen);
```

描述：添加 CRL。

参数：

hAppHandle[in]	应用接口句柄
pucDerCrl[in]	DER 编码的 CRL
uiDerCrlLen[in]	DER 编码 CRL 的长度

返回值：

0	成功
非 0	失败，返回错误代码

备注：本函数仅限于管理员使用。

7.2.11 验证用户证书

原型：
```
int SAF_VerifyCertificate(
    void * hAppHandle,
    unsigned char * pucUsrCertificate,
    unsigned int uiUsrCertificateLen);
```

描述：验证用户证书的有效性，包括验证有效期、证书信任列表、吊销状态。

参数：

hAppHandle[in]	应用接口句柄
pucUsrCertificate[in]	DER 编码的证书
uiUsrCertificateLen[in]	证书长度

返回值：　0　　成功

　　非 0　　失败，返回错误代码，应符合附录 B 中的错误代码定义

7.2.12　根据 CRL 文件获取用户注销状态

原型：

```
int SAF_VerifyCertificateByCrl(
    void * hAppHandle,
    unsigned char * pucUsrCertificate,
    unsigned int uiUsrCertificateLen
    unsigned char * pucDerCrl,
    unsigned int uiDerCrlLen,
    unsigned char * pucRevokeDate);
```

描述：　根据 CRL 文件验证用户证书是否被注销。

参数：

hAppHandle[in]	应用接口句柄
pucUsrCertificate[in]	DER 编码的证书
uiUsrCertificateLen[in]	证书长度
pucDerCrl[in]	DER 编码的 CRL
uiDerCrlLen[in]	CRL 长度
pucRevokeDate[out]	若证书有效此参数不返回，若证书被注销，则返回注销日期，格式为 YYYY-MM-DD HH:MM:SS

返回值：　0　　成功

　　非 0　　失败，返回的错误代码应符合附录 B 中的错误代码定义

7.2.13　根据 OCSP 获取证书状态

原型：

```
int SAF_GetCertificateStateByOCSP(
    void * hAppHandle,
    unsigned char * pucOcspHostURL,
    unsigned int uiOcspHostURLLen,
    unsigned char * pucUsrCertificate,
    unsigned int uiUsrCertificateLen,
    unsigned char * pucCACertificate,
    unsigned int uiCACertificateLen,
    unsigned char * pucRevokeDate);
```

描述：　从 OCSP 获取用户证书的实时状态。

参数：

hAppHandle[in]	应用接口句柄
pucOcspHostURL [in]	ocsp 服务的 URL
uiOcspHostURLLen [in]	URL 长度
pucUsrCertificate[in]	DER 编码证书
uiUsrCertificateLen[in]	证书长度

	pucCACertificate[in]	DER 编码颁发者证书
	uiCACertificateLen[in]	颁发者证书长度
	pucRevokeDate[out]	若证书有效此参数不返回，若证书被吊销，则返回吊销日期，格式为 YYYY-MM-DD HH:MM:SS
返回值：	0	成功
	非 0	失败，返回错误代码，应符合附录 B 中的错误代码定义

7.2.14 通过 LDAP 方式获取证书

原型：

```
int SAF_GetCertFromLdap(
    void * hAppHandle,
    char * pcLdapHostURL,
    unsigned int uiLdapHostURLLen,
    unsigned char * pucQueryDN,
    unsigned int uiQueryDNLen,
    unsigned char * pucOutCert,
    unsigned int * puiOutCertLen);
```

描述：通过 LDAP 方式获取证书。

参数：	hAppHandle[in]	应用接口句柄
	pcLdapHostURL [in]	ldap 服务器 IP 地址
	uiLdapHostURLLen [in]	ldap 服务器端口
	pucQueryDN [in]	待查找的证书的查询条件
	uiQueryDNLen [in]	查询条件长度
	pucOutCert[out]	查询到的 base64 编码的证书，如查询出多证书，则证书信息中间以“&”分割
	puiOutCertLen[in,out]	找到的证书长度
返回值：	0	成功
	非 0	失败，返回错误代码

7.2.15 通过 LDAP 方式获取证书对应的 CRL

原型：

```
int SAF_GetCrlFromLdap (
    void * hAppHandle,
    char * pcLdapHostURL,
    unsigned int uiLdapHostURLLen,
    unsigned char * pucCertificate,
    unsigned int uiCertificateLen,
    unsigned char * pucCrlData,
    unsigned int * puiCrlDataLen);
```

描述：通过 LDAP 方式根据证书获取对应的 CRL。

参数：

hAppHandle[in]	应用接口句柄
pcLdapHostURL [in]	ldap 服务 URL
uiLdapHostURLLen [in]	URL 长度
pucCertificate [in]	DER 编码的数字证书
uiCertificateLen [in]	证书长度
pucCrlData [out]	获取的 DER 编码的 CRL 文件
puiCrlDataLen [in,out]	CRL 文件长度

返回值：

0	成功
非 0	失败，返回错误代码

7.2.16 取证书信息

原型：

```
int SAF_GetCertificateInfo(
    void * hAppHandle,
    unsigned char * pucCertificate,
    unsigned int uiCertificateLen,
    unsigned int uiInfoType,
    unsigned char * pucInfo,
    unsigned int * puiInfoLen);
```

描述：解析证书，获取证书中的信息。

参数：

hAppHandle[in]	应用接口句柄
pucCertificate[in]	DER 编码的证书
uiCertificateLen[in]	证书长度
uiInfoType[in]	指定的证书解析标识，应符合 GM/T 0006
pucInfo[out]	获取的证书信息
puiInfoLen[in,out]	获取的证书信息长度

返回值：

0	成功
非 0	失败，返回错误代码

7.2.17 取证书扩展信息

原型：

```
int SAF_GetExtTypeInfo (
    void * hAppHandle,
    unsigned char * pucDerCert,
    unsigned int uiDerCertLen,
    unsigned int uiInfoType,
    unsigned char * pucPriOid,
    unsigned int uiPriOidLen,
    unsigned char * pucInfo,
    unsigned int * puiInfoLen);
```

描述：　获取证书的扩展信息。

参数：

hAppHandle[in]	应用接口句柄
pucDerCert[in]	DER 格式的数字证书
uiDerCertLen[in]	数字证书长度
uiInfoType[in]	指定的证书扩展项解析标识，应符合 GM/T 0006
pucPriOid[in]	扩展项的 OID，如果不是私有扩展项类型，该参数无效
uiPriOidLen[in]	扩展项 OID 长度，如果不是私有扩展项类型，该参数无效
pucInfo[out]	获取的证书信息
puiInfoLen[in,out]	获取的证书信息长度

返回值：

0	成功
非 0	失败，返回错误代码

7.2.18 列举用户证书

原型：

```
int SAF_EnumCertificates(
    void * hAppHandle,
    SGD_USR_CERT_ENUMLIST * usrCerts);
```

描述：　列举出插入当前计算机的所有密码设备内的有效证书。

参数：

hAppHandle[in]	应用接口句柄
usrCerts[out]	返回的用户证书列表

返回值：

0	成功
非 0	失败，返回错误代码

备注：　usrCerts 结构内的对象数据由本函数分配空间。本函数用于客户端枚举证书。

7.2.19 列举用户的密钥容器信息

原型：

```
Int SAF_EnumKeyContainerInfo(
    void * hAppHandle,
    SGD_ KEYCONTAINERINFO_ENUMLIST * keyContainerInfo);
```

描述：　列举插入当前计算机的所有密码设备的容器信息。

参数：

hAppHandle[in]	应用接口句柄
keyContainerInfo [out]	返回的密钥容器信息

返回值：

SAR_OK	成功
其他	失败

备注：　keyContainerInfo 结构内的对象数据由本函数分配空间。仅在客户端使用。

7.2.20 释放列举用户证书的内存

原型：
```
int SAF_EnumCertificatesFree(
    void * hAppHandle,
    SGD_USR_CERT_ENUMLIST * usrCerts);
```

描述：释放 SAF_EnumCertificates 函数中分配的内存。

参数：

hAppHandle[in]	应用接口句柄
usrCerts[in]	证书信息

返回值：

0	成功
非 0	失败，返回错误代码

7.2.21 释放列举密钥容器信息的内存

原型：
```
int SAF_EnumkeyContainerInfoFree(
    void * hAppHandle,
    SGD_ KEYCONTAINERINFO_ENUMLIST * keyContainerInfo);
```

描述：释放 SAF_EnumKeyContainerInfo 函数中分配的内存。

参数：

hAppHandle[in]	应用接口句柄
keyContainerInfo[in]	密钥容器信息

返回值：

0	成功
非 0	失败，返回错误代码

7.3 密码运算类函数

7.3.1 概述

密码运算类函数通过与密码设备交互实现具体的密码运算，并将密码运算后的结果返回给应用程序，是应用程序实现数据保密性、完整性和不可抵赖性等安全机制的基础，包括以下具体函数：

a) 单块 base64 编码：SAF_Base64_Encode；
b) 单块 base64 解码：SAF_Base64_Decode；
c) 创建 base64 对象：SAF_Base64_CreateBase64Obj；
d) 销毁 base64 对象：SAF_Base64_DestroyBase64Obj；
e) 通过 base64 对象继续编码：SAF_Base64_EncodeUpdate；
f) 通过 base64 对象编码结束：SAF_Base64_EncodeFinal；
g) 通过 base64 对象继续解码：SAF_Base64_DecodeUpdate；
h) 通过 base64 对象解码结束：SAF_Base64_DecodeFinal；
i) 生成随机数：SAF_GenRandom；
j) HASH 运算：SAF_Hash；
k) 创建 HASH 对象：SAF_CreateHashObj；
l) 删除 HASH 对象：SAF_DestroyHashObj；
m) 通过对象进行多块 HASH 运算：SAF_HashUpdate；

n) 结束 HASH 运算:SAF_HashFinal;
o) 生成 RSA 密钥对:SAF_GenRsaKeyPair;
p) 获取 RSA 公钥:SAF_GetRsaPublicKey;
q) RSA 签名运算:SAF_RsaSign;
r) 对文件进行 RSA 签名运算:SAF_RsaSignFile;
s) RSA 验证签名运算:SAF_RsaVerifySign;
t) 对文件及其签名进行 RSA 验证:SAF_RsaVerifySignFile;
u) 基于证书的 RSA 公钥验证:SAF_VerifySignByCert;
v) 生成 SM2 密钥对:SAF_GenSm2KeyPair;
w) 获取 SM2 公钥:SAF_GetSm2PublicKey;
x) SM2 签名:SAF_Sm2Sign;
y) SM2 验证:SAF_Sm2VerifySign;
z) SM2 公钥加密:SAF_Sm2PublicKeyEnc;
aa) 基于证书的 SM2 公钥加密:SAF_Sm2PublicKeyEncByCert;
bb) 基于证书的 SM2 解密:SAF_Sm2DecByCert;
cc) 基于证书的 SM2 公钥验证:SAF_Sm2VerifySignByCert;
dd) 创建对称算法对象:SAF_CreateSymmAlgoObj;
ee) 生成会话密钥并用外部公钥加密输出:SAF_GenerateKeyWithEPK;
ff) 导入加密的会话密钥:SAF_ImportEncedKey;
gg) 生成密钥协商参数并输出:SAF_GenerateAgreementDataWithSM2;
hh) 计算会话密钥:SAF_GenerateKeyWithSM2;
ii) 产生协商数据并计算会话密钥:SAF_GenerateAgreementDataAndKeyWithSM2;
jj) 销毁对称算法对象:SAF_DestroySymmAlgoObj;
kk) 销毁会话密钥句柄:SAF_DestroyKeyHandle;
ll) 单块加密运算:SAF_SymmEncrypt;
mm) 多块加密运算:SAF_SymmEncryptUpdate;
nn) 结束加密运算:SAF_SymmEncryptFinal;
oo) 单块解密运算:SAF_SymmDecrypt;
pp) 多块解密运算:SAF_SymmDecryptUpdate;
qq) 结束解密运算:SAF_SymmDecryptFinal;
rr) 单组数据消息鉴别码运算:SAF_Mac;
ss) 多组数据消息鉴别码运算:SAF_MacUpdate;
tt) 结束消息鉴别码运算:SAF_MacFinal。

各函数返回值应符合附录 B 中的错误代码定义。

7.3.2 单块 base64 编码

原型:

```
int SAF_Base64_Encode(
    unsigned char * pucInData,
    unsigned int uiInDataLen,
    unsigned char * pucOutData,
    unsigned int * puiOutDataLen);
```

描述：　对输入的数据进行 base64 编码，编码格式符合 IETF RFC 4648。

pucInData[in]	编码前的数据
uiInDataLen[in]	编码前的数据长度
pucOutData[out]	编码后的数据
puiOutDataLen[in,out]	编码后的数据长度

返回值：

0	成功
非 0	失败，返回错误代码

7.3.3 单块 base64 解码

原型：

```
int SAF_Base64_Decode(
    unsigned char * pucInData,
    unsigned int uiInDataLen,
    unsigned char * pucOutData,
    unsigned int * puiOutDataLen);
```

描述：　对输入的数据进行 base64 解码，编码格式符合 IETF RFC 4648。

pucInData[in]	解码前的数据
uiInDataLen[in]	解码前的数据长度
pucOutData[out]	解码后的数据
puiOutDataLen[in,out]	解码后的数据长度

返回值：

0	成功
非 0	失败，返回错误代码

7.3.4 创建 base64 对象

原型：

```
int SAF_Base64_CreateBase64Obj(
    void * hAppHandle,
    void ** phBase64Obj);
```

描述：　为任意长度数据的 base64 编解码创建 base64 对象，编码格式符合 IETF RFC 4648。

参数：

hAppHandle[in]	应用句柄
phBase64Obj[out]	指向创建的 base64 对象句柄的指针

返回值：

0	成功
非 0	失败，返回错误代码

备注：　本函数与 SAF_Base64_DestroyBase64Obj，SAF_Base64_EncodeUpdate，SGD_Base64_EncodeFinal，SAF_Base64_DecodeUpdate，SAF_Base64_DecodeFinal 函数共同使用以支持任意长度数据的 base64 编解码。

7.3.5 销毁 base64 对象

原型：
```
int SAF_Base64_DestroyBase64Obj(
    void * hAppHandle,
    void * hBase64Obj);
```

描述：删除 base64 对象。

参数：

hAppHandle [in]	应用句柄
hBase64Obj[in]	待删除的 base64 对象句柄

返回值：

0	成功
非 0	失败，返回错误代码

7.3.6 通过 base64 对象继续编码

原型：
```
int SAF_Base64_EncodeUpdate(
    void * hBase64Obj,
    unsigned char * pucInData,
    unsigned int   uiInDataLen,
    unsigned char * pucOutData,
    unsigned int * puiOutDataLen);
```

描述：通过 base64 对象对数据多块 base64 编码，编码格式符合 IETF RFC 4648。

参数：

hBase64Obj[in]	base64 对象
pucInData[in]	编码前的数据
uiInDataLen[in]	编码前的数据长度
pucOutData[out]	编码后的数据
puiOutDataLen[in,out]	返回编码后的数据长度

返回值：

0	成功
非 0	失败，返回错误代码

备注：编码完成后，还应调用 SAF_Base64_EncodeFinal

7.3.7 通过 base64 对象编码结束

原型：
```
int SAF_Base64_EncodeFinal(
    void * hBase64Obj,
    unsigned char * pucOutData,
    unsigned int * puiOutDataLen);
```

描述：通过 base64 对象对数据编码结束，编码格式符合 IETF RFC 4648。

参数：

hBase64Obj[in]	base64 对象

	pucOutData[out]	编码后的数据
	puiOutDataLen[in,out]	返回编码后的数据长度
返回值：	0	成功
	非 0	失败，返回错误代码

7.3.8 通过 base64 对象继续解码

原型：

```
int SAF_Base64_DecodeUpdate(
    void * hBase64Obj,
    unsigned char * pucInData,
    unsigned int    uiInDataLen,
    unsigned char * pucOutData,
    unsigned int * puiOutDataLen);
```

描述：通过 base64 对象对数据多块 base64 解码，编码格式符合 IETF RFC 4648。

参数：	hBase64Obj[in]	base64 对象
	pucInData[in]	解码前的数据
	uiInDataLen[in]	解码前的数据长度
	pucOutData[out]	解码后的数据
	puiOutDataLen[in,out]	返回解码后的数据长度
返回值：	0	成功
	非 0	失败，返回错误代码

备注：解码完成后，还应调用 SAF_Base64_DecodeFinal 结束

7.3.9 通过 base64 对象解码结束

原型：

```
int SAF_Base64_DecodeFinal(
    void * hBase64Obj,
    unsigned char * pucOutData,
    unsigned int * puiOutDataLen);
```

描述：通过 base64 对象对数据解码结束，编码格式符合 IETF RFC 4648。

参数：	hBase64Obj[in]	base64 对象
	pucOutData[out]	解码后的数据
	puiOutDataLen[in,out]	返回解码后的数据长度
返回值：	0	成功
	非 0	失败，返回错误代码

7.3.10 生成随机数

原型：

```
int SAF_GenRandom(
    void * hAppHandle,
    unsigned int uiRandLen,
    unsigned char * pucRand);
```

描述：生成指定长度的随机数。

hAppHandle[in]	应用句柄
uiRandLen[in]	随机数长度
pucRand[out]	指定长度的随机数，内存由外部调用者分配

返回值：

0	成功
非 0	失败，返回错误代码

7.3.11 HASH 运算

原型：

```
int SAF_Hash(
    void * hAppHandle,
    unsigned int uiAlgoType,
    unsigned char * pucInData,
    unsigned int uiInDataLen,
    unsigned char * pucPublicKey,
    unsigned int uiPublicKeyLen,
    unsigned char * pucID,
    unsigned int uiIDLen,
    unsigned char * pucOutData,
    unsigned int * puiOutDataLen);
```

描述：HASH 运算，对给定长度数据的 HASH 运算。

参数：

hAppHandle[in]	应用句柄
uiAlgoType[in]	HASH 算法
pucInData[in]	输入数据
uiInDataLen[in]	输入数据长度
pucPublicKey[in]	签名者公钥，当 uiAlgoType 为 SGD_SM3 时有效
uiPublicKeyLen[in]	签名者公钥长度
pucID[in]	签名者的 ID 值，当 uiAlgoType 为 SGD_SM3 时有效
uiIDLen[in]	签名者 ID 的长度，当 uiAlgoType 为 SGD_SM3 时有效
pucOutData[out]	HASH
puiOutDataLen[out]	HASH 长度

返回值：　0　　成功

　　非 0　　失败，返回错误代码

备注：　当 uiAlgoType 为 SGD_SM3 且 uiIDLen 不为 0 的情况下 pucPublicKey、pucID 有效，执行 SM2 算法签名预处理 1 操作。计算过程应符合 GM/T 0009。

7.3.12 创建 HASH 对象

原型：
```
int SAF_CreateHashObj(
    void * hAppHandle,
    void * * phHashObj,
    unsigned int uiAlgorithmType
    unsigned char * pucPublicKey,
    unsigned int uiPublicKeyLen,
    unsigned char * pucID
    unsigned int uiIDLen);
```

描述：　创建 HASH 对象，对任意长度数据的 HASH 运算。

参数：　hAppHandle [in]　　应用句柄

　　phHashObj[out]　　创建的 HASH 对象

　　uiAlgorithmType[in]　　HASH 算法

　　pucPublicKey[in]　　签名者公钥，当 uiAlgoType 为 SGD_SM3 时有效

　　uiPublicKeyLen[in]　　签名者公钥长度

　　pucID[in]　　签名者的 ID 值，当 uiAlgoType 为 SGD_SM3 时有效

　　uiIDLen[in]　　签名者 ID 的长度，当 uiAlgoType 为 SGD_SM3 时有效

返回值：　0　　成功

　　非 0　　失败，返回错误代码

备注：　当 uiAlgoType 为 SGD_SM3 且 uiIDLen 不为 0 的情况下 pucPublicKey、pucID 有效，执行 SM2 算法签名预处理 1 操作。计算过程应符合 GM/T 0009。

7.3.13 删除 HASH 对象

原型：
```
int SAF_DestroyHashObj(
    void * hAppHandle,
    void * hHashObj);
```

描述：　删除 HASH 对象。

参数：　hAppHandle [in]　　应用句柄

　　hHashObj[in]　　待删除的 HASH 对象

返回值：　0　　成功

　　非 0　　失败，返回错误代码

7.3.14 通过对象进行多块 HASH 运算

原型：
```
int SAF_HashUpdate(
    void * hHashObj,
    unsigned char * pucInData,
    unsigned int uiInDataLen);
```

描述：　继续 HASH 运算。

参数：　hHashObj[in]　　HASH 对象

　　pucInData[in]　　输入数据

　　uiInDataLen[in]　　输入数据长度

返回值：　0　　成功

　　非 0　　失败，返回错误代码

备注：　运算完成后，还应调用 SAF_HashFinal 结束。

7.3.15 结束 HASH 运算

原型：
```
int SAF_HashFinal(
    void * hHashObj,
    unsigned char * pucOutData,
    unsigned int * puiOutDataLen);
```

描述：　结束 HASH 运算。

参数：　hHashObj[in]　　HASH 对象

　　pucOutData[out]　　输出的 HASH 值

　　puiOutDataLen[out]　　HASH 值的长度

返回值：　0　　成功

　　非 0　　失败，返回错误代码

7.3.16 生成 RSA 密钥对

原型：
```
int SAF_GenRsaKeyPair (
    void * hAppHandle,
    unsigned char * pucContainerName,
    unsigned int uiContainerNameLen,
    unsigned int uiKeyBits,
    unsigned int uiKeyUsage,
    unsigned int uiExportFlag);
```

描述：　产生指定名称的容器并在该容器内生成 RSA 密钥对。

参数：

hAppHandle[in]	应用接口句柄
pucContainerName[in]	密钥的容器名
uiContainerNameLen[in]	密钥的容器名长度
uiKeyBits[in]	密钥模长
uiKeyUsage[in]	密钥用途：
SGD_KEYUSAGE_SIGN	表示签名
SGD_KEYUSAGE_KEYEXCHANGE	表示密钥交换(加密)
uiExportFlag[in]	1 表示生成的密钥对可导出 0 表示不可导出

返回值：

0	成功
非 0	失败，返回错误代码

备注：　如果指定的容器名已存在，则在该容器内增加或者替换密钥对。

7.3.17　获取 RSA 公钥

原型：

```
int SAF_GetRsaPublicKey(
    void * hAppHandle,
    unsigned char * pucContainerName,
    unsigned int uiContainerLen,
    unsigned int uiKeyUsage,
    unsigned char * pucPublicKey,
    unsigned int * puiPublicKeyLen);
```

描述：　取出符合 PKCS #1 的 RSA 公钥。

参数：

hAppHandle[in]	应用接口句柄
pucContainerName[in]	密钥的容器名或密码机的密码号
uiContainerLen[in]	密钥的容器名长度
uiKeyUsage[in]	密钥用途
SGD_KEYUSAGE_SIGN	表示签名
SGD_KEYUSAGE_KEYEXCHANGE	表示加密
pucPublicKey[out]	输出的 DER 格式的公钥数据
puiPublicKeyLen[in,out]	输出公钥数据的长度

返回值：

0	成功
非 0	失败，返回错误代码

7.3.18 RSA 签名运算

原型：

```
int SAF_RsaSign(
    void * hAppHandle,
    unsigned char * pucContainerName,
    unsigned int uiContainerNameLen,
    unsigned int uiHashAlgorithmID,
    unsigned char * pucInData,,
    unsigned int uiInDataLen,
    unsigned char * pucSignData,
    unsigned int * puiSignDataLen);
```

描述：按照 PKCS #1 的要求对一定长度的字符串进行签名运算。

参数：

hAppHandle[in]	应用接口句柄
pucContainerName[in]	密钥的容器名或密钥号
uiContainerNameLen[in]	密钥的容器名长度
uiHashAlgorithmID[in]	HASH 算法
pucInData[in]	签名原文
uiInDataLen[in]	签名原文的长度
pucSignData[out]	输出签名结果数据
puiSignDataLen[in,out]	输出的签名结果数据长度

返回值：

0	成功
非 0	失败，返回错误代码

7.3.19 对文件进行 RSA 签名运算

原型：

```
int SAF_RsaSignFile(
    void * hAppHandle,
    unsigned char * pucContainerName,
    unsigned int uiContainerNameLen,
    unsigned int uiHashAlgorithmID,
    unsigned char * pucFileName,
    unsigned char * pucSignData,
    unsigned int * puiSignDataLen);
```

描述：按照 PKCS #1 的要求对指定的文件进行签名运算。

参数：

hAppHandle[in]	应用接口句柄
pucContainerName[in]	密钥的容器名
uiContainerNameLen[in]	密钥的容器名长度

	uiHashAlgorithmID[in]	HASH 算法
	pucFileName[in]	要签名的全路径文件名
	pucSignData[out]	输出的 DER 格式的签名结果数据
	puiSignDataLen[in,out]	输出的签名结果数据长度
返回值：	0	成功
	非 0	失败，返回错误代码

备注：　此函数主要用于对附件的签名，也适用于对大文件的签名。

7.3.20 RSA 验证签名运算

原型：

```
int SAF_RsaVerifySign(
    void * hAppHandle,
    unsigned int uiHashAlgorithmID,
    unsigned char * pucPublicKey,
    unsigned int uiPublicKeyLen,
    unsigned char * pucInData,
    unsigned int uiInDataLen,
    unsigned char * pucSignData,
    unsigned int uiSignDataLen);
```

描述：　符合 PKCS #1 的验证签名运算。

参数：	hAppHandle [in]	应用句柄
	uiHashAlgorithmID[in]	HASH 算法
	pucPublicKey[in]	DER 编码的公钥
	uiPublicKeyLen[in]	DER 编码的公钥长度
	pucInData[in]	签名原文
	uiInDataLen[in]	签名原文的长度
	pucSignData[in]	DER 编码的签名数据
	uiSignDataLen[in]	签名数据长度
返回值：	0	成功
	非 0	失败，返回错误代码

7.3.21 对文件及其签名进行 RSA 验证

原型：
```
int SAF_RsaVerifySignFile(
    void * hAppHandle,
    unsigned int uiHashAlgorithmID,
    unsigned char * pucPublicKey,
    unsigned int uiPublicKeyLen,
    unsigned char * pucFileName,
    unsigned char * pucSignData,
    unsigned int uiSignDataLen);
```

描述：对文件及其签名值，进行符合 PKCS #1 的验证签名运算。

参数：

hAppHandle [in]	应用句柄
uiHashAlgorithmID[in]	HASH 算法
pucPublicKey[in]	DER 编码的公钥
uiPublicKeyLen[in]	DER 编码的公钥长度
pucFileName[in]	待验证签名的文件名
pucSignData[in]	DER 编码的签名数据
uiSignDataLen[in]	签名数据长度

返回值：

0	成功
非 0	失败，返回错误代码

备注：本函数用于对附件的签名进行验证，也可对大文件的签名进行验证。

7.3.22 基于证书的 RSA 公钥验证

原型：
```
int SAF_VerifySignByCert (
    void * hAppHandle,
    unsigned int uiHashAlgorithmID,
    unsigned char * pucCertificate,
    unsigned int  uiCertificateLen,
    unsigned char * pucInData,
    unsigned int uiInDataLen,
    unsigned char * pucSignData,
    unsigned int uiSignDataLen);
```

描述：按照 PKCS #1 的要求使用数字证书对签名值进行验证。

参数：

hAppHandle [in]	应用句柄
uiHashAlgorithmID[in]	HASH 算法标识

	pucCertificate[in]	DER 编码的数字证书
	uiCertificateLen[in]	DER 编码的数字证书长度
	pucInData[in]	签名原文
	uiInDataLen[in]	签名原文的长度
	pucSignData[in]	签名数据
	uiSignDataLen[in]	签名数据长度
返回值：	0	成功
	非 0	失败，返回错误代码

7.3.23 生成 SM2 密钥对

原型：

```
int SAF_GenSm2KeyPair(
    void * hAppHandle,
    unsigned char * pucContainerName,
    unsigned int uiContainerLen,
    unsigned int uiKeyBits,
    unsigned int uiKeyUsage,
    unsigned int uiExportFlag);
```

描述：生成指定名称的容器，并在该容器内生成 SM2 密钥对。

参数：	hAppHandle[in]	应用接口句柄
	pucContainerName[in]	密钥的容器名
	uiContainerLen[in]	密钥的容器名长度
	uiKeyBits[in]	密钥模长
	uiKeyUsage[in]	密钥用途
	SGD_SM2_1	表示椭圆曲线签名
	SGD_SM2_2	表示椭圆曲线密钥交换协议
	SGD_SM2_3	表示椭圆曲线加密
	uiExportFlag[in]	1 表示生成的密钥对可导出 0 表示不可导出
返回值：	0	成功
	非 0	失败，返回错误代码

7.3.24 获取 SM2 公钥

原型：

```
int SAF_GetSm2PublicKey(
    void * hAppHandle,
    unsigned char * pucContainerName,
    unsigned int uiContainerLen,
    unsigned int uiKeyUsage,
    unsigned char * pucPublicKey,
    unsigned int * puiPublicKeyLen);
```

描述：取出 SM2 公钥。

参数：

hAppHandle[in]	应用接口句柄
pucContainerName[in]	密钥的容器名
uiContainerLen[in]	密钥的容器名长度
uiKeyUsage[in]	密钥用途
SGD_SM2_1	表示签名公钥
SGD_SM2_3	表示加密公钥
pucPublicKey[out]	输出的 DER 编码的公钥数据
puiPublicKeyLen[in,out]	输出公钥数据的长度

返回值：

0	成功
非 0	失败，返回错误代码

7.3.25 SM2 签名

原型：

```
int SAF_Sm2Sign (
    void * hAppHandle,
    unsigned char * pucContainerName,
    unsigned int uiContainerNameLen,
    unsigned int uiAlgorithmID,
    unsigned char * pucInData,
    unsigned int uiInDataLen,
    unsigned char * pucSignData,
    unsigned int * puiSignDataLen);
```

描述：使用 SM2 私钥对数据进行签名运算。

参数：

hAppHandle[in]	应用接口句柄
pucContainerName[in]	密钥的容器名
uiContainerNameLen[in]	密钥的容器名长度

	uiAlgorithmID[in]	签名算法标识
	pucInData[in]	待签名数据
	uiInDataLen[in]	待签名数据的长度
	pucSignData[out]	输出的 DER 编码的签名数据
	puiSignDataLen[in,out]	输出的签名结果数据长度
返回值：	0	成功
	非 0	失败，返回错误代码

7.3.26 SM2 验证

原型：

```
int SAF_ Sm2VerifySign(
    void * hAppHandle,
    unsigned char * pucPublicKey,
    unsigned int uiPublicKeyLen,
    unsigned int uiAlgorithmID,
    unsigned char * pucInData,
    unsigned int uiInDataLen,
    unsigned char * pucSignData,
    unsigned int uiSignDataLen);
```

描述：利用 SM2 公钥验证签名。

参数：	hAppHandle [in]	应用句柄
	pucPublicKey[in]	DER 编码的公钥
	uiPublicKeyLen[in]	公钥长度
	uiAlgorithmID[in]	SM2 签名算法
	pucInData[in]	待验证数据
	uiInDataLen[in]	待验证数据的长度
	pucSignData[in]	DER 编码的签名数据
	uiSignDataLen[in]	签名数据长度
返回值：	0	成功
	非 0	失败，返回错误代码

7.3.27 SM2 公钥加密

原型：
```
int SAF_Sm2PublicKeyEnc(
    void * hAppHandle,
    unsigned char * pucPublicKey,
    unsigned int uiPublicKeyLen,
    unsigned int uiAlgorithmID,
    unsigned char * pucInData,
    unsigned int uiInDataLen,
    unsigned char * pucOutData,
    unsigned int * puiOutDataLen);
```

描述：SM2 公钥加密运算。

参数：

hAppHandle [in]	应用句柄
pucPublicKey[in]	DER 编码的公钥
uiPublicKeyLen[in]	DER 编码的公钥长度
uiAlgorithmID[in]	SM2 算法标识
pucInData[in]	输入数据，该数据不能为密钥
uiInDataLen[in]	输入数据长度
pucOutData[out]	输出数据，DER 编码
puiOutDataLen[in,out]	输出数据长度

返回值：

0	成功
非 0	失败，返回错误代码

7.3.28 基于证书的 SM2 公钥加密

原型：
```
int SAF_Sm2PublicKeyEncByCert(
    void * hAppHandle,
    unsigned char * pucCertificate,
    unsigned int uiCertificateLen,
    unsigned int uiAlgorithmID,
    unsigned char * pucInData,
    unsigned int uiInDataLen,
    unsigned char * pucOutData,
    unsigned int * puiOutDataLen);
```

描述：基于证书的 SM2 公钥加密。

参数：

hAppHandle [in]	应用句柄
pucCertificate[in]	DER 编码的数字证书

	uiCertificateLen[in]	数字证书长度
	uiAlgorithmID[in]	SM2 算法标识
	pucInData[in]	输入数据
	uiInDataLen[in]	输入数据长度
	pucOutData[out]	输出 DER 编码的数据
	puiOutDataLen[in,out]	输出数据长度
返回值：	0	成功
	非 0	失败，返回错误代码

7.3.29 基于证书的 SM2 解密

原型：

```
int SAF_Sm2DecByCert(
    void * hAppHandle,
    int certIDType,
    unsigned char * certID,
    unsigned int certIDLen,
    unsigned int uiAlgorithmID,
    unsigned char * pucInData,
    unsigned int uiInDataLen,
    unsigned char * pucOutData,
    unsigned int * puiOutDataLen);
```

描述：基于证书的 SM2 解密。

参数：	hAppHandle [in]	应用句柄
	certIDType [in]	证书标识参数的具体含义，取值应符合 GM/T 0016 中证书解析标识的定义
	certID [in]	证书标识
	certIDLen [in]	证书标识长度
	uiAlgorithmID[in]	SM2 算法标识
	pucInData[in]	输入 DER 编码密文，应符合 GM/T 0009 中加密数据格式的定义
	uiInDataLen[in]	输入数据长度
	pucOutData[out]	输出明文
	puiOutDataLen[in,out]	输出数据长度
返回值：	0	成功
	非 0	失败，返回错误代码

7.3.30 基于证书的 SM2 公钥验证

原型：

```
int SAF_Sm2VerifySignByCert(
    void * hAppHandle,
    unsigned int uiAlgorithmID,
    unsigned char * pucCertificate,
    unsigned int uiCertificateLen,
    unsigned char * pucInData,
    unsigned int uiInDataLen,
    unsigned char * pucSignData,
    unsigned int uiSignDataLen);
```

描述：基于证书的 SM2 公钥验证。

参数：

hAppHandle [in]	应用句柄
uiAlgorithmID[in]	SM2 签名算法标识
pucCertificate[in]	DER 编码的数字证书
uiCertificateLen[in]	数字证书长度
pucInData[in]	待验证数据
uiInDataLen[in]	待验证数据的长度
pucSignData[in]	DER 编码的签名数据
uiSignDataLen[in]	签名数据长度

返回值：

0	成功
非 0	失败，返回错误代码

7.3.31 创建对称算法对象

原型：

```
int SAF_CreateSymmAlgoObj(
    void * hAppHandle,
    void * * phSymmKeyObj,
    unsigned char * pucContainerName,
    unsigned int uiContainerLen,
    unsigned char * pucIV,
    unsigned int uiIVLen,
    unsigned int uiEncOrDec,
    unsigned int uiCryptoAlgID);
```

描述：本地产生对称算法对象。

参数：

hAppHandle[in]	应用接口句柄
phSymmKeyObj[out]	返回的对称算法对象

	pucContainerName[in]	密钥的容器名
	uiContainerLen[in]	密钥的容器名长度
	pucIV[in]	初始向量，ECB 模式时此参数忽略
	uiIVLen[in]	初始向量长度，ECB 模式时此参数忽略
	uiEncOrDec[in]	1 加密 0 解密
	uiCryptoAlgID[in]	加密算法标识
返回值：	0	成功
	非 0	失败，返回错误代码

7.3.32 生成会话密钥并用外部公钥加密输出

原型：

```
int SAF_GenerateKeyWithEPK(
    void * hSymmKeyObj,
    unsigned char * pucPublicKey,
    unsigned int uiPublicKeyLen,
    unsigned char * pucSymmKey,
    unsigned int uiSymmKeyLen,
    void * * phKeyHandle);
```

描述：生成会话密钥并用外部公钥加密输出。

参数：	hSymmKeyObj[in]	对称算法对象
	pucPublicKey[in]	输入的 DER 编码的公钥数据
	uiPublicKeyLen[in]	输入公钥数据的长度
	pucSymmKey[out]	输出的加密后的会话密钥
	uiSymmKeyLen[out]	输出的加密会话密钥长度
	phKeyHandle[out]	输出会话密钥的句柄
返回值：	0	成功
	非 0	失败，返回错误代码

7.3.33 导入加密的会话密钥

原型：

```
int SAF_ImportEncedKey(
    void * hSymmKeyObj,
    unsigned char * pucSymmKey,
    unsigned int uiSymmKeyLen,
    void * * phKeyHandle);
```

描述：　导入加密的会话密钥，使用指定的私钥解密，产生会话密钥句柄并输出。

参数：

hSymmKeyObj[in]	对称算法对象
pucSymmKey[in]	输入的加密后的会话密钥
uiSymmKeyLen[in]	输入的加密会话密钥长度
phKeyHandle[out]	输出会话密钥的句柄

返回值：

0	成功
非 0	失败，返回错误代码

7.3.34 生成密钥协商参数并输出

原型：

```
int SAF_GenerateAgreementDataWithSM2 (
        void * hSymmKeyObj,
        unsigned int uiISKIndex,
        unsigned int uiKeyBits,
        unsigned char * pucSponsorID,
        unsigned int uiSponsorIDLength,
        unsigned char * pucSponsorPublicKey,
        unsigned int * puiSponsorPublicKeyLen,
        unsigned char * pucSponsorTmpPublicKey,
        unsigned int * puiSponsorTmpPublicKeyLen,
        void * * phAgreementHandle);
```

描述：　使用 SM2 密钥协商算法，为计算会话密钥而产生协商参数，同时返回指定索引位置的 SM2 公钥、临时 SM2 密钥对的公钥及协商句柄。

参数：

hSymmKeyObj [in]	对称算法对象
uiISKIndex[in]	密码设备内部存储加密私钥的索引值，该私钥用于参与密钥协商
uiKeyBits[in]	要求协商的密钥长度
pucSponsorID[in]	参与密钥协商的发起方 ID 值
uiSponsorIDLength[in]	发起方 ID 长度
pucSponsorPublicKey[out]	返回的发起方 SM2 公钥
puiSponsorPublicKeyLen[in,out]	返回的发起方 SM2 公钥长度
pucSponsorTmpPublicKey[out]	返回的发起方临时 SM2 公钥
puiSponsorTmpPublicKeyLen [in,out]	返回的发起方临时 SM2 公钥长度
phAgreementHandle[out]	返回的密钥协商句柄

返回值：

0	成功
非 0	失败，返回错误代码

备注：　为协商会话密钥，协商的发起方应首先调用本函数计算协商参数。协商参数的计算过程应符合 GM/T 0009。

7.3.35 计算会话密钥

原型：

```
int SAF_GenerateKeyWithSM2(
    void * hAgreementHandle,
    unsigned char * pucResponseID,
    unsigned int uiResponseIDLength,
    unsigned char * pucResponsePublicKey,
    unsigned int uiResponsePublicKeyLen,
    unsigned char * pucResponseTmpPublicKey,
    unsigned int uiResponseTmpPublicKeyLen,
    void * * phKeyHandle);
```

描述：使用 SM2 密钥协商算法，使用自身协商句柄和响应方的协商参数计算会话密钥，同时返回会话密钥句柄。

参数：

hAgreementHandle [in]	密钥协商句柄
pucResponseID[in]	外部输入的响应方 ID 值
uiResponseIDLength[in]	外部输入的响应方 ID 长度
pucResponsePublicKey[in]	外部输入的响应方 SM2 公钥
uiResponsePublicKeyLen[in]	外部输入的响应方 SM2 公钥长度
pucResponseTmpPublicKey[in]	外部输入的响应方临时 SM2 公钥
uiResponseTmpPublicKeyLen[in]	外部输入的响应方临时 SM2 公钥长度
phKeyHandle[out]	返回的密钥句柄

返回值：

0	成功
非 0	失败，返回错误代码

备注：协商的发起方获得响应方的协商参数后调用本函数，计算会话密钥。会话密钥的计算过程应符合 GM/T 0009。

7.3.36 产生协商数据并计算会话密钥

原型：

```
int SAF_GenerateAgreementDataAndKeyWithSM2 (
    void * hSymmKeyObj,
    unsigned int uiISKIndex,
    unsigned int uiKeyBits,
    unsigned char * pucResponseID,
    unsigned int uiResponseIDLength,
    unsigned char * pucSponsorID,
    unsigned int uiSponsorIDLength,
    unsigned char * pucSponsorPublicKey,
    unsigned int uiSponsorPublicKeyLen,
```

```
    unsigned char * pucSponsorTmpPublicKey,
    unsigned int uiSponsorTmpPublicKeyLen,
    unsigned char * pucResponsePublicKey,
    unsigned int * puiResponsePublicKeyLen,
    unsigned char * pucResponseTmpPublicKey,
    unsigned int * puiResponseTmpPublicKeyLen,
    void * * phKeyHandle);
```

描述：　使用 SM2 密钥协商算法，产生协商参数并计算会话密钥，同时返回产生的协商参数和密钥句柄。

参数：

hSymmKeyObj[in]	对称算法对象
uiISKIndex[in]	密码设备内部存储加密私钥的索引值，该私钥用于参与密钥协商
uiKeyBits[in]	协商后要求输出的密钥长度
pucResponseID[in]	响应方 ID 值
uiResponseIDLength[in]	响应方 ID 长度
pucSponsorID[in]	发起方 ID 值
uiSponsorIDLength[in]	发起方 ID 长度
pucSponsorPublicKey[in]	外部输入的发起方 SM2 公钥
pucSponsorPublicKeyLen[in]	外部输入的发起方 SM2 公钥长度
uiSponsorTmpPublicKey[in]	外部输入的发起方临时 SM2 公钥
uiSponsorTmpPublicKeyLen[in]	外部输入的发起方临时 SM2 公钥长度
pucResponsePublicKey[out]	返回的响应方 SM2 公钥
puiResponsePublicKeyLen[in,out]	返回的响应方 SM2 公钥长度
pucResponseTmpPublicKey[out]	返回的响应方临时 SM2 公钥
puiResponseTmpPublicKeyLen[in,out]	返回的响应方临时 SM2 公钥长度
phKeyHandle[out]	返回的对称算法密钥句柄

返回值：

0	成功
非 0	失败，返回错误代码

备注：　本函数由响应方调用。会话密钥的计算过程应符合 GM/T 0009。

7.3.37 销毁对称算法对象

原型：

```
int SAF_DestroySymmAlgoObj(
    void * hSymmKeyObj);
```

描述：　销毁对称算法对象

参数： hSymmKeyObj[in] 对称算法密钥句柄

返回值： 0 成功

非 0 失败，返回错误代码

7.3.38 销毁会话密钥句柄

原型：
```
int SAF_DestroyKeyHandle(
    void * hKeyHandle);
```

描述： 销毁会话密钥句柄。

参数： hKeyHandle [in] 会话密钥句柄

返回值： 0 成功

非 0 失败，返回错误代码

7.3.39 单块加密运算

原型：
```
int SAF_SymmEncrypt(
    void * hKeyHandle,
    unsigned char * pucInData,
    unsigned int uiInDataLen,
    unsigned char * pucOutData,
    unsigned int * puiOutDataLen);
```

描述： 加密运算。

参数： hKeyHandle[in] 对称算法密钥句柄

pucInData[in] 输入数据

uiInDataLen[in] 输入数据长度

pucOutData[out] 输出数据

puiOutDataLen[in,out] 输出数据长度

返回值： 0 成功

非 0 失败，返回错误代码

7.3.40 多块加密运算

原型：
```
int SAF_SymmEncryptUpdate(
    void * hKeyHandle,
    unsigned char * pucInData,
    unsigned int uiInDataLen,
    unsigned char * pucOutData,
    unsigned int * puiOutDataLen);
```

描述：　多块加密运算

参数：　hKeyHandle[in]　对称算法密钥句柄

pucInData[in]　输入数据

uiInDataLen[in]　输入数据长度

pucOutData[out]　输出数据

puiOutDataLen[in,out]　输出数据长度

返回值：　0　成功

非 0　失败，返回错误代码

备注：　运算完成后，还应调用 SAF_SymmEncryptFinal 结束。

7.3.41 结束加密运算

原型：

```
int SAF_SymmEncryptFinal(
    void * hKeyHandle,
    unsigned char * pucOutData,
    unsigned int * puiOutDataLen);
```

描述：　结束加密运算

参数：　hKeyHandle[in]　对称算法密钥句柄

pucOutData[out]　输出数据

puiOutDataLen[in,out]　输出数据长度

返回值：　0　成功

非 0　失败，返回错误代码

7.3.42 单块解密运算

原型：

```
int SAF_SymmDecrypt(
    void * hKeyHandle,
    unsigned char * pucInData,
    unsigned int uiInDataLen,
    unsigned char * pucOutData,
    unsigned int * puiOutDataLen);
```

描述：　解密运算。

参数：　hKeyHandle[in]　对称算法密钥句柄

pucInData[in]　输入数据

uiInDataLen[in]　输入数据长度

pucOutData[out]　输出数据

	puiOutDataLen[in,out]	输出数据长度
返回值：	0	成功
	非 0	失败，返回错误代码

7.3.43 多块解密运算

原型：

```
int SAF_SymmDecryptUpdate(
    void * hKeyHandle,
    unsigned char * pucInData,
    unsigned int uiInDataLen,
    unsigned char * pucOutData,
    unsigned int * puiOutDataLen);
```

描述：继续解密运算。

参数：	hKeyHandle[in]	对称算法密钥句柄
	pucInData[in]	输入数据
	uiInDataLen[in]	输入数据长度
	pucOutData[out]	输出数据
	puiOutDataLen[in,out]	输出数据长度
返回值：	0	成功
	非 0	失败，返回错误代码

备注：运算完成后，还应调用 SAF_SymmDecryptFinal 结束。

7.3.44 结束解密运算

原型：

```
int SAF_SymmDecryptFinal(
    void * hKeyHandle,
    unsigned char * pucOutData,
    unsigned int * puiOutDataLen);
```

描述：结束解密运算。

参数：	hKeyHandle[in]	对称算法密钥句柄
	pucOutData[out]	输出数据
	puiOutDataLen[in,out]	输出数据长度
返回值：	0	成功
	非 0	失败，返回错误代码

7.3.45 单组数据消息鉴别码运算

原型：
```
int SAF_Mac(
    void * hKeyHandle,
    unsigned char * pucInData,
    unsigned int uiInDataLen,
    unsigned char * pucOutData,
    unsigned int * puiOutDataLen);
```

描述：单组数据消息鉴别码运算。

参数：

hKeyHandle[in]	对称算法密钥句柄
pucInData[in]	输入数据
uiInDataLen[in]	输入数据长度
pucOutData[out]	输出的 MAC
puiOutDataLen[out]	MAC 长度

返回值：

0	成功
非 0	失败，返回错误代码

7.3.46 多组数据消息鉴别码运算

原型：
```
int SAF_MacUpdate(
    void * hKeyHandle,
    unsigned char * pucInData,
    unsigned int * puiInDataLen);
```

描述：继续多组数据消息鉴别码运算。

参数：

hKeyHandle[in]	对称算法密钥句柄
pucInData[in]	输入数据
puiInDataLen[in]	输入数据长度

返回值：

0	成功
非 0	失败，返回错误代码

备注：运算完成后，还应调用 SAF_MacFinal 结束。

7.3.47 结束消息鉴别码运算

原型：
```
int SAF_MacFinal(
    void * hKeyHandle,
    unsigned char * pucOutData,
    unsigned int uiOutDataLen);
```

描述： 结束消息鉴别码运算。

参数：

hKeyHandle[in]	对称算法密钥句柄
pucOutData[out]	输出的 MAC
uiOutDataLen[out]	MAC 长度

返回值：

0	成功
非 0	失败，返回错误代码

7.4 消息类函数

7.4.1 概述

消息类函数主要是将数据按照 PKCS#7 格式进行封装，实现应用系统互联互通和信息共享，包含以下具体函数：

a) 编码 PKCS #7 格式的带签名的数字信封数据：SAF_Pkcs7_EncodeData；
b) 解码 PKCS #7 格式的带签名的数字信封数据：SAF_Pkcs7_DecodeData；
c) 编码 PKCS #7 格式的签名数据：SAF_Pkcs7_EncodeSignedData；
d) 解码 PKCS #7 格式的签名数据：SAF_Pkcs7_DecodeSignedData；
e) 编码 PKCS #7 格式的数字信封数据：SAF_Pkcs7_EncodeEnvelopedData；
f) 解码 PKCS #7 格式的数字信封数据：SAF_Pkcs7_DecodeEnvelopedData；
g) 编码 PKCS #7 格式的摘要数据：SAF_Pkcs7_EncodeDigestedData；
h) 解码 PKCS #7 格式的摘要数据：SAF_Pkcs7_DecodeDigestedData；
i) 编码基于 SM2 算法的带签名的数字信封数据：SAF_SM2_EncodeSignedAndEnvelopedData；
j) 解码基于 SM2 算法的带签名的数字信封数据：SAF_SM2_DecodeSignedAndEnvelopedData；
k) 编码基于 SM2 算法的签名数据：SAF_SM2_EncodeSignedData；
l) 解码基于 SM2 算法的签名数据：SAF_SM2_DecodeSignedData；
m) 编码基于 SM2 算法的数字信封：SAF_SM2_EncodeEnvelopedData；
n) 解码基于 SM2 算法的数字信封：SAF_SM2_DecodeEnvelopedData；
o) 解析 PKCS #7 格式的签名数据：SAF_Parse_Pkcs7_SignedData。

各函数返回值应符合附录 B 中的错误代码定义。

7.4.2 编码 PKCS #7 格式的带签名的数字信封数据

原型：

```
int SAF_Pkcs7_EncodeData(
    void * hAppHandle,
    unsigned char * pucSignContainerName,
    unsigned int uiSignContainerNameLen,
    unsigned int uiSignKeyUsage,
    unsigned char * pucSignerCertificate,
    unsigned int uiSignerCertificateLen,
    unsigned int uiDigestAlgorithms,
    unsigned char * pucEncCertificate,
    unsigned int uiEncCertificateLen,
    unsigned int uiSymmAlgorithm,
```

```
        unsigned char * pucData,
        unsigned int uiDataLen,
        unsigned char * pucDerP7Data,
        unsigned int * puiDerP7DataLen);
```

描述：　编码 PKCS #7 格式的带签名的数字信封数据。

参数：

hAppHandle[in]	应用接口句柄
pucSignContainerName[in]	签名私钥的容器名
uiSignContainerNameLen[in]	签名私钥的容器名长度
uiSignKeyUsage[in]	私钥的用途
pucSignerCertificate[in]	签名者证书
uiSignerCertificateLen[in]	签名者证书长度
uiDigestAlgorithms[in]	HASH 算法
pucEncCertificate[in]	接收者证书
uiEncCertificateLen[in]	接收者证书长度
uiSymmAlgorithm[in]	对称算法参数
pucData[in]	签名原文
uiDataLen[in]	签名原文长度
pucDerP7Data[out]	编码后的数据
puiDerP7DataLen[in,out]	编码后的数据长度

返回值：

0	成功
非 0	失败，返回错误代码

7.4.3 解码 PKCS #7 格式的带签名的数字信封数据

原型：

```
int SAF_Pkcs7_DecodeData(
    void * hAppHandle,
    unsigned char * pucDecContainerName,
    unsigned int uiDecContainerNameLen,
    unsigned int uiDecKeyUsage,
    unsigned char * pucDerP7Data,
    unsigned int uiDerP7DataLen,
    unsigned char * pucData,
    unsigned int * puiDataLen,
    unsigned char * pucSignerCertificate,
    unsigned int * puiSignerCertificateLen,
    unsigned int * puiDigestAlgorithms
    unsigned char * pucSignedData,
    unsigned int * puiSignedDataLen);
```

描述：　解码 PKCS #7 格式的带签名的数字信封数据。

参数：

hAppHandle[in]	应用接口句柄
pucDecContainerName[in]	解密用私钥的容器名
uiDecContainerNameLen[in]	解密用私钥的容器名长度
uiDecKeyUsage[in]	解密用私钥的用途
pucDerP7Data[in]	编码后的数据
uiDerP7DataLen[in]	编码后的数据长度
pucData[out]	签名原文
puiDataLen[in,out]	签名原文长度
pucSignerCertificate[out]	签名者证书
puiSignerCertificateLen [in, out]	签名者证书长度
puiDigestAlgorithms[out]	HASH 算法
pucSignedData[out]	Base64 编码的签名值
puiSignedDataLen[in,out]	签名值长度

返回值：

0	成功
非 0	失败，返回错误代码

7.4.4 编码 PKCS #7 格式的签名数据

原型：

```
int SAF_Pkcs7_EncodeSignedData(
    void * hAppHandle,
    unsigned char * pucSignContainerName,
    unsigned int uiSignContainerNameLen,
    unsigned int uiSignKeyUsage,
    unsigned char * pucSignerCertificate,
    unsigned int uiSignerCertificateLen,
    unsigned int uiDigestAlgorithms,
    unsigned char * pucData,
    unsigned int uiDataLen,
    unsigned char * pucDerP7SignedData,
    unsigned int * puiDerP7SignedDataLen);
```

描述：　编码 PKCS #7 格式的签名数据。

参数：

hAppHandle[in]	应用接口句柄
pucSignContainerName[in]	签名私钥的容器名
uiSignContainerNameLen[in]	签名私钥的容器名长度
uiSignKeyUsage[in]	私钥的用途

	pucSignerCertificate[in]	签名者证书
	uiSignerCertificateLen[in]	签名者证书长度
	uiDigestAlgorithms[in]	HASH 算法
	pucData[in]	待签名的数据
	uiDataLen[in]	待签名的数据长度
	pucDerP7SignedData[out]	带签名值的 P7 数据
	puiDerP7SignedDataLen[in,out]	带签名值的 P7 数据长度
返回值：	0	成功
	非 0	失败，返回错误代码

7.4.5 解码 PKCS #7 格式的签名数据

原型：

```
int SAF_Pkcs7_DecodeSignedData(
    void * hAppHandle,
    unsigned char * pucDerP7SignedData,
    unsigned int uiDerP7SignedDataLen,
    unsigned char * pucSignerCertificate,
    unsigned int * puiSignerCertificateLen,
    unsigned int * puiDigestAlgorithms,
    unsigned char * pucData,
    unsigned int * puiDataLen,
    unsigned char * pucSign,
    unsigned int * puiSignLen);
```

描述：解码 PKCS #7 格式的签名数据。

参数：	hAppHandle[in]	应用接口句柄
	pucDerP7SignedData[in]	签名后的数据
	uiDerP7SignedDataLen[in]	签名后的数据长度
	pucSignerCertificate[out]	签名者证书
	puiSignerCertificateLen[in,out]	签名者证书长度
	puiDigestAlgorithms[out]	HASH 算法
	pucData[out]	被签名的数据
	puiDataLen[in,out]	被签名的数据长度
	pucSign[out]	签名值
	puiSignLen[in,out]	签名值的长度
返回值：	0	成功
	非 0	失败，返回错误代码

7.4.6 编码 PKCS #7 格式的数字信封数据

原型：
```
int SAF_Pkcs7_EncodeEnvelopedData(
    void * hAppHandle,
    unsigned char * pucData,
    unsigned int uiDataLen,
    unsigned char * pucEncCertificate,
    unsigned int uiEncCertificateLen,
    unsigned int uiSymmAlgorithm,
    unsigned char * pucDerP7EnvelopedData,
    unsigned int * puiDerP7EnvelopedDataLen);
```

描述：编码 PKCS #7 格式的数字信封数据。

参数：

hAppHandle[in]	应用接口句柄
pucData[in]	原文数据
uiDataLen[in]	原文数据长度
pucEncCertificate[in]	接收者证书
uiEncCertificateLen[in]	接收者证书长度
uiSymmAlgorithm[in]	对称算法参数
pucDerP7EnvelopedData[out]	编码后的数字信封数据
puiDerP7EnvelopedDataLen[in,out]	编码后的数字信封数据长度

返回值：

0	成功
非 0	失败，返回错误代码

7.4.7 解码 PKCS #7 格式的数字信封数据

原型：
```
int SAF_Pkcs7_DecodeEnvelopedData(
    void * hAppHandle,
    unsigned char * pucDecContainerName,
    unsigned int uiDecContainerNameLen,
    unsigned int uiDecKeyUsage,
    unsigned char * pucDerP7EnvelopedData,
    unsigned int uiDerP7EnvelopedDataLen,
    unsigned char * pucData,
    unsigned int * puiDataLen);
```

描述：解码 PKCS #7 格式的数字信封数据。

参数：

hAppHandle[in]	应用接口句柄
pucDecContainerName[in]	解密用私钥的容器名

	uiDecContainerNameLen[in]	解密用私钥的容器名长度
	uiDecKeyUsage[in]	解密用私钥的用途
	pucDerP7EnvelopedData[in]	数字信封数据
	uiDerP7EnvelopedDataLen[in]	数字信封数据长度
	pucData[out]	解码后得到的数据原文
	puiDataLen [in,out]	解码后得到的数据原文数据长度
返回值：	0	成功
	非 0	失败，返回错误代码

7.4.8 编码 PKCS #7 格式的摘要数据

原型：

```
int SAF_Pkcs7_EncodeDigestedData(
    void * hAppHandle,
    unsigned int uiDigestAlgorithm,
    unsigned char * pucData,
    unsigned int puiDataLen,
    unsigned char * pucDerP7DigestedData,
    unsigned int * puiDerP7DigestedDataLen);
```

描述：使用指定的杂凑算法计算原文的摘要，并打包成符合 PKCS #7 摘要数据格式的数据包。

参数：	hAppHandle[in]	应用接口句柄
	uiDigestAlgorithm[in]	杂凑算法标识
	pucData[in]	原文数据
	puiDataLen [in]	原文数据长度
	pucDerP7DigestedData[out]	符合 PKCS #7 摘要数据格式的数据包
	puiDerP7DigestedDataLen[in,out]	符合 PKCS #7 摘要数据格式的数据包长度
返回值：	0	成功
	非 0	失败，返回错误代码

7.4.9 解码 PKCS #7 格式的摘要数据

原型：

```
int SAF_Pkcs7_DecodeDigestedData(
    void * hAppHandle,
    unsigned int uiDigestAlgorithm,
    unsigned char * pucDerP7DigestedData,
    unsigned int puiDerP7DigestedDataLen,
    unsigned char * pucData,
    unsigned int * puiDataLen,
    unsigned char * pucDigest,
    unsigned int * puiDigestLen);
```

描述：　从符合 PKCS #7 摘要数据格式的数据包中解析出原文及摘要值。

参数：

hAppHandle[in]	应用接口句柄
uiDigestAlgorithm[in]	杂凑算法标识
pucDerP7DigestedData[in]	符合 PKCS #7 摘要数据格式的数据包
puiDerP7DigestedDataLen[in]	符合 PKCS #7 摘要数据格式的数据包长度
pucData[out]	原文数据
puiDataLen [in,out]	原文数据长度
pucDigest[out]	摘要值
puiDigestLen[in,out]	摘要值长度

返回值：

0	成功
非 0	失败，返回错误代码

7.4.10 编码基于 SM2 算法的带签名的数字信封数据

原型：

```
int SAF_SM2_EncodeSignedAndEnvelopedData(
    void * hAppHandle,
    unsigned char * pucSignContainerName,
    unsigned int uiSignContainerNameLen,
    unsigned int uiSignKeyUsage,
    unsigned char * pucSignerCertificate,
    unsigned int uiSignerCertificateLen,
    unsigned int uiDigestAlgorithms,
    unsigned char * pucEncCertificate,
    unsigned int uiEncCertificateLen,
    unsigned int uiSymmAlgorithm,
    unsigned char * pucData,
    unsigned int uiDataLen,
    unsigned char * pucDerSignedAndEnvelopedData,
    unsigned int * puiDerSignedAndEnvelopedDataLen);
```

描述：　编码基于 SM2 算法的带签名的数字信封数据。

参数：

hAppHandle[in]	应用接口句柄
pucSignContainerName[in]	签名私钥的容器名
uiSignContainerNameLen[in]	签名私钥的容器名长度
uiSignKeyUsage[in]	私钥的用途
pucSignerCertificate[in]	签名者证书
uiSignerCertificateLen[in]	签名者证书长度
uiDigestAlgorithms[in]	HASH 算法

	pucEncCertificate[in]	接收者证书
	uiEncCertificateLen[in]	接收者证书长度
	uiSymmAlgorithm[in]	对称算法参数
	pucData[in]	签名原文
	uiDataLen[in]	签名原文长度
	pucDerSignedAndEnvelopedData [out]	DER 编码后的 SignedAndEnvelopedData 数字信封数据，SignedAndEnvelopedData 数据格式定义应符合GM/T 0010
	puiDerSignedAndEnvelopedDataLen[in,out]	带签名的数字信封数据长度
返回值：	0	成功
	非 0	失败，返回错误代码

7.4.11 解码基于 SM2 算法的带签名的数字信封数据

原型：

```
int SAF_SM2_DecodeSignedAndEnvelopedData(
    void * hAppHandle,
    unsigned char * pucDecContainerName,
    unsigned int uiDecContainerNameLen,
    unsigned int uiDecKeyUsage,
    unsigned char * pucDerSignedAndEnvelopedData,
    unsigned int uiDerSignedAndEnvelopedDataLen,
    unsigned char * pucData,
    unsigned int * puiDataLen,
    unsigned char * pucSignerCertificate,
    unsigned int * puiSignerCertificateLen,
    unsigned int * puiDigestAlgorithms);
```

描述：解码基于 SM2 算法的带签名的数字信封数据。

参数：	hAppHandle[in]	应用接口句柄
	pucDecContainerName[in]	解密用私钥的容器名
	uiDecContainerNameLen[in]	解密用私钥的容器名长度
	uiDecKeyUsage[in]	解密用私钥的用途
	pucDerSignedAndEnvelopedData[in]	DER 编码后的 SignedAndEnvelopedData 数字信封数据，SignedAndEnvelopedData 数据格式定义应符合 GM/T 0010
	uiDerSignedAndEnvelopedDataLen[in]	编码后的带签名的数字信封数据长度
	pucData[out]	解码后得到的签名原文
	puiDataLen[in,out]	签名原文长度

	pucSignerCertificate[out]	签名者证书
	puiSignerCertificateLen[in,out]	签名者证书长度
	puiDigestAlgorithms[out]	HASH 算法标识
返回值：	0	成功
	非 0	失败，返回错误代码

7.4.12 编码基于 SM2 算法的签名数据

原型：

```
int SAF_SM2_EncodeSignedData(
    void * hAppHandle,
    unsigned char * pucSignContainerName,
    unsigned int uiSignContainerNameLen,
    unsigned int uiSignKeyUsage,
    unsigned char * pucSignerCertificate,
    unsigned int uiSignerCertificateLen,
    unsigned int uiDigestAlgorithms,
    unsigned char * pucData,
    unsigned int uiDataLen,
    unsigned char * pucDerSignedData,
    unsigned int * puiDerSignedDataLen);
```

描述：编码基于 SM2 算法的签名数据。

参数：	hAppHandle[in]	应用接口句柄
	pucSignContainerName[in]	签名私钥的容器名
	uiSignContainerNameLen[in]	签名私钥的容器名长度
	uiSignKeyUsage[in]	私钥的用途
	pucSignerCertificate[in]	签名者证书
	uiSignerCertificateLen[in]	签名者证书长度
	uiDigestAlgorithms[in]	HASH 算法标识
	pucData[in]	待签名的数据
	uiDataLen[in]	待签名的数据长度
	pucDerSignedData[out]	DER 编码的 SignedData 签名数据，SignedData 数据格式定义应符合 GM/T 0010
	puiDerSignedDataLen[in,out]	带签名数据长度
返回值：	0	成功
	非 0	失败，返回错误代码

7.4.13 解码基于 SM2 算法的签名数据

原型：

```
int SAF_SM2_DecodeSignedData(
    void * hAppHandle,
    unsigned char * pucDerSignedData,
    unsigned int uiDerSignedDataLen,
    unsigned char * pucSignerCertificate,
    unsigned int * puiSignerCertificateLen,
    unsigned int * puiDigestAlgorithms,
    unsigned char * pucData,
    unsigned int * puiDataLen,
    unsigned char * pucSign,
    unsigned int * puiSignLen);
```

描述：解码基于 SM2 算法的签名数据。

参数：

hAppHandle[in]	应用接口句柄
pucDerSignedData[in]	DER 编码的 SignedData 签名数据，SignedData 数据格式定义应符合 GM/T 0010
uiDerSignedDataLen[in]	签名数据的长度
pucSignerCertificate[out]	DER 编码的签名者证书
puiSignerCertificateLen[in,out]	签名者证书长度
puiDigestAlgorithms[out]	HASH 算法标识
pucData[out]	被签名的数据
puiDataLen[in,out]	被签名的数据长度
pucSign[out]	签名值
puiSignLen[in,out]	签名值的长度

返回值：

0	成功
非 0	失败，返回错误代码

7.4.14 编码基于 SM2 算法的数字信封

原型：

```
int SAF_SM2_EncodeEnvelopedData(
    void * hAppHandle,
    unsigned char * pucData,
    unsigned int uiDataLen,
    unsigned char * pucEncCertificate,
    unsigned int uiEncCertificateLen,
    unsigned int uiSymmAlgorithm,
    unsigned char * pucDerEnvelopedData,
    unsigned int * puiDerEnvelopedDataLen);
```

描述：　编码基于 SM2 算法的数字信封数据。

参数：

hAppHandle[in]	应用接口句柄
pucData[in]	原文数据
uiDataLen[in]	原文数据长度
pucEncCertificate[in]	DER 编码的接收者证书
uiEncCertificateLen[in]	接收者证书长度
uiSymmAlgorithm[in]	对称算法标识
pucDerEnvelopedData[out]	DER 编码后的 EnvelopedData 数字信封数据，EnvelopedData 数据格式定义应符合 GM/T 0010
puiDerEnvelopedDataLen[in,out]	编码后的数字信封数据长度

返回值：

0	成功
非 0	失败，返回错误代码

7.4.15 解码基于 SM2 算法的数字信封

原型：

```
int SAF_SM2_DecodeEnvelopedData(
    void * hAppHandle,
    unsigned char * pucDecContainerName,
    unsigned int uiDecContainerNameLen,
    unsigned int uiDecKeyUsage,
    unsigned char * pucDerEnvelopedData,
    unsigned int uiDerEnvelopedDataLen,
    unsigned char * pucData,
    unsigned int * puiDataLen);
```

描述：　解码基于 SM2 算法的数字信封数据。

参数：

hAppHandle[in]	应用接口句柄
pucDecContainerName[in]	解密用私钥的容器名
uiDecContainerNameLen[in]	解密用私钥的容器名长度
uiDecKeyUsage[in]	解密用私钥的用途
pucDerEnvelopedData[in]	DER 编码的 EnvelopedData 数字信封数据，EnvelopedData 数据格式定义应符合 GM/T 0010
uiDerEnvelopedDataLen[in]	数字信封数据长度
pucData[out]	解码后得到的数据原文
puiDataLen [in,out]	解码后得到的数据原文长度

返回值：

0	成功
非 0	失败，返回错误代码

7.4.16 解析 PKCS #7 格式的签名数据

原型：
```
int SAF_Parse_Pkcs7_SignedData(
    void * hAppHandle,
    unsigned int nType,
    unsigned char * pucDerP7SignedData,
    unsigned int uiDerP7SignedDataLen,
    unsigned char * pucInfo,
    unsigned int * puiInfoLen);
```

描述：解码基于 SM2 算法的数字信封数据。

参数：

hAppHandle[in]	应用接口句柄
nType[in]	解析类型： 1:版本号 2:算法类型,RSA 或 SM2 3:签名原文 4:签名证书 5:为签名值
pucDerP7SignedData[in]	DER 编码后的 PKCS #7 签名数据
uiDerP7SignedDataLen[in]	DER 编码后的 PKCS #7 签名数据长度
pucInfo[out]	解析出来的信息
puiInfoLen[in,out]	解析出来的信息长度

返回值：

0	成功
非 0	失败,返回错误代码

8 验证方法

8.1 验证环境

接口提供方实现本文件所规定的接口,提供测试程序和测试程序源代码。接口需求方可选择使用接口提供方提供的测试程序验证接口,也可以自编写测试程序来验证接口。验证接口,应具备以下验证环境：

a) 运行接口服务所需的操作系统及测试程序软件；
b) 支持密码设备应用接口调用的密码设备；
c) 支持存储证书和密钥的存储介质(如智能密码钥匙)。

8.2 密码服务环境操作验证

8.2.1 密码服务空间验证

验证目的：

验证通过密码服务接口是否正确创建,使用,清除密码服务空间操作。

验证条件：

密码服务已就绪。

验证过程：

a) 正常情况验证：

步骤 1：调用 SAF_Initialize 初始化环境；

步骤 2：调用 SAF_GetVersion 获取接口版本信息；

步骤 3：调用 SAF_Finalize 清除环境；

步骤 4：调用 SAF_GetVersion 获取接口版本信息；

b) 异常情况验证：

步骤 1：调用 SAF_GetVersion 获取接口版本信息。

通过标准：

a) 正常情况验证：

步骤 2 获取接口版本信息成功，步骤 4 获取接口版本信息失败返回错误码；

b) 异常情况验证：

步骤 1 获取接口版本信息失败返回错误码。

8.2.2 用户登录验证

验证目的：

验证通过密码服务接口都是正确登录用户，修改用户口令，注销登录。

验证条件：

密码服务已就绪，密码服务空间已完成初始化。

验证过程：

a) 正常情况验证：

步骤 1：调用 SAF_Login 登录用户；

步骤 2：调用 SAF_ChangePin 修改口令；

步骤 3：调用 SAF_Logout 注销登录；

步骤 4：调用 SAF_ChangePin 修改口令；

b) 异常情况验证：

步骤 1：调用 SAF_ChangePin 修改口令。

通过标准：

a) 正常情况验证：

步骤 2 获取接口版本信息成功，步骤 4 获取接口版本信息失败返回错误码。

b) 异常情况验证：

步骤 1 获取接口版本信息失败返回错误码。

8.3 证书类函数功能验证

8.3.1 根 CA 证书验证

验证目的：

验证通过密码服务接口是否正确执行根 CA 证书的添加、删除操作。

验证条件：

密码服务已就绪，密码服务空间已完成初始化。

验证过程：

a) 添加 CA 根证书：

步骤 1：调用 SAF_GetRootCaCertificateCount 获取 CA 根证书的个数 m；

步骤 2：调用 SAF_AddTrustedRootCaCertificate 添加 CA 根证书 rootCA1；

步骤 3：调用 SAF_GetRootCaCertificateCount 获取 CA 根证书的个数 n；

步骤 4：输入索引 n－1，调用 SAF_GetRootCaCertificateCount 获取 CA 根证书 rootCA2；

b) 删除 CA 根证书：

步骤 1：执行验证过程 a)；

步骤 2：调用 SAF_RemoveRootCaCertificate 删除指定 CA 根证书；

步骤 3：调用 SAF_GetRootCaCertificateCount 获取 CA 根证书的个数 n1；

步骤 4：输入位置索引 n1－1，调用 SAF_GetRootCaCertificateCount 获取已删除的 CA 根证书。

通过标准：

a) 正常情况验证：

n＝m＋1 且能够得到与设置相同的 CA 根证书(rootCA1 ＝ rootCA2)；

b) 异常情况验证：

n1＝m 且 SAF_GetRootCaCertificateCount 返回错误码 SAR_CertNotFountErr。

8.3.2 CA 证书验证

验证目的：

验证通过密码服务接口是否正确执行 CA 证书的添加、删除操作。

验证条件：

密码服务已就绪，密码服务空间已完成初始化，添加根 CA 证书。

验证过程：

a) 添加 CA 证书：

步骤 1：调用 SAF_GetCaCertificateCount 获取 CA 证书的个数 m；

步骤 2：调用 SAF_AddCaCertificate 添加 CA 证书 CA1；

步骤 3：调用 SAF_GetCaCertificateCount 获取 CA 证书的个数 n；

步骤 4：输入索引 n－1，调用 SAF_GetCaCertificate 获取 CA 根证书；

b) 删除 CA 证书：

步骤 1：执行验证过程 a)；

步骤 2：调用 SAF_RemoveCaCertificate 删除指定 CA 证书；

步骤 3：调用 SAF_GetCaCertificateCount 获取 CA 证书的个数 n1；

步骤 4：输入位置索引 n1－1，调用 SAF_GetCaCertificate 获取已删除的 CA 证书。

通过标准：

a) 正常情况验证：

n＝m＋1 且能够得到与设置相同的 CA 证书(CA1 ＝ CA2)；

b) 异常情况验证：

n1＝m 且 SAF_GetCaCertificateCount 返回错误码 SAR_CertNotFountErr。

8.3.3 证书状态验证

验证目的：

验证通过密码服务接口是否正确添加 CRL、验证用户证书、根据 CRL 文件获取用户注销状

态、根据 OCSP 获取证书状态、通过 LDAP 方式获取证书、通过 LDAP 方式获取证书对应的 CRL 操作。

验证条件：

密码服务已就绪，密码服务空间已完成初始化，添加根 CA 证书，添加 CA 证书。

验证过程：

a) 正常情况验证：

步骤 1：调用 SAF_AddCrl 添加 CRL；

步骤 2：调用 SAF_VerifyCertificate 验证用户证书；

步骤 3：调用 SAF_VerifyCertificateByCrl 根据 CRL 文件获取用户证书注销状态；

步骤 4：调用 SAF_GetCertificateStateByOCSP 根据 OCSP 获取用户证书注销状态；

步骤 5：调用 SAF_GetCertFromLdap 通过 LDAP 方式获取证书对应的 CRL；

b) 异常情况验证：

使用非法参数调用本接口，应返回错误码。

通过标准：

a) 正常情况验证：

待验证的证书通过 CRL、OCSP、LDAP 验证证书状态；

b) 异常情况验证：

可得到预期结果。

8.3.4 用户证书验证

验证目的：

验证通过密码服务接口是否正确获取证书信息、获取证书扩展信息、列举用户证书、列举用户密钥容器信息、释放列举用户证书的内存、释放列举密钥容器信息的内存。

验证条件：

密码服务已就绪，密码服务空间已完成初始化，添加根 CA 证书，添加 CA 证书。

验证过程：

a) 正常情况验证：

步骤 1：调用 SAF_GetCertificateInfo 获取证书信息；

步骤 2：调用 SAF_GetExtTypeInfo 获取证书扩展信息；

步骤 3：调用 SAF_EnumCertificates 列举用户证书；

步骤 4：调用 SAF_EnumKeyContainerInfo 列举用户的密钥容器信息；

步骤 5：调用 SAF_EnumCertificatesFree 释放列举用户证书的内存；

步骤 6：调用 SAF_EnumkeyContainerInfoFree 释放列举密钥容器信息的内存；

b) 异常情况验证：

使用非法参数调用本接口，应返回错误码。

通过标准：

a) 正常情况验证：

获取用户信息、扩展信息，释放用户证书、密钥容器的内存；

b) 异常情况验证：

可得到预期结果。

8.4 签名验签验证

8.4.1 RSA 签名运算验证

验证目的：

验证是否能正确使用预定 RSA 签名私钥，对预定数据进行数字签名，并输出签名结果。

验证条件：

设备已就绪，环境已初始化，已列举用户证书。

验证过程：

a) 正常情况验证：

步骤 1：调用 SAF_Login，验证用户口令；

步骤 2：调用 SAF_RsaSign 接口，指定 SHA256 哈希算法，对预定数据进签名，返回签名结果；

步骤 3：调用 SAF_GetRsaPublicKey 接口，导出 RSA 签名公钥；

b) 异常情况验证：

使用非法参数调用本接口，应返回错误码；

用户未登录，应返回错误码；

RSA 密钥对不存在，应返回错误码。

通过标准：

a) 正常情况验证：

用步骤 3 返回的签名公钥对步骤 2 返回的签名结果验签应成功。

b) 异常情况验证：

得到预期结果。

8.4.2 RSA 验证签名运算验证

验证目的：

验证是否能正确使用外部输入的 RSA 公钥对数据进行签名验证。

验证条件：

设备已就绪，环境已初始化，已列举用户证书，已登录。

验证过程：

a) 正常情况验证：

步骤 1：使用预置的 RSA 签名公钥、原文和签名值，调用 SAF_RsaVerifySign 接口；

步骤 2：调用 SAF_RsaSign 接口，指定 SHA256 哈希算法，对预定数据进行签名，得到签名结果；

步骤 3：调用 SAF_GetRsaPublicKey 接口，导出 RSA 签名公钥；

步骤 4：使用步骤 2 的原文和签名结果、步骤 3 得到的签名公钥，调用 SAF_RsaVerifySign 接口；

b) 异常情况验证：

使用非法参数调用本接口，应返回错误码。

通过标准：

a) 正常情况验证：

待验证的 RSA 公钥和原文、签名值匹配应验签成功，不匹配应验签失败。

b) 异常情况验证：

可得到预期结果。

8.4.3 基于证书的 RSA 的公钥验证

验证目的：

验证是否能正确使用外部输入的 RSA 证书对数据进行签名验证。

验证条件：

设备已就绪，环境已初始化，已列举用户证书，已登录。

验证过程：

a) 正常情况验证：

步骤 1：使用预置的 RSA 签名证书、原文和签名值，调用 SAF_VerifySignByCert 接口；

步骤 2：调用 SAF_RsaSign 接口，指定 SHA256 哈希算法，对预定数据进行签名，得到签名结果；

步骤 3：使用步骤 2 的原文和签名结果、列举证书时取到的签名证书，调用 SAF_VerifySignByCert 接口；

b) 异常情况验证：

使用非法参数调用本接口，应返回错误码。

通过标准：

a) 正常情况验证：

待验证的 RSA 证书和原文、签名值匹配应验签成功，不匹配应验签失败。

b) 异常情况验证：

可得到预期结果。

8.4.4 SM2 签名运算

验证目的：

验证是否能正确使用预定 SM2 签名私钥，对预定数据进行数字签名，并输出签名结果。

验证条件：

设备已就绪，环境已初始化，已列举用户证书。

验证过程：

a) 正常情况验证：

步骤 1：调用 SAF_Login，验证用户口令；

步骤 2：调用 SAF_Sm2Sign 接口，对预定数据进行签名，返回签名结果；

步骤 3：调用 SAF_GetSm2PublicKey 接口，导出 SM2 签名公钥；

b) 异常情况验证：

使用非法参数调用本接口，应返回错误码；

用户未登录，应返回错误码；

SM2 密钥对不存在，应返回错误码。

通过标准：

a) 正常情况验证：

用步骤 3 返回的签名公钥对步骤 2 返回的签名结果验签应成功；

b) 异常情况验证：

得到预期结果。

8.4.5 SM2 验证签名运算

验证目的：

验证是否能正确使用外部输入的 SM2 公钥对数据进行签名验证。

验证条件：

设备已就绪，环境已初始化，已列举用户证书，已登录。

验证过程如下。

a) 正常情况验证：

步骤1：使用预置的SM2签名公钥、原文和签名值，调用SAF_Sm2VerifySign接口；

步骤2：调用SAF_Sm2Sign接口，对预定数据进行签名，得到签名结果；

步骤3：调用SAF_GetSm2PublicKey接口，导出SM2签名公钥；

步骤4：使用步骤2的原文和签名结果、步骤3得到的签名公钥，调用SAF_Sm2VerifySign接口。

b) 异常情况验证：

使用非法参数调用本接口，应返回错误码。

通过标准：

a) 正常情况验证：

待验证的SM2公钥和原文、签名值匹配应验签成功，不匹配应验签失败；

b) 异常情况验证：

可得到预期结果。

8.4.6 基于证书的SM2公钥验证

验证目的：

验证是否能正确使用外部输入的SM2证书对数据进行签名验证。

验证条件：

设备已就绪，环境已初始化，已列举用户证书，已登录。

验证过程：

a) 正常情况验证：

步骤1：使用预置的SM2签名证书、原文和签名值，调用SAF_Sm2VerifySignByCert接口；

步骤2：调用SAF_Sm2Sign接口，对预定数据进行签名，得到签名结果；

步骤3：使用步骤2的原文和签名结果、列举证书时取到的签名证书，调用SAF_Sm2VerifySignByCert接口；

b) 异常情况验证：

使用非法参数调用本接口，应返回错误码。

通过标准：

a) 正常情况验证：

待验证的SM2证书和原文、签名值匹配应验签成功，不匹配应验签失败；

b) 异常情况验证：

可得到预期结果。

8.5 摘要计算验证

8.5.1 单块摘要计算

验证目的：

验证是否能正确进行单块摘要运算。

验证条件：

无。

验证过程：

a) 正常情况验证：

调用 SAF_Hash 接口，对预定数据计算摘要值；

b) 异常情况验证：

使用非法参数调用本接口，应返回错误码。

通过标准：

a) 正常情况验证：

计算得到的摘要值应与预定结果一致，计算时设置 pucOutData ＝NULL 时，应能通过 puiOutDataLen 返回结果数据实际长度；

b) 异常情况验证：

得到预期结果。

8.5.2 多块摘要计算

验证目的：

验证是否能正确进行多块摘要运算。

验证条件：

初始化环境函数 SAF_Initialize 已经成功调用。

验证过程：

a) 正常情况验证：

步骤 1：调用 SAF_CreateHashObj 接口，创建 HASH 对象；

步骤 2：调用 SAF_HashUpdate 接口，对多块预定数据继续进行 HASH 运算；

步骤 3：调用 SAF_HashFinal 接口，结束 HASH 运算，得到摘要值；

步骤 4：调用 SAF_DestroyHashObj 接口，删除 HASH 对象；

b) 异常情况验证：

使用非法参数调用上述接口，应返回错误码。

通过标准：

a) 正常情况验证：

步骤 3 中得到的摘要值应与预定结果一致，步骤 3 中设置 pucOutData ＝NULL 时，应能通过 puiOutDataLen 返回摘要结果实际长度；

b) 异常情况验证：

得到预期结果。

8.6 非对称加解密验证

8.6.1 SM2 公钥加密

验证目的：

验证是否能正确采用外部 SM2 公钥对输入数据进行加密，并输出加密结果。

验证条件：

完成环境初始化。

验证过程：

a) 正常情况验证：

使用预定 SM2 密钥对，调用 SAF_Sm2PublicKeyEnc 接口，对预定原文进行加密，返回加密结果；

b) 异常情况验证：

使用非法参数调用本接口，应返回错误码。

通过标准：

a) 正常情况验证：
使用预定私钥对加密密文进行解密，得到结果与预定原文相同；加密数据格式应符合符合GM/T 0009中加密数据格式的定义；

b) 异常情况验证：
得到预期结果。

8.6.2 基于证书的SM2公钥加密

验证目的：

验证是否能正确采用外部证书的SM2公钥对输入数据进行加密，并输出加密结果。

验证条件：

完成环境初始化，使用预定SM2私钥申请证书。

验证过程：

a) 正常情况验证：
使用预定容器内调用SAF_Sm2PublicKeyEncByCert接口，对预定原文进行加密，返回加密结果；

b) 异常情况验证：
使用非法参数调用本接口，应返回错误码。

通过标准：

a) 正常情况验证：
使用预定私钥对加密密文进行解密，得到结果与预定原文相同。加密数据格式应符合符合GM/T 0009中加密数据格式的定义；

b) 异常情况验证：
得到预期结果。

8.6.3 基于证书的SM2解密

验证目的：

验证是否能正确采用容器内证书标识对应私钥进行解密，并输出原文。

验证条件：

完成环境初始化，证书标识对应私钥已存在。

验证过程：

a) 正常情况验证：
使用预定容器内调用SAF_Sm2PublicKeyDecByCert接口，对预定密文进行解密，返回明文数据；

b) 异常情况验证：
使用非法参数调用本接口，应返回错误码；
解密密钥不存在，应返回错误码。

通过标准：

a) 正常情况验证：
返回的明文数据与预定明文一致；

b) 异常情况验证：
得到预期结果。

8.7 对称加解密验证

8.7.1 单块加密

验证目的：

验证是否能正确进行单块数据的加密操作。

验证条件：

初始化环境函数 SAF_Initialize、用户登录函数 SAF_Login、创建对称算法对象 SAF_CreateSymmAlgoObj 已经成功调用，会话密钥已生成或已导入或已协商。

验证过程：

a) 正常情况验证：
调用 SAF_SymmEncrypt 接口，对预定数据计算密文值；

b) 异常情况验证：
使用非法参数调用 SAF_SymmEncrypt 接口，应返回错误码。

通过标准：

a) 正常情况验证：
计算中得到的密文值应与预定结果一致，计算中设置 pucOutData ＝NULL 时，应能通过 puiOutDataLen 返回结果数据实际长度；

b) 异常情况验证：
得到预期结果。

8.7.2 单块解密

验证目的：

验证是否能正确进行多块数据的加密操作。

验证条件：

初始化环境函数 SAF_Initialize、用户登录函数 SAF_Login、创建对称算法对象 SAF_CreateSymmAlgoObj 已经成功调用，会话密钥已生成或已导入或已协商。

验证过程：

a) 正常情况验证：
步骤 1：调用 SAF_SymmEncryptUpdate 接口，对多个预定明文数据分组进行加密，返回密文数据；
步骤 2：调用 SAF_SymmEncryptFinal 接口结束加密，得到密文数据；

b) 异常情况验证：
使用非法参数调用上述接口，应返回错误码；不执行步骤 1，直接执行步骤 2 应返回错误码；

通过标准：

a) 正常情况验证：
步骤 1 和步骤 2 返回的加密密文与预定密文一致。步骤 1 和步骤 2 中设置 pucOutData ＝NULL 时，应能通过 puiOutDataLen 返回结果数据实际长度；

b) 异常情况验证：
得到预期结果。

8.7.3 多块加密

验证目的：

验证是否能正确进行单块数据解密运算。

验证条件：

初始化环境函数 SAF_Initialize、用户登录函数 SAF_Login、创建对称算法对象 SAF_CreateSymmAlgoObj 已经成功调用，会话密钥已生成或已导入或已协商。

验证过程：

a) 正常情况验证：

步骤 1：调用 SAF_SymmEncryptUpdate 接口，对多个预定明文数据分组进行加密，返回密文数据；

步骤 2：调用 SAF_SymmEncryptFinal 接口结束加密，得到密文数据；

b) 异常情况验证：

使用非法参数调用上述接口，应返回错误码；

不执行步骤 1，直接执行步骤 2 应返回错误码。

通过标准：

a) 正常情况验证：

计算中得到的密文值应与预定结果一致，计算中设置 pucOutData = NULL 时，应能通过 puiOutDataLen 返回结果数据实际长度；

b) 异常情况验证：

得到预期结果。

8.7.4 多块解密

验证目的：

验证是否能正确进行多块数据解密运算。

验证条件：

初始化环境函数 SAF_Initialize、用户登录函数 SAF_Login、创建对称算法对象 SAF_CreateSymmAlgoObj 已经成功调用，会话密钥已生成或已导入或已协商。

验证过程：

a) 正常情况验证：

步骤 1：调用 SAF_SymmDecryptUpdate 接口，对多个预定密文数据分组进行解密，返回明文数据；

步骤 2：调用 SAF_SymmDecryptFinal 接口结束解密，得到明文数据；

b) 异常情况验证：

使用非法参数调用上述接口，应返回错误码；不执行步骤 1，直接执行步骤 2 应返回错误码。

通过标准：

a) 正常情况验证：

步骤 1 和步骤 2 返回的解密明文与预定明文一致，步骤 1 和步骤 2 中设置 pucOutData = NULL 时，应能通过 puiOutDataLen 返回结果数据实际长度；

b) 异常情况验证：

得到预期结果。

8.8 生成密钥对验证

验证目的：

验证是否能正确生成符合规范的 SM2 密钥对。

验证条件：

无。

验证过程：

a) 正常情况验证：

步骤 1：调用 SAF_Initialize 接口初始化环境；

步骤 2：调用 SAF_Login 接口进行用户登录；

步骤 3：调用 SAF_GenSm2KeyPair 接口生成 SM2 密钥对；

步骤 4：调用 SAF_GetSm2PublicKey 接口取出 SM2 公钥；

步骤 5：调用 SAF_Logout 接口注销用户登录；

步骤 6：调用 SAF_Finalize 接口清除环境；

b) 异常情况验证：

步骤 1、2、3 中输入异常参数，应返回错误码。

通过标准：

a) 正常情况验证：

取出的 SM2 公钥内容应为 GM/T 0009 SM2 密码算法使用规范定义的 SM2PublicKey，采用 ASN.1 DER 编码后进行 Base64 编码；公钥内容经验证有效，验证方法按照 GM/T 0003.1 定义公钥内容验证；

b) 异常情况验证：

得到预期结果。

8.9 PKCS#7 运算验证

8.9.1 编码 PKCS#7 格式的带签名的数据签名信封

验证目的：

验证是否能正确进行 PKCS #7 格式的带签名的数字信封编码运算。

验证条件：

无。

验证过程：

a) 正常情况验证：

调用 SAF_Pkcs7_EncodeData 接口，对预定数据进行 PKCS #7 格式的带签名的数字信封编码运算；

b) 异常情况验证：

使用非法参数调用本接口，应返回错误码。

通过标准：

a) 正常情况验证：

计算得到的结果值应与预定结果一致；

b) 异常情况验证：

得到预期结果。

8.9.2 解码 PKCS#7 格式的带签名的数据签名信封

验证目的：

验证是否能正确进行 PKCS #7 格式的带签名的数字信封解码运算。

验证条件：

无。

验证过程：

a) 正常情况验证：

调用 SAF_Pkcs7_DecodeData 接口，对预定数据进行 PKCS #7 格式的带签名的数字信封解码运算；

b) 异常情况验证：

使用非法参数调用本接口，应返回错误码。

通过标准：

a) 正常情况验证：

计算得到的结果值应与预定结果一致；

b) 异常情况验证：

得到预期结果。

8.10 SM2 消息类运算验证

8.10.1 编码基于 SM2 算法的带签名的数字信封

验证目的：

验证是否能正确进行 SM2 算法的带签名的数字信封编码运算。

验证条件：

无。

验证过程：

a) 正常情况验证：

调用 SAF_SM2_EncodeSignedAndEnvelopedData 接口，对预定数据进行 SM2 的带签名的数字信封编码运算；

b) 异常情况验证：

使用非法参数调用本接口，应返回错误码。

通过标准：

a) 正常情况验证：

计算得到的结果值应与预定结果一致；

b) 异常情况验证：

得到预期结果。

8.10.2 解码基于 SM2 算法的带签名的数据信封

验证目的：

验证是否能正确进行基于 SM2 算法的带签名的数字信封解码运算。

验证条件：

无。

验证过程：

a) 正常情况验证：

调用 SAF_SM2_DecodeSignedAndEnvelopedData 接口，对预定数据进行进行基于 SM2 算法的带签名的数字信封解码运算；

b) 异常情况验证：

使用非法参数调用本接口，应返回错误码。

通过标准：

a) 正常情况验证：
计算得到的结果值应与预定结果一致；

b) 异常情况验证：
得到预期结果。

8.11 Base64 编码验证

8.11.1 创建 Base64 对象

验证目的：

验证是否能正确创建 Base 对象。

验证条件：

无。

验证过程：

a) 正常情况验证：
调用 SAF_Base64_CreateBase64Obj 接口，为任意长度数据的 Base64 编解码创建 Base64 对象；

b) 异常情况验证：
使用非法参数调用本接口，应返回错误码。

通过标准：

a) 正常情况验证：
计算得到的结果值应与预定结果一致；

b) 异常情况验证：
得到预期结果。

8.11.2 Base64 编码

验证目的：

验证是否能正确进行 Base64 编码。

验证条件：

正确创建 Base64 对象。

验证过程：

a) 正常情况验证：
调用 SAF_Base64_Encode 接口，对预定数据进行 Base64 编码运算；

b) 异常情况验证：
使用非法参数调用本接口，应返回错误码。

通过标准：

a) 正常情况验证：
计算得到的结果值应与预定结果一致；

b) 异常情况验证：
得到预期结果。

8.11.3 Base64 解码

验证目的：

验证是否能正确进行 Base64 解码。

验证条件：

正确创建 Base64 对象。

验证过程：

a) 正常情况验证：

调用 SAF_Base64_Decode 接口，对预定数据进行 Base64 编码运算；

b) 异常情况验证：

使用非法参数调用本接口，应该返回错误码。

通过标准：

a) 正常情况验证：

计算得到的结果值应与预定结果一致；

b) 异常情况验证：

得到预期结果。

附 录 A
（规范性）
SM9 密码算法应用接口

A.1 SM9 算法相关数据结构

A.1.1 SM9 主私钥数据结构

SM9 主私钥数据结构符合表 A.1。

表 A.1 SM9 主私钥数据结构

字段名称	数据长度/字节	含义
bits	4	模数的实际位长度
s	SM9ref_MAX_LEN	主私钥

C 语言的数据结构定义：

```
#define SM9ref_MAX_BITS          256
#define SM9ref_MAX_LEN           ((SM9ref_MAX_BITS+7) / 8)
typedef struct SM9refMasterPrivateKey_st
{
ULONG bits;
BYTE s[SM9ref_MAX_LEN];
} SM9MasterPrivateKey;
```

备注：SM9 主私钥类型用于 SM9 签名主私钥和 SM9 加密主私钥

A.1.2 SM9 签名主公钥数据结构

SM9 签名主公钥数据结构符合表 A.2。

表 A.2 SM9 签名主公钥数据结构

字段名称	数据长度/字节	含义
bits	4	模数的实际位长度
xa	SM9ref_MAX_LEN	曲线上点 X 坐标高维
xb	SM9ref_MAX_LEN	曲线上点 X 坐标低维
ya	SM9ref_MAX_LEN	曲线上点 Y 坐标高维
yb	SM9ref_MAX_LEN	曲线上点 Y 坐标低维

C 语言的数据结构定义：

```
typedef struct SM9refSignMasterPublicKey_st
{
  ULONG bits;
  BYTE xa[SM9ref_MAX_LEN];
```

```
    BYTE xb[SM9ref_MAX_LEN];
    BYTE ya[SM9ref_MAX_LEN];
    BYTE yb[SM9ref_MAX_LEN];
} SM9SignMasterPublicKey;
```

A.1.3 SM9 加密主公钥数据结构

SM9 加密主公钥数据结构符合表 A.3。

表 A.3 SM9 加密主公钥数据结构

字段名称	数据长度/字节	含义
bits	4	模数的实际位长度
x	SM9ref_MAX_LEN	曲线上点 X 坐标
y	SM9ref_MAX_LEN	曲线上点 Y 坐标

C 语言的数据结构定义：

```
typedef struct SM9refEncMasterPublicKey_st
{
  ULONG bits;
  BYTE x[SM9ref_MAX_LEN];
    BYTE y[SM9ref_MAX_LEN];
} SM9EncMasterPublicKey;
```

A.1.4 SM9 用户签名私钥数据结构

SM9 用户签名私钥数据结构符合表 A.4。

表 A.4 SM9 用户签名私钥数据结构

字段名称	数据长度/字节	含义
bits	4	模数的实际位长度
x	SM9ref_MAX_LEN	曲线上点 X 坐标
y	SM9ref_MAX_LEN	曲线上点 Y 坐标

C 语言的数据结构定义：

```
typedef struct SM9refSignUserPrivateKey_st
{
  ULONG bits;
  BYTE x[SM9ref_MAX_LEN];
    BYTE y[SM9ref_MAX_LEN];
} SM9SignUserPrivateKey;
```

A.1.5 SM9 用户加密私钥数据结构

SM9 用户加密私钥数据结构符合表 A.5。

表 A.5 SM9 用户加密私钥数据结构

字段名称	数据长度/字节	含义
bits	4	模数的实际位长度
xa	SM9ref_MAX_LEN	曲线上点 X 坐标高维
xb	SM9ref_MAX_LEN	曲线上点 X 坐标低维
ya	SM9ref_MAX_LEN	曲线上点 Y 坐标高维
yb	SM9ref_MAX_LEN	曲线上点 Y 坐标低维

C 语言的数据结构定义：

```
typedef struct SM9refEncUserPrivateKey_st
{
  ULONG bits;
  BYTE xa[SM9ref_MAX_LEN];
    BYTE xb[SM9ref_MAX_LEN];
    BYTE ya[SM9ref_MAX_LEN];
    BYTE yb[SM9ref_MAX_LEN];
} SM9EncUserPrivateKey;
```

A.1.6 SM9 加密数据结构

SM9 加密数据结构符合表 A.6。

表 A.6 SM9 加密数据结构

字段名称	数据长度/字节	含义
EncType	4	SM9 加密算法机制
x	SM9ref_MAX_LEN	椭圆曲线上点 X 坐标
y	SM9ref_MAX_LEN	椭圆曲线上点 Y 坐标
h	32	MAC 值
L	4	密文数据长度
C	L	密文数据

C 语言的数据结构定义：

```
typedef struct SM9refCipher_st
{
  ULONG encType;
  BYTE x[SM9ref_MAX_LEN];
    BYTE y[SM9ref_MAX_LEN];
  BYTE h[32];
    ULONG L;
    BYTE C[1]
} SM9Cipher;
```

A.1.7 SM9 签名数据结构

SM9 签名数据结构符合表 A.7。

表 A.7 SM9 签名数据结构

字段名称	数据长度/字节	含义
h	SM9ref_MAX_LEN	签名结果的 H 部分
x	SM9ref_MAX_LEN	曲线上点的 X 坐标
y	SM9ref_MAX_LEN	曲线上点的 Y 坐标

C 语言的数据结构定义：

```
typedef struct SM9refSignature_st
{
  BYTE h[SM9ref_MAX_LEN];
  BYTE x[SM9ref_MAX_LEN];
    BYTE y[SM9ref_MAX_LEN];
} SM9Signature;
```

A.1.8 SM9 密钥封装数据结构

SM9 密钥封装数据结构符合表 A.8。

表 A.8 SM9 密钥封装数据结构

字段名称	数据长度/字节	含义
x	SM9ref_MAX_LEN	封装的交换密文曲线上点 X 坐标
y	SM9ref_MAX_LEN	封装的交换密文曲线上点 Y 坐标

C 语言的数据结构定义：

```
typedef struct SM9refKeyPackage_st
{
  BYTE x[SM9ref_MAX_LEN];
    BYTE y[SM9ref_MAX_LEN];
} SM9KeyPackage;
```

A.1.9 SM9 用户加密密钥对保护结构

SM9 用户加密密钥对保护结构符合表 A.9。

表 A.9 SM9 用户加密密钥对保护结构

字段名称	数据长度/字节	含义
version	4	版本号
ulSymmAlgID	4	对称算法标识，限定 ECB 模式
bits	4	用户加密密钥对的密钥位长

表 A.9 SM9 用户加密密钥对保护结构（续）

字段名称	数据长度/字节	含义
encryptedPriKey	SM9ref_MAX_LEN * 4	对称算法加密的密文，加密的原文为 SM9EncUserPrivateKey 结构体中除去长度的剩余部分
encMasterPubKey	SM9EncMastPublicKey 类型	KGC 的加密主公钥
tmpMasterPubKey	SM9EncMastPublicKey 类型	产生对称密钥的密钥封装数据的主公钥
userIDLen	4	用户标识长度
userID	1 024	用户标识，实际有效长度为 userIDLen
keyLen	4	对称密钥比特位数
keyPackage	SM9KeyPackage 类型	加密用户加密私钥的对称密钥的密钥封装数据

C 语言的数据结构定义：

```
typedef struct SM9refEncEnvelopedKey_st
{
  ULONG version;
    ULONG ulSymmAlgID;
  ULONG bits;
  BYTE encryptedPriKey[SM9ref_MAX_LEN * 4];
  SM9EncMastPublicKey encMastPubKey;
  SM9EncMastPublicKey tmpMastPubKey;
    ULONG userIDLen;
    BYTE userID[256];
    ULONG keyLen;
    SM9KeyPackage keyPackage;
} SM9EncEnvelopedKey;
```

A.1.10 SM9 用户签名密钥对保护结构

SM9 用户签名密钥对保护结构符合表 A.10。

表 A.10 SM9 用户签名密钥对保护结构 C 语言的数据结构定义

字段名称	数据长度/字节	含义
version	4	版本号
ulSymmAlgID	4	对称算法标识，限定 ECB 模式
bits	4	用户加密密钥对的密钥位长
encryptedPriKey	SM9ref_MAX_LEN * 2	对称算法加密的密文，加密的原文为 SM9SignUserPrivateKey 结构体中除去长度的剩余部分
signMasterPubKey	SM9SignMastPublicKey 类型	KGC 的签名主公钥
tmpMasterPubKey	SM9EncMastPublicKey 类型	产生对称密钥的密钥封装数据的主公钥
userIDLen	4	用户标识长度

表 A.10 SM9 用户签名密钥对保护结构 C 语言的数据结构定义（续）

字段名称	数据长度/字节	含义
userID	256	用户标识，实际有效长度为 userIDLen
keyLen	4	对称密钥比特位数
keyPackage	SM9KeyPackage 类型	加密用户加密私钥的对称密钥的密钥封装数据

```
typedef struct SM9refSignEnvelopedKey_st
{
  ULONG version;
    ULONG ulSymmAlgID; // 对称算法标识，限定 ECB 模式
  ULONG bits;
  BYTE encryptedPriKey [SM9ref_MAX_LEN * 4];
  SM9SignMastPublicKey signMastPubKey;
    SM9EncMastPublicKey tmpMastPubKey;
    ULONG userIDLen;
    BYTE userID[256];
    ULONG keyLen;SM9KeyPackage keyPackage;
} SM9SignEnvelopedKey;
```

A.2 SM9 算法接口函数

A.2.1 导入 SM9 主公钥

原型：

```
int SAF_ImportSM9MasterPublicKey(
    void            * hAppHandle,
    unsigned int    uiKeyIndex,
    unsigned int    uiKeyType,
    unsigned char   * pucMasterPublickKey,
    unsigned int    uiMasterPublickKeyLen);
```

描述：导入 SM9 主公钥到指定索引。

参数：

hAppHandle[in]	应用接口句柄
uiKeyIndex[in]	主公钥导入索引
uiKeyType[in]	主公钥类型 1：签名主公钥 2：加密主公钥
pucMasterPublickKey [in]	DER 编码的主公钥数据
uiMasterPublickKeyLen [in]	DER 编码的主公钥数据长度

返回值：

0	成功
非 0	失败，返回错误代码

A.2.2 导入 SM9 用户密钥对

原型：

```
int SAF_ImportSM9UserKey(
    void            * hAppHandle,
    unsigned char   * pucContainerName,
    unsigned int    uiContainerLen,
    unsigned int    uiKeyType,
    unsigned char   * pUserKey,
    unsigned int    uiUserKeyLen);
```

描述：导入 SM9 用户标识和密钥到指定容器。

参数：

hAppHandle[in]	应用接口句柄
pucContainerName[in]	密钥的容器名
uiContainerLen[in]	密钥的容器名长度
uiKeyType[in]	用户密钥保护结构类型 1:签名密钥密钥封装保护结构 2:签名密钥数字信封保护结构 3:加密密钥密钥封装保护结构 4:加密密钥数字信封保护结构
pUserKey [in]	DER 编码的用户密钥保护结构数据
uiUserKeyLen [in]	DER 编码的用户密钥保护结构数据长度

返回值：

0	成功
非 0	失败,返回错误代码

A.2.3 SM9 签名

原型：

```
int SAF_SM9Sign(
    void            * hAppHandle,
    unsigned char   * pucContainerName,
    unsigned int    uiContainerLen,
    unsigned int    uiMasterPublicKeyIndex,
    unsigned char   * pucData,
    unsigned int    uiDataLength,
    unsigned char   * pucSignData,
    unsigned int    * puiSignDataLen);
```

描述：使用 SM9 签名私钥对数据进行签名运算,SM9 运算过程应符合 GB/T 41389。

参数：

hAppHandle[in]	应用接口句柄
pucContainerName[in]	密钥的容器名

	uiContainerLen[in]	密钥的容器名长度
	uiMasterPublicKeyIndex [in]	签名主公钥的索引值
	pucData[in]	待签名数据的 SM3 杂凑值
	uiDataLength[in]	待签名数据的 SM3 杂凑值长度，值为 32
	pucSignData[out]	缓冲区指针，输出为 DER 编码的签名值数据
	puiSignDataLen[in,out]	输入为签名值数据缓冲区长度，输出为签名值数据实际长度
返回值：	0	成功
	非 0	失败，返回错误代码

A.2.4 SM9 签名验证

原型：

```
int SAF_SM9Verify(
    void            * hAppHandle,
    unsigned int    uiMasterPublicKeyIndex,
    unsigned char   * pucUserID,
    unsigned int    uiUserIDLen,
    unsigned char   * pucData,
    unsigned int    uiDataLength,
    unsigned char   * pucSignData,
    unsigned int    uiSignDataLen);
```

描述：使用 SM9 用户标识和签名主公钥对数据进行签名验证运算，SM9 运算过程应符合GB/T 41389。

参数：	hAppHandle[in]	应用接口句柄
	uiMasterPublicKeyIndex [in]	签名主公钥的索引值
	pucUserID[in]	DER 编码的用户标识数据
	uiUserIDLen[in]	DER 编码的用户标识数据长度
	pucData[in]	被签名数据的 SM3 杂凑值
	uiDataLength[in]	被签名数据的 SM3 杂凑值长度，值为 32
	pucSignData[in]	DER 编码的签名值数据
	uiSignDataLen[in]	DER 编码的签名值数据长度
返回值：	0	成功，签名验证通过
	非 0	失败，返回错误代码

A.2.5 SM9 密钥封装

原型：

```
int SAF_SM9KeyEncapsulate(
    void            * hSymmKeyObj,
    unsigned int    uiMasterPublicKeyIndex,
    unsigned char   * pucUserID,
    unsigned int    uiUserIDLen,
    unsigned int    uiKeyBits
    unsigned char   * pucKeyPackage
    unsigned int    * pulKeyPackageLen,
    void            * * phKeyHandle);
```

描述：使用 SM9 用户标识和签名主公钥进行密钥封装运算，SM9 运算过程应符合GB/T 41389。

参数：

hSymmKeyObj [in]	对称算法对象
uiMasterPublicKeyIndex [in]	加密主公钥的索引值
pucUserID[in]	DER 编码的用户标识数据
uiUserIDLen[in]	DER 编码的用户标识数据长度
uiKeyBits [in]	封装密钥的位长度
pucKeyPackage [out]	缓冲区指针，输出 DER 编码的密钥封装数据
pulKeyPackageLen [in,out]	输入为密钥封装数据缓冲区长度，输出为 DER 编码的密钥封装数据实际长度
phKeyHandle[out]	输出会话密钥的句柄

返回值：

0	成功
非 0	失败，返回错误代码

A.2.6 SM9 密钥解封装

原型：

```
int SAF_SM9KeyDecapsulate(
    void            * hSymmKeyObj,
    unsigned int    uiKeyBits
    unsigned char   * pucKeyPackage
    unsigned int    ulKeyPackageLen,
    void            * * phKeyHandle);
```

描述：使用 SM9 用户密钥进行密钥解封装运算，SM9 运算过程应符合 GB/T 41389。

参数：

hSymmKeyObj [in]	对称算法对象
pucUserID[in]	DER 编码的用户标识数据
uiUserIDLen[in]	DER 编码的用户标识数据长度
uiKeyBits [in]	封装密钥的位长度

	pucKeyPackage [in]	DER 编码的密钥封装数据
	ulKeyPackageLen [in]	DER 编码的密钥封装数据长度
	phKeyHandle[out]	输出会话密钥的句柄
返回值：	0	成功
	非 0	失败，返回错误代码

A.2.7 SM9 加密

原型：

```
int SAF_SM9Encrypt(
    void            *hAppHandle,
    unsigned int    uiMasterPublicKeyIndex,
    unsigned char   *pucUserID,
    unsigned int    uiUserIDLen,
    unsigned int    ulAlgID,
    unsigned char   *pIV,
    unsigned int    ulIVLen,
    unsigned char   *pucData,
    unsigned int    uiDataLength,
    unsigned char   *pucEncData,
    unsigned int    *pulEncDataLen);
```

描述：使用 SM9 用户标识对数据进行加密运算，SM9 运算过程应符合 GB/T 41389。

参数：	hAppHandle[in]	应用接口句柄
	uiMasterKeyIndex[in]	加密主公钥的索引值
	pucUserID[in]	DER 编码的用户标识数据
	uiUserIDLen[in]	DER 编码的用户标识数据长度
	ulAlgID[in]	SM9 加密算法标识
	pIV[in]	IV 数据，若为 NULL，则由密码设备产生
	ulIVLen[in]	IV 数据长度
	pucData[in]	待加密数据
	uiDataLength[in]	待加密数据长度
	pucEncData [out]	缓冲区指针，用于存放输出 DER 编码的密文数据
	pulEncDataLen[in,out]	输入为缓冲区长度，输出为 DER 编码的密文数据实际长度
返回值：	0	成功
	非 0	失败，返回错误代码

A.2.8 SM9 解密

原型：

```
int SAF_SM9Decrypt(
    void            * hAppHandle,
    unsigned char   * pucContainerName,
    unsigned int    uiContainerLen,
    unsigned char   * pucEncData,
    unsigned int    ulEncDataLen,
    unsigned char   * pucData,
    unsigned int    * puiDataLength);
```

描述：使用 SM9 用户加密密钥对(包括用户私钥和用户标识)对数据进行解密运算，SM9 运算过程应符合 GB/T 41389。

参数：

hAppHandle[in]	应用接口句柄
pucContainerName[in]	密钥的容器名
uiContainerLen[in]	密钥的容器名长度
pucEncData [in]	DER 编码的密文数据
ulEncDataLen[in]	DER 编码的密文数据长度
pucData[out]	缓冲区指针，用于存放解密数据
puiDataLength[in,out]	输入为缓冲区长度，输出为解密数据实际长度

返回值：

0	成功
非 0	失败，返回错误代码

A.2.9 生成密钥协商参数并输出

原型：

```
int SAF_SM9GenerateAgreementData (
        void * hSymmKeyObj,
        unsigned int uiMasterPublicKeyIndex,
        unsigned int uiKeyBits,
        unsigned char * pucResponsorID,
        unsigned int uiResponsorIDLen,
        unsigned char * pucSponsorTmpPublicKey,
        unsigned int * puiSponsorTmpPublicKeyLen,
        void * * phAgreementHandle);
```

描述：使用 SM9 密钥协商算法，为计算会话密钥而产生协商参数，同时返回发起方临时 SM9 密钥对的公钥及协商句柄，SM9 运算过程应符合 GB/T 41389。

参数：

hSymmKeyObj [in]	对称算法对象
uiMasterKeyIndex[in]	加密主公钥的索引值

uiKeyBits[in]	要求协商的密钥长度
pucResponsorID [in]	DER 编码的响应方标识
uiResponsorIDLen [in]	DER 编码的响应方标识长度
pucSponsorTmpPublicKey [out]	缓冲区指针，用于保存返回的 DER 编码的发起方临时公钥
puiSponsorTmpPublicKeyLen [in,out]	输入为缓冲区长度，输出为返回的 DER 编码的发起方临时公钥实际长度
phAgreementHandle[out]	返回的密钥协商句柄

返回值：

0	成功
非 0	失败，返回错误代码

备注：为协商会话密钥，协商的发起方应首先调用本函数计算会话密钥。

A.2.10 计算会话密钥

原型：

```
int SAF_ SM9GenerateKey (
    void            * hAgreementHandle,
    unsigned char   * pucResponseTmpPublicKey,
    unsigned int    uiResponseTmpPublicKeyLen,
    void            * * phKeyHandle);
```

描述：使用 SM9 密钥协商算法，使用自身协商句柄和响应方的协商参数计算会话密钥，同时返回会话密钥句柄，SM9 运算过程应符合 GB/T 41389。

参数：

hAgreementHandle [in]	密钥协商句柄
pucResponseTmpPublicKey[in]	外部输入的 DER 编码的响应方临时公钥
uiResponseTmpPublicKeyLen[in]	外部输入的 DER 编码的响应方临时公钥长度
phKeyHandle[out]	返回的密钥句柄

返回值：

0	成功
非 0	失败，返回错误代码

备注：协商的发起方获得响应方的协商参数后调用本函数，计算会话密钥。

A.2.11 产生协商数据并计算会话密钥

原型：

```
int SAF_ SM9GenerateAgreementDataAndKey (
    void            * hSymmKeyObj,
    unsigned int    uiMasterPublicKeyIndex,
    unsigned int    uiKeyBits,
    unsigned char   * pucSponsorID,
```

```
unsigned int      uiSponsorIDLength,
unsigned char     * pucSponsorTmpPublicKey,
unsigned int      * puiSponsorTmpPublicKeyLen,
unsigned char     * pucResponseTmpPublicKey,
unsigned int      uiResponseTmpPublicKeyLen,
void              * * phKeyHandle);
```

描述：　使用 SM9 密钥协商算法，产生协商参数并计算会话密钥，同时返回产生的协商参数和密钥句柄，SM9 运算过程应符合 GB/T 41389。

参数：

hSymmKeyObj [in]	对称算法对象
uiMasterKeyIndex[in]	加密主公钥的索引值
uiKeyBits[in]	要求协商的密钥长度
pucSponsorID [in]	DER 编码的发起方标识
uiSponsorIDLength [in]	DER 编码的发起方标识长度
pucSponsorTmpPublicKey [in]	外部输入的 DER 编码的发起方临时公钥
puiSponsorTmpPublicKeyLen [in]	外部输入的 DER 编码的发起方临时公钥长度
pucResponseTmpPublicKey[in]	缓冲区指针，用于保存返回的 DER 编码的响应方临时公钥
uiResponseTmpPublicKeyLen[in,out]	输入为缓冲区长度，输出为返回的 DER 编码的响应方临时公钥实际长度
phKeyHandle[out]	返回的密钥句柄

返回值：

0	成功
非 0	失败，返回错误代码

备注：　本函数由响应方调用。

附 录 B
（规范性）
密码服务接口错误代码定义

密码服务接口错误代码的定义见表 B.1。

表 B.1 错误代码定义

错误代码名称	错误代码预定义值	错误代码说明
SAR_OK	0	成功
SAR_UnknownErr	0X02000001	异常错误
SAR_NotSupportYetErr	0X02000002	不支持的服务
SAR_FileErr	0X02000003	文件操作错误
SAR_ProviderTypeErr	0X02000004	服务提供者参数类型错误
SAR_LoadProviderErr	0X02000005	导入服务提供者接口错误
SAR_LoadDevMngApiErr	0X02000006	导入设备管理接口错误
SAR_AlgoTypeErr	0X02000007	算法类型错误
SAR_NameLenErr	0X02000008	名称长度错误
SAR_KeyUsageErr	0X02000009	密钥用途错误
SAR_ModulusLenErr	0X02000010	模的长度错误
SAR_NotInitializeErr	0X02000011	未初始化
SAR_ObjErr	0X02000012	对象错误
SAR_MemoryErr	0X02000100	内存错误
SAR_TimeoutErr	0X02000101	超时
SAR_IndataLenErr	0X02000200	输入数据长度错误
SAR_IndataErr	0X02000201	输入数据错误
SAR_GenRandErr	0X02000300	生成随机数错误
SAR_HashObjErr	0X02000301	HASH 对象错
SAR_HashErr	0X02000302	HASH 运算错误
SAR_GenRsaKeyErr	0X02000303	产生 RSA 密钥错
SAR_RsaModulusLenErr	0X02000304	RSA 密钥模长错误
SAR_CspImprtPubKeyErr	0X02000305	CSP 服务导入公钥错误
SAR_RsaEncErr	0X02000306	RSA 加密错误
SAR_RSGDecErr	0X02000307	RSA 解密错误
SAR_HashNotEqualErr	0X02000308	HASH 值不相等
SAR_KeyNotFountErr	0X02000309	密钥未发现
SAR_CertNotFountErr	0X02000310	证书未发现
SAR_NotExportErr	0X02000311	对象未导出

表 B.1 错误代码定义（续）

错误代码名称	错误代码预定义值	错误代码说明
SAR_CertNotYetValidErr	0X02000317	证书未生效
SAR_CertHasExpiredErr	0X02000318	证书已过期
SAR_CertVerifyErr	0X02000319	证书验证错误
SAR_CertEncodeErr	0X02000320	证书编码错误
SAR_DecryptPadErr	0X02000400	解密时做补丁错误
SAR_MacLenErr	0X02000401	MAC 长度错误
SAR_KeyInfoTypeErr	0X02000402	密钥类型错误
SAR_NotLogin	0X02000403	没有进行登录认证
SAR_KeyCompromise	0X02000404	证书被吊销原因：密钥泄露
SAR_CACompromise	0X02000405	证书被吊销原因：CA 泄露
SAR_AffiliationChanged	0X02000406	证书被吊销原因：隶属关系已修改
SAR_Superseded	0X02000407	证书被吊销原因：被替代
SAR_CessationOfOperation	0X02000408	证书被吊销原因：被终止
SAR_CertificateHold	0X02000409	证书被吊销原因：证书挂起
SAR_PrivilegeWithdrawn	0X02000410	证书被吊销原因：证书特权被撤销
SAR_Sm9MasterKeyErr	0X02000500	SM9 主密钥错误
SAR_Sm9UserKeyErr	0X02000501	SM9 用户密钥错误
SAR_Sm9VerifyErr	0X02000502	SM9 签名验证失败
SAR_Sm9KeyExchangeErr	0X02000503	SM9 密钥协商失败
SAR_Sm9DecapsulateErr	0X02000504	SM9 密钥解封装失败
SAR_Sm9DecErr	0X02000505	SM9 解密失败

参 考 文 献

[1] GB/T 15851.3 信息技术 安全技术 带消息恢复的数字签名方案 第3部分:基于离散对数的机制

[2] GB/T 17903.1—1999 信息技术 安全技术 抗抵赖 第1部分:概述

[3] GB/T 17903.2—1999 信息技术 安全技术 抗抵赖 第2部分:使用对称技术的机制

[4] GB/T 17903.3—1999 信息技术 安全技术 抗抵赖 第3部分:使用非对称技术的机制

[5] GB/T 17964.1—2000 信息技术 安全技术 加密算法 第1部分:概述

[6] GB/T 17964.2—2000 信息技术 安全技术 加密算法 第2部分:非对称加密

[7] GB/T 179643—2000 信息技术 安全技术 加密算法 第3部分:对称加密

[8] GB/T 18238.1—2000 信息技术 安全技术 散列函数 第1部分:概述

[9] GB/T 18238.2—2002 信息技术 安全技术 散列函数 第2部分:采用n位块密码的散列函数

[10] GB/T 18238.3—2002 信息技术 安全技术 散列函数 第3部分:专用散列函数

[11] GB/T 19713—2005 信息技术 安全技术 公钥基础设施 在线证书状态协议

[12] GB/T 19771—2005 信息技术 安全技术 公钥基础设施 PKI组件最小互操作规范

[13] ISO/IEC 8825-1:2021 Information technology—ASN.1 encoding rules—Part 1:Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)

[14] IETF RFC 1777 Lightweight Directory Access Protocol

[15] IETF RFC 2459 Internet X.509 Public Key Infrastructure Certificate and CRL Profile

[16] IETF RFC 2560 X.509 Internet Public Key Infrastructure Online Certificate Status Protocol

[17] IETF RFC 2587 Internet X.509 Public Key Infrastructure LDAPv2 Schema

[18] IETF RFC 3647 Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework

[19] IETF RFC 4648 The Base16, Base32, and Base64 Data Encodings

[20] PKCS #1 RSA Encryption Version 1.5

[21] PKCS #7 Cryptographic Message Syntax Version 1.5

[22] PKCS #11 Cryptographic Token Interface Standard

[23] RSA Security Public-Key Cryptography Standards (PKCS)