

目录

目录	1
SKF使用说明及开发问题集锦	3
文件清单	3
动态库版本1	3
动态库版本2	3
静态库版本	3
说明	4
使用须知	4
提醒	4
开启SKF国密库日志	4
TF卡设备使用须知（非TF卡设备忽略）	4
默认密钥	5
默认设备认证码	5
DEFAULT应用默认PIN码	5
接口使用	5
设备认证	5
默认认证码	5
对应接口	5
设备认证流程	5
使用须知	6
参考代码	6
导入ECC密钥对	8
相关接口	8
权限要求	8
参考代码	8
导入RSA密钥对	10
相关接口	11
权限要求	11
参考代码	11
非对称解密	17
相关接口	17
参考代码	19
说明	22
证书导入	23
相关接口	23

导入要求	23
参考代码	23
签名验签	25
相关接口	25
权限要求	27
参考代码	27
对称算法运算	30
相关接口	30
使用须知	32
参考代码	32
杂凑算法运算	35
相关接口	35
参考代码	36
创建应用	39
对应接口	40
权限要求	40
参考代码	40
创建容器	41
相关接口	41
权限要求	41
参考代码	41
固件升级	43
说明	43
参考代码	44
SKF使用demo	46
开发问题集锦	50
SKF_EnumDev()枚举设备时返回0x0A000023错误，无法枚举到设备怎么办？	50
怎么创建应用？	50
怎么删除应用？	51
怎么创建容器？	51
怎么删除容器？	51
创建应用失败？	51
删除应用失败？	51
SKF调用流程是怎么样的？	51
SKF_SetSymmKey返回SKF_NO_ROOM怎么办？	51
怎么开启SKF国密库日志？	51

SKF使用说明及开发问题集锦

文件清单

动态库版本1

平台	文件清单
<u>windows</u>	1. SKF_***_no_log.dll 2. SKF_***_with_log.dll
<u>linux/arm/android</u>	1. libskf_***_no_log.so 2. libskf_***_with_log.so

动态库版本2

平台	文件清单
<u>windows</u>	1. SKF.dll 2. operation.dll.no.log/operation.dll.have.log
<u>linux/arm/android</u>	1. libskf.so 2. liboperation.so.no.log/liboperation.so.no.log

windows平台：需将opeation.dll.[no/have].log修改为operation.dll(liboperation.so)后使用。

linux/arm/android平台：需将[lib]opeation.[dll/so].[no/have].log修改为operation.dll(liboperation.so)后使用。

如果使用*.have.log版本，SKF会在SKF.dll/libskf.so所在目录生成skflog.txt日志文件。

静态库版本

平台	文件清单	系统依赖库	链接顺序
<u>windows</u>	1. cdrv.lib 2. clog.lib 3. libcore.lib 4. libcutils.lib 5. SKF_***_no_log.lib 6. SKF_***_with_log.lib	1. ws2_32.lib	无
<u>linux/arm/android</u>	1. libcdrv.a 2. libclog.a 3. libcore.a 4. libcutils.a 5. libskf_***_no_log.a 6. libskf_***_with_log.a	1. pthread 2. dl	-lskf -lcore -lcdrv -lclog -lcutils -lpthread -ldl

说明

- SKF无论动态库版本还是静态库版本，都为带日志和不带日志版本，其中：
 - `***_no_log.***`为不带日志版本；
 - `***_with_log.***`为带日志版本，日志输出路径详见《使用须知-开启SKF国密库日志》；
- 当使用静态库版本时，由于部分依赖系统库，因此在使用时一定要注意。`linux/arm/android`等平台还需要注意静态库的链接顺序。

使用须知

提醒

使用SKF国密库时，有以下几点需特别注意：

- 如在Linux、Arm、Android等平台，必须在root权限下运行；
- 如Linux、Arm、Android等平台提示无法找到`libskf.so`文件，可通过以下方法解决：
 - 将`libskf.so`文件拷贝至`/usr/lib/`等系统目录；
 - 执行 `export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH` 命令；
- 当android平台开发APP时，必须先调用V_SetAppPath0接口设置APP路径，如“Android/data/XXXX”，其中XXXX是APP名称。

开启SKF国密库日志

如果使用`***_with_log.***`带日志的SKF国密库，产生日志的方法如下：

◦ windows平台

- 在`C:\log`目录下创建 `skf.txt` 文件；
- 如果 `skf.txt` 文件中填写了日志文件路径，则日志输出到自定义路径；
- 如果 `skf.txt` 文件为空，则默认输出到`C:\log\skflog.txt`；

◦ linux平台

- 在`/tmp`目录下创建 `skf.txt` 文件；
- 如果 `skf.txt` 文件中填写了日志文件路径，则日志输出到自定义路径；
- 如果 `skf.txt` 文件为空，则默认输出到`/tmp/skflog.txt`；

◦ android平台

- 在可执行程序运行的当前目录下创建 `skf.txt` 文件；
- 如果 `skf.txt` 文件中填写了日志文件路径，则日志输出到自定义路径；
- 如果 `skf.txt` 文件为空，则默认输出到可执行程序运行的当前目录下`skflog.txt`；

TF卡设备使用须知（非TF卡设备忽略）

在Linux、Arm、Android等平台下使用SKF国密库前，必须已通过 `mount` 命令将TF卡挂载上盘，否则会导致无法枚举到设备。

```
mount -t vfat /dev/sd1 /mnt/media_rw/
```

其中：

- `/dev/sd1` 为TF卡设备节点；
- `/mnt/media_rw/` 为任意挂载路径，该路径必须已经存在，且不建议该路径包含中文字符；

默认密钥

默认设备认证码

设备初始化默认设备认证码是：**"C*CORE SYS @ SZ"** 或 **"1234567812345678"**

说明：

- 1. 根据不同客户需求，设备出厂默认设备认证码主要有以上两种，用户可一一尝试；
- 2. 如果客户定制的其他默认设备认证码，请不要使用以上两个认证码；
- 3. **设备认证码必须是16字节长度的数组**；
- 4. 如使用"C*CORE SYS @ SZ"外部认证码，**请注意最后还有一个空格符**；

DEFAULT应用默认PIN码

默认用户PIN码：**"111111"**

默认管理员PIN码：**"111111"**

接口使用

设备认证

默认认证码

详见**默认设备认证码**章节。

对应接口

7.2.3 设备认证

函数原型	ULONG DEVAPI SKF_DevAuth (DEVHANDLE hDev, BYTE *pbAuthData, ULONG ulLen)	
功能描述	设备认证是设备对应用程序的认证。认证过程参见 8.2.3。	
参数	hDev	[IN] 连接时返回的设备句柄。
	pbAuthData	[IN] 认证数据。
	ulLen	[IN] 认证数据的长度。
返回值	SAR_OK:	成功。
	其他:	错误码。

设备认证流程

8.2.3 设备认证

必须通过设备认证后才能在设备内创建和删除应用。

设备认证使用分组密码算法和设备认证密钥进行。认证的流程如下：

34

- 1) 被认证方调用 SKF_GenRandom 函数从设备获取 8 字节随机数 RND，并用 0x00 将其填充至密码算法的分块长度，组成数据块 D0；
- 2) 被认证方对 D0 加密，得到加密结果 D1，并调用 SKF_DevAuth()，将 D1 发送至设备；
- 3) 设备收到 D1 后，验证 D1 是否正确。正确则通过设备认证，否则设备认证失败。

使用须知

- 设备认证主要用于创建或删除应用。
- 设备认证完成后需马上调用创建/删除应用接口，中间不能穿插其他任何接口调用，否则会丢失设备认证权限，导致创建应用时权限不足。

参考代码

```
int dev_auth(core_t *core, settings_t *settings, menu_t *menu)
{
    int ret = 0;
    int algo = 0;
    int auth_len = 0;
    char *key = settings->get_str(settings, "key", "C*CORE SYS @ SZ ");
    api_t *api = core->get_api(core);
    api_symbol_t *symbol = api->get_symbol(api);
    api_instance_t *instance = api->get_instance(api);
    unsigned char auth_key[128] = {0};
    unsigned char auth_data[64] = {0};
    unsigned char random_data[16] = {0};
    mclog_api mclog = symbol->mclog;
    SKF_FUNCLIST *skf = instance->skf;
    DEVHANDLE hdev = (DEVHANDLE)NULL;
    HANDLE hkey = (HANDLE)NULL;
    BLOCKCIPHERPARAM bp;
    DEVINFO info;

    /**
     * get device handle
     */
    // LOG_INFO(symbol->mclog, -1, "%s perform start\n", algo_name);
    hdev = (DEVHANDLE)instance->values->get(instance->values, SKF_DEVICE_HANDLE);
    if (!hdev) {
        ret = SAR_INVALIDHANDLEERR;
        LOG_ERROR(symbol->mclog, -1, "invalid device handle\n");
        goto end;
    }
}
```

```
/**
 * get device info
 */
memset(&info, 0, sizeof(info));
ret = skf->SKF_GetDevInfo(hdev, &info);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_GetDevInfo() failed: %#x\n", ret);
    goto end;
}

/**
 * gen random_data
 */
ret = skf->SKF_GenRandom(hdev, random_data, 8);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_GenRandom() failed: %#x\n", ret);
    goto end;
}

/**
 * set sym key
 */
algo = info.DevAuthAlgId;
memcpy(auth_key, key, 16);
memset(auth_key + 16, 0, sizeof(auth_key) - 16);
ret = skf->SKF_SetSymmKey(hdev, (void *)auth_key, algo, &hkey);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_SetSymmKey() failed: %#x\n", ret);
    goto end;
}

/**
 * sym enc init
 */
memset(&bp, 0, sizeof(bp));
ret = skf->SKF_EncryptInit(hkey, bp);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_EncryptInit() failed: %#x\n", ret);
    goto end;
}

/**
 * encrtpt device auth
 */
auth_len = sizeof(auth_data);
ret = skf->SKF_Encrypt(hkey, random_data, sizeof(random_data), auth_data, (void *)&auth_len);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_Encrypt() failed: %#x\n", ret);
    goto end;
}

/**
 * device auth
 */
ret = skf->SKF_DevAuth(hdev, auth_data, auth_len);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_DevAuth() failed: %#x\n", ret);
    goto end;
}
LOG_INFO(mclog, -1, "device auth [g]succ[n]\n");
```

```

end:
    skf->SKF_CloseHandle(hkey);
    return ret;
}

```

导入ECC密钥对

相关接口

7.6.12 导入 ECC 加密密钥对

函数原型 `ULONG DEVAPI SKF_ImportECCKeyPair (`
 `HCONTAINER hContainer, PENVELOPEDKEYBLOB pEnvelopedKeyBlob)`

功能描述 导入 ECC 公私钥对。

参数

<code>hContainer</code>	[IN] 密钥容器句柄。
<code>pEnvelopedKeyBlob</code>	[IN] 受保护的加密密钥对。

返回值

<code>SAR_OK</code> :	成功。
其他:	错误码。

备注 权限要求：需要用户权限。

权限要求

▲ 需要**用户权限**，设备权限提升使用 `SKF_VerifyPIN` 接口

参考代码

```

int sm2_import_keypair(core_t *core, settings_t *settings, menu_t *menu)
{
    int ret = 0;
    int retry = 0;
    int ctn_type = 0;
    char *pin = "111111";
    char *app = "DEFAULT";
    char *ctn = "DEFAULT";
    api_t *api = core->get_api(core);
    api_symbol_t *symbol = api->get_symbol(api);
    api_instance_t *instance = api->get_instance(api);
    menu_controller_t *controller = menu->get_controller(menu);
    settings_t *property = controller->get_property(controller);
    mclog_api mclog = symbol->mclog;
    ECCPUBLICKEYBLOB *psm2pubkey = NULL;
    PENVELOPEDKEYBLOB env = NULL;
    SKF_FUNC_LIST *skf = instance->skf;
    DEVHANDLE hdev = (DEVHANDLE) NULL;
    HAPPLICATION happ = (HAPPLICATION) NULL;
    HCONTAINER hctn = (HCONTAINER) NULL;
    HANDLE hkey = (HANDLE) NULL;
    BLOCKCIPHERPARAM bp;
    unsigned int pubkeylen = 0;
    unsigned char pubkey[512] = {0};
    unsigned char enc_data[512] = {0};
    unsigned char keypair[] = {

```

```

//x
0x19, 0x79, 0x5d, 0xf7, 0x01, 0xf3, 0x9d, 0x1f, 0xb2, 0x20, 0xc4, 0x5f, 0xa7, 0xfa, 0x4e, 0xbf,
0xad, 0xd1, 0x70, 0x25, 0x37, 0xb9, 0x46, 0xcd, 0x3d, 0x48, 0x04, 0xb3, 0x7f, 0xbc, 0x3e, 0xa5,
//y
0x2b, 0x2c, 0xee, 0xd6, 0xcc, 0x04, 0x2b, 0x5b, 0xbb, 0x56, 0x8d, 0xed, 0x3b, 0x36, 0x73, 0xf2,
0x88, 0xe1, 0x9c, 0xc4, 0x9a, 0xe3, 0xc3, 0x50, 0xd2, 0xb8, 0x09, 0x03, 0xd8, 0x6d, 0x91, 0x2c,
//d
0x3f, 0x91, 0x68, 0xe8, 0x6d, 0x2a, 0xac, 0xaa, 0x2c, 0x81, 0xd8, 0xba, 0x24, 0x9b, 0xc9, 0x5a,
0x60, 0xe0, 0x47, 0x50, 0xa2, 0xee, 0xaa, 0x63, 0x26, 0x2b, 0x54, 0xc4, 0x75, 0x51, 0xb8, 0xdc
};
unsigned char key[32] = {0x47, 0x50, 0x42, 0x02, 0x20, 0x3F, 0xE1, 0x92, 0x66, 0x2A, 0xCB, 0xD2, 0x9D, 0,
0, 0};
unsigned int enclen = 0;

/**
 * show info
 */
LOG_INFO(symbol->mclog, -1, "\n%p", "sm2 import keypair",
          "app", "%s", app,
          "ctn", "%s", ctn,
          NULL);

/**
 * get device handle
 */
// LOG_INFO(symbol->mclog, -1, "%s perform start\n", algo_name);
hdev = (DEVHANDLE)instance->values->get(instance->values, SKF_DEVICE_HANDLE);
if (!hdev) {
    ret = SAR_INVALIDHANDLEERR;
    LOG_ERROR(symbol->mclog, -1, "invalid device handle\n");
    goto end;
}

/**
 * open app
 */
ret = skf->SKF_OpenApplication(hdev, app, &happ);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_OpenApplication() failed: %#x\n", ret);
    goto end;
}

/**
 * verify pin
 */
ret = skf->SKF_VerifyPIN(happ, USER_TYPE, pin, (void *)&retry);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_VerifyPIN() failed: %#x\n", ret);
    goto end;
}

/**
 * open container
 */
ret = skf->SKF_OpenContainer(happ, ctn, &hctn);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_OpenContainer() failed: %#x\n", ret);
    goto end;
}

```

```

/**
 * check keypair
 */
pubkeylen = sizeof(pubkey);
psm2pubkey = (ECCPUBLICKEYBLOB *)pubkey;
ret = skf->SKF_ExportPublicKey(hctn, 0, (void *)pubkey, &pubkeylen);
if (!ret && ctn_type == CTNF_ECC && psm2pubkey->BitLen == 256) {
    LOG_INFO(mclog, -1, "sm2 keypair already ready imported\n");
    goto end;
}

/**
 * generate keypair
 */
ret = skf->SKF_ExportPublicKey(hctn, 1, (void *)pubkey, &pubkeylen);
if (ret) {
    ret = skf->SKF_GenECCKeyPair(hctn, SGD_SM2_1, (void *)pubkey);
    if (ret) goto end;
}

/**
 * encrypt prikey
 */
env = (void *)enc_data;
env->Version = 1;
env->ulBits = 256;
env->PubKey.BitLen = 256;
env->ulSymmAlgID = SGD_SM1_ECB;
memcpy(env->PubKey.XCoordinate + 32, keypair, 32);
memcpy(env->PubKey.YCoordinate + 32, keypair + 32, 32);
ret = skf->SKF_ExtECCEncrypt(hdev, (void *)pubkey, key, 32, &env->ECCCipherBlob);
if (ret) goto end;

ret = skf->SKF_SetSymmKey(hdev, key, SGD_SM1_ECB, &hkey);
if (ret) goto end;

memset(&bp, 0, sizeof(bp));
ret = skf->SKF_EncryptInit(hkey, bp);
if (ret) goto end;

enclen = 128;
ret = skf->SKF_Encrypt(hkey, keypair + 64, 32, env->cbEncryptedPriKey + 32, (void *)&enclen);
if (ret) goto end;

ret = skf->SKF_ImportECCKeyPair(hctn, env);
if (ret) goto end;
LOG_INFO(mclog, -1, "sm2 import keypair [g]succ[n]\n");

end:
if (hkey) skf->SKF_CloseHandle(hkey);
if (hctn) skf->SKF_CloseContainer(hctn);
if (happ) skf->SKF_CloseApplication(happ);
return ret;
}

```

导入RSA密钥对

相关接口

7.6.5 导入 RSA 加密密钥对

函数原型	ULONG DEVAPI SKF_ImportRSAKeyPair (HCONTAINER hContainer, ULONG ulSymAlgId, BYTE *pbWrappedKey, ULONG ulWrappedKeyLen, BYTE *pbEncryptedData, ULONG ulEncryptedDataLen)		
功能描述	导入 RSA 加密公私钥对。		
参数	hContainer	[IN]	容器句柄。
	ulSymAlgId	[IN]	对称算法密钥标识。
	pbWrappedKey	[IN]	使用该容器内签名公钥保护的对称算法密钥。
	ulWrappedKeyLen	[IN]	保护的对称算法密钥长度。
	pbEncryptedData	[IN]	对称算法密钥保护的 RSA 加密私钥。私钥的格式遵循 PKCS #1 v2.1: RSA Cryptography Standard 中的私钥格式定义。
	ulEncryptedDataLen	[IN]	对称算法密钥保护的 RSA 加密公私钥对长度。
返回值	SAR_OK:	成功。	
	其他:	错误码。	
备注	权限要求: 需要用户权限。		

权限要求

▲ 需要用户权限，设备权限提升使用 SKF_VerifyPIN 接口

参考代码

```
static int rsa_import_keypair(core_t *core, settings_t *settings, menu_t *menu)
{
    int ret = 0;
    int type = 0;
    int retry = 0;
    char *pin = "111111";
    char *app = "DEFAULT";
    char *ctn = "DEFAULT";
    int bits = 1024;
    int is_pkcs1_keypair = 0;
    api_t *api = core->get_api(core);
    linked_list_t *items = NULL;
    api_symbol_t *symbol = api->get_symbol(api);
    api_instance_t *instance = api->get_instance(api);
    menu_controllor_t *controllor = menu->get_controllor(menu);
    settings_t *property = controllor->get_property(controllor);
    mclog_api mclog = symbol->mclog;
    SKF_FUNC_LIST *skf = instance->skf;
    RSAPUBLICKEYBLOB *prsapubkey = NULL;
    DEVHANDLE hdev = (DEVHANDLE)NULL;
    HAPPLICATION happ = (HAPPLICATION)NULL;
    HCONTAINER hctn = (HCONTAINER)NULL;
    HANDLE hkey = (HANDLE)NULL;
    BLOCKCIPHERPARAM bp;
    unsigned char pubkey[2048] = {0};
    unsigned char sessionkey[2048] = {0};
    unsigned char enc_data[2048] = {0};
}
```

```

// unsigned int keylen = sizeof(pubkey);
unsigned int sessionkeylen = sizeof(sessionkey);
unsigned int enclen = sizeof(enc_data);
u8 keypair_1024[] = {
    //modulus
    0xd4,0x02,0xde,0x00,0x43,0xbb,0x6c,0x89,0x07,0x30,0x56,0x18,0x9b,0x1e,0x7d,
    0xb6,0x3e,0x57,0x0c,0x70,0x88,0x5d,0xdb,0xdf,0x84,0x1d,0x78,0x20,0xa5,0x82,
    0xfb,0x68,0xbb,0xde,0xe0,0x44,0xd9,0x89,0x8d,0xed,0x74,0x32,0x10,0x00,0xe5,
    0x63,0x2e,0x8e,0x0a,0xeb,0xed,0xc6,0x55,0x78,0xef,0xe6,0x8a,0xcb,0xf9,0xf8,
    0x50,0x5d,0xe7,0xbb,0xc2,0x99,0xf4,0xb7,0x82,0x72,0xfa,0xc3,0xf7,0x6c,0x61,
    0xf8,0x8c,0x9d,0x81,0x57,0x89,0xc6,0xb9,0xa2,0xd2,0x82,0x1c,0x3d,0x25,0xb9,
    0x44,0xc9,0x82,0x8c,0x4b,0xe2,0x9c,0xd5,0x5a,0x59,0xa7,0xa6,0xfa,0xa5,0xfa,
    0x08,0x7a,0x58,0xa4,0x06,0x11,0xc3,0xa2,0x7f,0xc6,0x61,0x33,0xe0,0xc2,0x8a,
    0x0e,0xe4,0x23,0x5c,0xbb,0x84,0x5f,0x8b,
    //p
    0xfe,0x0e,0x81,0xb0,0xc3,0x30,0xb6,0x18,0xa5,0x53,0x5f,0xeb,0x08,0xde,
    0x82,0xb4,0x57,0xf3,0xae,0x2c,0x7d,0x70,0x76,0xe5,0x37,0xc0,0xe7,0xf9,0xe6,
    0x0d,0xa0,0xc6,0x74,0x6e,0xbe,0x75,0x70,0x96,0x83,0x3e,0x17,0x79,0x1a,0x6c,
    0x2d,0xe2,0xf2,0x10,0x95,0xd2,0x79,0x81,0xea,0x76,0xbe,0xe4,0x00,0x9d,0x14,
    0xa4,0xa0,0xb0,0xf7,0x15,
    //q
    0xd5,0xa2,0x06,0xf7,0xa9,0xb8,0xed,0xd7,0x1d,0xd8,0x0e,0xac,0xc6,0x61,
    0x94,0x54,0xbb,0x35,0xb1,0x15,0xfc,0xed,0x8f,0xf3,0xbf,0xe1,0xb4,0xd5,0xde,
    0x04,0xcc,0xfc,0x89,0x7d,0xe2,0x5e,0x32,0x96,0x52,0xb2,0x50,0x9f,0x4b,0x00,
    0x37,0x64,0x7a,0x07,0x00,0x62,0x75,0xb6,0x2f,0x4b,0xf2,0xcd,0xc1,0x6d,0x3d,
    0x8e,0x5b,0xd9,0xa4,0x1f,
    //dp
    0x2e,0x0c,0xa5,0x17,0x4c,0x19,0xfd,0x37,0xb4,0x67,0xcb,0x60,0x07,0xc8,0x85,
    0x3a,0x79,0x22,0xb3,0x34,0x5f,0x3c,0x4e,0x60,0xb7,0xdd,0x60,0x6e,0xdc,0x73,
    0x25,0xec,0x32,0xd8,0x8b,0xef,0x2f,0x8c,0x28,0x97,0xcd,0x9b,0x66,0xdd,0xaa,
    0xe7,0x92,0xe6,0xcc,0xb1,0x4c,0xd2,0xee,0x93,0xbd,0x80,0x08,0x58,0x70,0x90,
    0x72,0x8c,0x01,0x0d,
    //dq
    0x70,0x01,0x1b,0x16,0x0d,0xed,0xdf,0x04,0xc1,0xa8,0xdd,0x48,0xc8,0x59,0xb0,
    0xa3,0x1b,0xe3,0xf2,0x8c,0x4c,0xa7,0x60,0xa9,0xb3,0x18,0x6a,0xef,0x16,0x0f,
    0xfe,0x49,0x08,0xec,0xef,0x19,0xe3,0xfb,0xdc,0x2f,0x91,0x05,0x73,0x15,0x11,
    0xf5,0xa3,0xe4,0xb7,0xd6,0xe7,0x50,0x35,0x4b,0xe4,0x68,0xeb,0xd8,0x92,0x45,
    0x68,0xf2,0x9a,0x91,
    //coef
    0xb1,0x04,0x4a,0x7e,0x1d,0x2b,0x45,0xbb,0xa6,0x87,0x2f,0x4b,0x28,0xf5,
    0xaf,0x51,0x55,0xe5,0x6b,0x99,0x27,0x53,0x86,0xed,0x00,0x76,0x22,0x98,0xcd,
    0x8b,0xa5,0xa2,0xc8,0x1a,0xf8,0xa1,0x30,0xb9,0x97,0x72,0x0f,0x74,0xf5,0xdb,
    0x43,0x04,0x65,0x71,0x1e,0x95,0xb8,0x3e,0x62,0xee,0x0d,0xc0,0x31,0x3e,0xc5,
    0xc9,0xb0,0x0f,0xec,0x93,
    //d
    0x06,0x74,0xff,0xe5,0xea,0x2a,0x2f,0x28,0x0d,0x3b,0xff,0xa3,0x48,0x36,0x29,
    0x85,0xff,0x68,0x0f,0x4c,0xee,0x1a,0x4d,0xb4,0x05,0x06,0x10,0xbe,0x8d,0xcb,
    0xeb,0x74,0x49,0x24,0x84,0x98,0x14,0x38,0x08,0x46,0x8f,0x1f,0x67,0x35,0x5f,
    0xc7,0x21,0x87,0xd9,0xf5,0x4d,0x8c,0x98,0xd5,0xd2,0x18,0xa7,0x69,0x86,0xf7,
    0x75,0x1b,0x27,0x0f,0x96,0xa3,0x2d,0x53,0x58,0x48,0x92,0x88,0x6b,0xfe,0xfe,
    0x6a,0x97,0xd9,0xdc,0x7e,0xff,0x45,0xd3,0x96,0xd3,0x98,0x65,0xbe,0x63,0x58,
    0xf5,0x4c,0xb9,0xec,0x64,0x43,0x7a,0x47,0x84,0x3e,0xfc,0xab,0xeb,0xaa,0x4c,
    0x03,0x8f,0xdf,0x92,0x19,0xa4,0x5a,0x15,0x15,0xc2,0x6f,0xd2,0x07,0xd0,0x63,
    0x86,0x13,0xac,0x37,0x25,0x45,0xde,0x21
};
u8 keypair_2048[] = {
    0xC1, 0xD2, 0x66, 0xDF, 0xF3, 0x07, 0xD4, 0x32, 0xD1, 0x28, 0x92, 0xC3, 0x37, 0xE2, 0x30, 0x2A,
    0x04, 0x1C, 0x46, 0xA7, 0x2C, 0x1C, 0x73, 0x38, 0xFF, 0x44, 0xE3, 0xD8, 0xEF, 0x53, 0x3E, 0xDB,
    0x5B, 0x29, 0x8D, 0xC8, 0xDC, 0xD3, 0x2F, 0x7A, 0xA4, 0x25, 0x6E, 0xA7, 0x23, 0x44, 0x24, 0xA4,
    0xF3, 0xB7, 0xE1, 0x76, 0x54, 0x59, 0x8D, 0xB7, 0xFE, 0x13, 0x55, 0x47, 0x78, 0x3A, 0x17, 0x03,

```

0x92, 0x02, 0xCC, 0x4F, 0x34, 0x74, 0x61, 0xCB, 0x94, 0x05, 0xD3, 0xDA, 0xC2, 0x9C, 0x52, 0xC0,
0xD7, 0x81, 0x44, 0xC5, 0x6B, 0x6E, 0xB4, 0xF6, 0xC7, 0x7A, 0x51, 0xA9, 0xE1, 0x54, 0x1B, 0x2F,
0x8E, 0xFF, 0x9E, 0x6A, 0xAC, 0xED, 0x66, 0x66, 0x22, 0x14, 0x5F, 0xF4, 0x54, 0xA0, 0x18, 0xB5,
0x4D, 0xDF, 0xA5, 0xE3, 0xDF, 0x4F, 0xD2, 0xA5, 0xDC, 0xE4, 0xD2, 0x92, 0x99, 0x07, 0xE1, 0xA5,
0xDE, 0x66, 0x0E, 0x76, 0xE5, 0x6B, 0x0F, 0x2E, 0x11, 0x18, 0xBE, 0x35, 0x6A, 0xEE, 0x16, 0x7A,
0xA6, 0x3F, 0x99, 0x42, 0x26, 0xC8, 0x1D, 0x1F, 0x3D, 0xAF, 0xB8, 0xC1, 0x88, 0x72, 0x4C, 0xEA,
0x91, 0xB4, 0x92, 0x84, 0x62, 0x60, 0xE8, 0x50, 0x26, 0xAF, 0xEF, 0xC8, 0xBB, 0xB3, 0x54, 0x77,
0x0D, 0x3A, 0xA2, 0xA1, 0xB7, 0xDD, 0x9C, 0x02, 0x82, 0x30, 0x27, 0x99, 0xEE, 0xD3, 0x4F, 0x19,
0xE0, 0x1A, 0x84, 0x15, 0x79, 0xB0, 0x85, 0x2A, 0xD0, 0xAA, 0xF4, 0x31, 0xD4, 0x17, 0x47, 0x88,
0x75, 0x1F, 0x65, 0x22, 0x59, 0x9B, 0x10, 0x3C, 0xB9, 0x91, 0x08, 0xAA, 0xB5, 0xF6, 0x9A, 0x17,
0x06, 0x33, 0x51, 0x51, 0xFC, 0x6B, 0xE9, 0xC5, 0xB5, 0x9C, 0x11, 0x04, 0x7A, 0x5A, 0x1A, 0xEF,
0x1E, 0x0E, 0x37, 0xB9, 0xCF, 0xB2, 0xEF, 0xDA, 0xDD, 0x65, 0xBF, 0x6E, 0x6F, 0xDA, 0xDD, 0x5F,
0xF1, 0x55, 0xEC, 0x7E, 0xBA, 0xAD, 0x2C, 0x75, 0x76, 0xCC, 0x47, 0x67, 0xD9, 0xFF, 0x6E, 0xCC,
0x4C, 0xDF, 0x35, 0x45, 0x62, 0x2A, 0x5D, 0x75, 0xB4, 0x7A, 0x2B, 0x6D, 0x0E, 0x4B, 0x1B, 0x2D,
0x8D, 0x6C, 0x9E, 0x8C, 0x4D, 0x61, 0xCE, 0xC8, 0x52, 0x10, 0x2E, 0x3A, 0x7B, 0x44, 0x49, 0x60,
0x03, 0x36, 0x0E, 0x93, 0xE9, 0x24, 0xD7, 0x63, 0x70, 0x2B, 0x80, 0x3B, 0x47, 0xBD, 0x62, 0x93,
0x46, 0xE0, 0xC9, 0x5A, 0xA0, 0xEA, 0xA2, 0x07, 0xCD, 0x15, 0xC2, 0x3B, 0x44, 0xE9, 0xE5, 0xCF,
0x38, 0xB2, 0x83, 0xE4, 0x25, 0xE6, 0x0E, 0x19, 0x39, 0xA9, 0xA7, 0x87, 0xC1, 0x67, 0x59, 0xF7,
0xA2, 0xA6, 0xF9, 0x35, 0x5D, 0xC0, 0xB5, 0x96, 0xA3, 0x77, 0xFC, 0xB5, 0x8C, 0x81, 0xE6, 0x0E,
0xEA, 0x87, 0xAA, 0x8E, 0x25, 0x0F, 0x33, 0x1F, 0x70, 0x94, 0x61, 0x77, 0xC9, 0x55, 0x22, 0xF7,
0xCD, 0x99, 0x61, 0xBD, 0x6A, 0xE1, 0x4B, 0xB2, 0x77, 0x40, 0x0B, 0x96, 0xF2, 0x6E, 0xE8, 0x85,
0x0E, 0x5C, 0x22, 0xB5, 0x8C, 0xA7, 0x1A, 0x5A, 0x5E, 0xE4, 0x88, 0xC8, 0xA6, 0xA0, 0x83, 0x07,
0xE6, 0xFB, 0x88, 0xC2, 0xA2, 0xC0, 0x26, 0x61, 0xD7, 0xBA, 0x6A, 0x70, 0x4E, 0x97, 0x54, 0x25,
0x08, 0xE6, 0x63, 0x33, 0x30, 0xDC, 0x39, 0xF9, 0x76, 0xAE, 0x29, 0x1C, 0x0E, 0x0F, 0x0A, 0x38,
0xE6, 0xDE, 0x3F, 0x49, 0xD7, 0x22, 0x65, 0x8D, 0x57, 0x1C, 0x2C, 0x3D, 0x83, 0x5B, 0xCA, 0x93,
0x91, 0xA9, 0x0A, 0xAB, 0xA9, 0x5D, 0x97, 0x46, 0x00, 0xE8, 0xD4, 0x15, 0xC4, 0x71, 0x14, 0x48,
0xB6, 0xD2, 0xA2, 0xDF, 0x4E, 0x28, 0x4A, 0xB3, 0xF3, 0xE6, 0x14, 0x43, 0xD6, 0x96, 0xDC, 0x6D,
0xF1, 0xB1, 0x40, 0xE5, 0x2C, 0x07, 0x65, 0xBE, 0x14, 0x3B, 0x4C, 0x66, 0xDD, 0x09, 0x16, 0xD9,
0xB6, 0x6A, 0xAD, 0x2D, 0x09, 0x6B, 0x96, 0x5C, 0x5F, 0x64, 0x7F, 0x60, 0xC7, 0x4D, 0xA4, 0xF1,
0x35, 0xF5, 0xCD, 0xAD, 0x59, 0x4B, 0x0E, 0x83, 0xC6, 0xDC, 0x45, 0x70, 0xD4, 0x49, 0xFF, 0x0C,
0x0A, 0x87, 0x1C, 0xBC, 0x8E, 0x4D, 0x2C, 0xE1, 0x0B, 0xEE, 0x81, 0x12, 0xE1, 0x53, 0xB7, 0x66,
0xA4, 0x66, 0xBA, 0xD4, 0x20, 0x79, 0x21, 0x79, 0x5F, 0x32, 0xF2, 0x65, 0x5C, 0x8F, 0x91, 0x30,
0x23, 0x76, 0x0B, 0x9C, 0xBE, 0x5F, 0x3C, 0x04, 0x4B, 0x87, 0x73, 0xDE, 0x1A, 0xF3, 0xF5, 0x2A,
0x12, 0x00, 0x15, 0x63, 0x61, 0x26, 0x28, 0x67, 0xF6, 0xC6, 0xFF, 0x03, 0xC5, 0x33, 0x9A, 0xB4,
0x6C, 0xB4, 0x6B, 0x50, 0x64, 0x42, 0x22, 0xBF, 0xEF, 0x65, 0xE5, 0x95, 0x50, 0xD4, 0x6B, 0x11,
0xC1, 0x42, 0xB4, 0xAF, 0x66, 0xE1, 0xBB, 0x51, 0xBB, 0x65, 0xDC, 0xDA, 0x8F, 0x4B, 0x13, 0xF7,
0x85, 0x5C, 0x2E, 0x67, 0xFD, 0x41, 0xF2, 0x25, 0xEA, 0xF0, 0x22, 0x14, 0x61, 0x60, 0xCB, 0x76,
0x33, 0x20, 0x54, 0x6B, 0x4A, 0xB6, 0xC1, 0x83, 0x48, 0xF6, 0x18, 0x3B, 0x7E, 0xC9, 0xB2, 0xF4,
0x97, 0x41, 0x25, 0x2E, 0x8A, 0xFE, 0xF4, 0x66, 0x96, 0xD6, 0x16, 0x60, 0x8E, 0xB9, 0xDE, 0x1C,
0xBF, 0xAF, 0x21, 0x3D, 0x7D, 0x3D, 0xFC, 0x64, 0xF2, 0x34, 0x79, 0xF9, 0xF6, 0xAC, 0x8E, 0xC6,
0xE6, 0x09, 0xD7, 0x27, 0x80, 0x62, 0x91, 0x58, 0xA2, 0xDE, 0x65, 0x5B, 0xE9, 0x2E, 0xCB, 0x31,
0xF9, 0x53, 0x9A, 0xD6, 0x4A, 0x5F, 0xA8, 0xD6, 0x01, 0x95, 0x04, 0xAC, 0xB1, 0xE5, 0x5F, 0x70,
0xE3, 0x15, 0xB2, 0x0C, 0x02, 0xD8, 0x63, 0xCC, 0x62, 0x32, 0x25, 0x86, 0xF3, 0x54, 0x9B, 0xDA,
0xBC, 0xE1, 0x6A, 0xE4, 0xBC, 0x30, 0x78, 0x30, 0x59, 0x53, 0x35, 0x75, 0xAB, 0x7E, 0x4F, 0xD1,
0x97, 0x59, 0xD3, 0x96, 0xE9, 0x16, 0x2E, 0xB1, 0x46, 0x5E, 0x69, 0xE2, 0x75, 0xD8, 0x88, 0x8E,
0x82, 0x02, 0xE3, 0x6B, 0xF1, 0xE3, 0x51, 0xB9, 0x7F, 0x5F, 0xEB, 0x1E, 0x50, 0x6C, 0xB9, 0x4C,
0x8A, 0x90, 0xC0, 0x6D, 0xAC, 0x70, 0xDC, 0x21, 0x06, 0x1E, 0x07, 0xEA, 0x66, 0x07, 0xBF, 0x9D,
0xFE, 0x1C, 0x5B, 0x2E, 0xC7, 0x5D, 0xA8, 0xD6, 0x34, 0x9A, 0x58, 0x5E, 0x85, 0x39, 0x4B, 0x0B,
0xCC, 0x6A, 0xAC, 0x3B, 0x0A, 0x15, 0x2A, 0xE5, 0x0E, 0xBC, 0x47, 0xE8, 0xEF, 0x8F, 0x99, 0x67,
0x25, 0x05, 0x97, 0xF3, 0x6F, 0xC1, 0x11, 0x74, 0xA8, 0x38, 0x34, 0x07, 0x8D, 0x95, 0x4A, 0xF0,
0xD8, 0xC7, 0xF3, 0x2E, 0x66, 0x7E, 0x6D, 0x3B, 0x6F, 0x1E, 0x9C, 0x68, 0x8D, 0x9F, 0xAD, 0xB6,
0xF0, 0x38, 0x86, 0xC3, 0xD6, 0xAD, 0x67, 0x2E, 0xE4, 0xE5, 0x6E, 0x1C, 0x2B, 0xB1, 0x1F, 0xFB,
0x32, 0x51, 0x63, 0x2F, 0xC4, 0xB2, 0x45, 0x9E, 0xA3, 0xBA, 0x3A, 0xCF, 0xBF, 0x94, 0x53, 0x18,
0xB5, 0x54, 0x79, 0xEB, 0x30, 0x82, 0x7A, 0x4F, 0x53, 0xC6, 0x18, 0xB8, 0xDA, 0xF8, 0xEB, 0x9A,
0x9B, 0x02, 0x20, 0xB3, 0x02, 0xD1, 0x38, 0xC4, 0x48, 0x15, 0x07, 0xB3, 0x10, 0x6E, 0xC2, 0x6C,
0xA8, 0x71, 0x16, 0x71, 0x57, 0xD8, 0x9F, 0xF8, 0x22, 0x57, 0xF6, 0x8D, 0xC5, 0x62, 0x9F, 0x52,
0xE7, 0xF7, 0x1D, 0xFF, 0x7A, 0x08, 0xB4, 0xD2, 0xDB, 0x2C, 0xE0, 0x21, 0x2C, 0xE0, 0xD4, 0x83,
0x07, 0xEF, 0xA1, 0x60, 0xF7, 0x51, 0x18, 0x5D, 0xF0, 0xCF, 0x8A, 0x9F, 0xE5, 0xC5, 0xA6, 0x01,
0x24, 0xC8, 0xAB, 0xE1, 0xD9, 0xDF, 0x0F, 0xF4, 0x77, 0xE8, 0x7E, 0x38, 0xBA, 0x37, 0x8D, 0xF2,
0x93, 0xB2, 0xFA, 0x28, 0x3C, 0x73, 0x46, 0xA6, 0x94, 0xC0, 0x3B, 0x02, 0xA0, 0x03, 0xF1, 0xCA,

```

0x5F, 0xF8, 0xBE, 0x53, 0x02, 0x38, 0x4C, 0x64, 0xE5, 0x78, 0x75, 0xE5, 0x0A, 0x24, 0xF4, 0x05,
0xCE, 0x59, 0xDE, 0x70, 0x94, 0xD0, 0xCD, 0x7D, 0xCD, 0xE8, 0x96, 0x9B, 0x8E, 0x60, 0xAA, 0xBE,
0x8E, 0x4A, 0xA6, 0x92, 0x8B, 0x84, 0x58, 0x2F, 0x86, 0x86, 0x2B, 0xA5, 0xB4, 0x81, 0x62, 0x17,
0x55, 0xBE, 0xF6, 0xC3, 0xEA, 0xB3, 0xB4, 0xB2, 0xE5, 0xAF, 0x00, 0xBC, 0x5D, 0x69, 0xF2, 0x1A,
0x65, 0xB1, 0x66, 0x4C, 0xA2, 0x36, 0x45, 0x86, 0x33, 0xEE, 0x7F, 0x53, 0xD9, 0x03, 0xA2, 0x8F,
0xBE, 0xD3, 0x74, 0xC3, 0x3A, 0x73, 0x13, 0x65, 0x49, 0x70, 0xE0, 0x41, 0x8A, 0x3A, 0x6A, 0xFB,
0x22, 0x80, 0x5B, 0xD2, 0xFD, 0x31, 0xDF, 0x57, 0x7D, 0x34, 0x6F, 0x62, 0x71, 0xB9, 0x1F, 0x33,
0x1A, 0x70, 0x3B, 0x8B, 0x19, 0x97, 0x90, 0xE5, 0xD8, 0xB7, 0xAC, 0xFB, 0x7D, 0x3D, 0x84, 0x61,
};
unsigned char pkcs1_keypair_rsa1024[] = {
    0x30, 0x82, 0x02, 0x5c, 0x02, 0x01, 0x00, 0x02,
    0x81, 0x81, 0x00, 0xc4, 0x42, 0xee, 0x95, 0xfe,
    0xab, 0x6a, 0xff, 0x3e, 0x13, 0xd8, 0x1d, 0x67,
    0x13, 0x4d, 0x27, 0x0f, 0x57, 0x8f, 0x50, 0xff,
    0x71, 0x7d, 0x8a, 0x5d, 0x6b, 0x2a, 0x01, 0x39,
    0x9e, 0xb6, 0xcf, 0x81, 0xff, 0x90, 0x61, 0xd6,
    0x68, 0x08, 0xfa, 0x5c, 0x90, 0x92, 0xe1, 0x19,
    0x10, 0x6e, 0xec, 0x64, 0x4b, 0x55, 0x7d, 0xbd,
    0xad, 0x7d, 0x37, 0x04, 0xca, 0xbf, 0x97, 0xa0,
    0xd5, 0xd7, 0xcf, 0x33, 0x65, 0x81, 0xde, 0xc5,
    0xc2, 0x01, 0x4f, 0x2f, 0x53, 0x01, 0xa7, 0x8f,
    0xc8, 0x22, 0x30, 0x46, 0x87, 0xa3, 0x09, 0x5c,
    0x63, 0xcd, 0x21, 0xde, 0xc3, 0xd9, 0x95, 0xc0,
    0xff, 0x7d, 0xf3, 0x74, 0x08, 0x66, 0x6f, 0x80,
    0xfe, 0x7f, 0x91, 0x9a, 0xb9, 0xf2, 0x56, 0x28,
    0x89, 0x63, 0x9e, 0x96, 0x31, 0x27, 0x95, 0x15,
    0x1f, 0xfd, 0xb1, 0xb5, 0x8a, 0x21, 0x81, 0x01,
    0x2c, 0xe4, 0x85, 0x02, 0x03, 0x01, 0x00, 0x01,
    0x02, 0x81, 0x80, 0x06, 0x75, 0xb3, 0x56, 0x1c,
    0x8a, 0x50, 0x93, 0xfb, 0xf2, 0xb9, 0xce, 0xa6,
    0xab, 0xb1, 0xc8, 0x95, 0xe2, 0xde, 0xd0, 0x5e,
    0x98, 0x07, 0xdf, 0x9b, 0xd3, 0x8c, 0x9b, 0x7a,
    0xce, 0x65, 0xd0, 0x2b, 0xc4, 0x4d, 0xa6, 0x76,
    0x60, 0xe1, 0xdf, 0x21, 0x87, 0x03, 0x94, 0x6d,
    0xda, 0x66, 0xcc, 0x41, 0x12, 0x4b, 0x28, 0x3f,
    0xa0, 0x2e, 0xf1, 0xcd, 0xbd, 0xd8, 0xd3, 0x71,
    0xe7, 0xc6, 0x82, 0x9b, 0x24, 0x4c, 0x40, 0x77,
    0x42, 0xd9, 0xf2, 0x17, 0x0b, 0x38, 0x18, 0x85,
    0xb4, 0xe1, 0x4d, 0x06, 0x5a, 0xe0, 0xe7, 0x42,
    0xc4, 0x74, 0x8a, 0x83, 0x59, 0x98, 0x34, 0xff,
    0x78, 0x50, 0xd4, 0xfd, 0x4d, 0x41, 0xfb, 0x04,
    0xb9, 0x89, 0xfe, 0xe7, 0xaa, 0xb6, 0x15, 0x1a,
    0x94, 0x96, 0x2c, 0xaa, 0x5c, 0x5b, 0x6a, 0x98,
    0xb7, 0xdd, 0x42, 0xc1, 0x58, 0x8c, 0x41, 0x52,
    0xc3, 0x48, 0xc1, 0x02, 0x41, 0x00, 0xd0, 0x45,
    0xad, 0x11, 0xee, 0xff, 0x1d, 0xee, 0x5d, 0xca,
    0xa2, 0x1f, 0xdc, 0xe6, 0x63, 0xd6, 0xf0, 0x39,
    0x71, 0xf3, 0xc6, 0x22, 0x57, 0xbe, 0x54, 0x69,
    0xfe, 0x1a, 0x72, 0x85, 0xec, 0x90, 0x09, 0x59,
    0x99, 0xe4, 0xea, 0xf4, 0x9c, 0x7c, 0x77, 0xc7,
    0xcf, 0x97, 0x45, 0xc5, 0xda, 0xdc, 0xe9, 0x27,
    0x76, 0xbc, 0x3d, 0xfa, 0x5e, 0x49, 0xec, 0x03,
    0x4b, 0x16, 0x87, 0x65, 0x02, 0x1f, 0x02, 0x41,
    0x00, 0xf1, 0x3c, 0xa5, 0x1b, 0xd4, 0x1f, 0x9d,
    0xbe, 0x17, 0x12, 0xeb, 0x41, 0x87, 0x84, 0x36,
    0x9d, 0x21, 0xed, 0x01, 0xb4, 0x4f, 0x4c, 0x54,
    0x5e, 0x1d, 0xa0, 0xfc, 0xca, 0xd6, 0x79, 0xb4,
    0x67, 0xc7, 0xfa, 0x52, 0x08, 0x52, 0xb1, 0x4f,
    0x86, 0x01, 0xde, 0x8d, 0x1a, 0x78, 0x50, 0x4e,
    0x16, 0xb7, 0x30, 0x04, 0x9b, 0x69, 0xee, 0x04,

```

```

0xdb, 0x17, 0x97, 0x85, 0x7e, 0x19, 0x41, 0x6c,
0xdb, 0x02, 0x41, 0x00, 0x91, 0x50, 0x6c, 0x06,
0xdd, 0x38, 0x07, 0x0c, 0x2f, 0x48, 0x98, 0x81,
0x9e, 0xc0, 0xe8, 0xdc, 0x28, 0xd4, 0x0b, 0xdb,
0xc8, 0x5c, 0x61, 0xd8, 0x1c, 0x51, 0xaf, 0xc7,
0x88, 0x2c, 0x44, 0xf7, 0xe2, 0x6e, 0x2f, 0xdf,
0xf6, 0xc7, 0x20, 0xff, 0xff, 0x1a, 0xdc, 0x8e,
0x71, 0x52, 0x30, 0xf8, 0x7d, 0x7c, 0xd9, 0x6d,
0x51, 0xd9, 0x04, 0x3c, 0x93, 0x7d, 0x60, 0xc0,
0xa1, 0x4b, 0x2a, 0x85, 0x02, 0x40, 0x73, 0x6d,
0x2f, 0x1b, 0x57, 0xae, 0x21, 0x0e, 0x19, 0x80,
0x45, 0xec, 0xbe, 0xe8, 0xf6, 0x30, 0xcd, 0x50,
0xd8, 0x25, 0xec, 0x63, 0x7b, 0x62, 0xe5, 0x0e,
0x68, 0xa6, 0xad, 0x64, 0xe2, 0x7b, 0x5d, 0xe6,
0x5a, 0x65, 0xfd, 0x1b, 0x36, 0x0f, 0xca, 0xc9,
0x2f, 0xfe, 0xe0, 0x5e, 0x9c, 0x5e, 0xa6, 0x1c,
0x65, 0xb7, 0xef, 0x41, 0xa2, 0x35, 0x2a, 0xde,
0xa7, 0x53, 0x24, 0xc6, 0x77, 0x4d, 0x02, 0x40,
0x78, 0x72, 0xf9, 0x71, 0x10, 0xcf, 0x87, 0x8b,
0xb4, 0xe2, 0xcd, 0xee, 0x73, 0x87, 0xad, 0xf4,
0x87, 0x2d, 0x31, 0x4b, 0xe8, 0xbe, 0xc9, 0xc9,
0x52, 0xdc, 0xd4, 0x6a, 0xa0, 0x8a, 0xc6, 0x35,
0xad, 0x4e, 0xa7, 0x44, 0x53, 0x37, 0xd9, 0xb3,
0x33, 0x47, 0x80, 0xb9, 0xff, 0x43, 0x0e, 0x49,
0xbd, 0x7f, 0x68, 0x63, 0x7f, 0xe1, 0x97, 0x14,
0x72, 0x3e, 0x96, 0x15, 0x6c, 0x32, 0x43, 0x49,
};

/**
 * get device handle
 */
hdev = (DEVHANDLE)instance->values->get(instance->values, SKF_DEVICE_HANDLE);
if (!hdev) {
    ret = SAR_INVALIDHANDLEERR;
    LOG_ERROR(symbol->mclog, -1, "invalid device handle\n");
    goto end;
}
skf->SKF_LockDev(hdev, -1);

/**
 * open app
 */
ret = skf->SKF_OpenApplication(hdev, app, &happ);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_OpenApplication() failed: %#x\n", ret);
    goto endl;
}

/**
 * verify pin
 */
ret = skf->SKF_VerifyPIN(happ, USER_TYPE, pin, (void *)&retry);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_VerifyPIN() failed: %#x\n", ret);
    goto end;
}

/**
 * open container
 */

```

```
ret = skf->SKF_OpenContainer(happ, ctn, &hctn);
if (ret) {
    /**
     * create container
     */
    ret = skf->SKF_CreateContainer(happ, ctn, &hctn);
    if (ret) {
        LOG_ERROR(symbol->mclog, -1, "SKF_CreateContainer() failed: %#x\n", ret);
        goto end;
    }
}

/**
 * get container type
 */
ret = skf->SKF_GetContainerType(hctn, (void *)&type);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_GetContainerType() failed: %#x\n", ret);
    goto end;
}

/**
 * check container type
 */
if (type == CTNF_ECC) {
    skf->SKF_CloseContainer(hctn);
    memset(&hctn, 0, sizeof(hctn));

    ret = skf->SKF_DeleteContainer(happ, ctn);
    if (ret) {
        LOG_ERROR(symbol->mclog, -1, "SKF_DeleteContainer() failed: %#x\n", ret);
        goto end;
    }

    ret = skf->SKF_CreateContainer(happ, ctn, &hctn);
    if (ret) {
        LOG_ERROR(symbol->mclog, -1, "SKF_CreateContainer() failed: %#x\n", ret);
        goto end;
    }
}

/**
 * verify pin
 */
ret = skf->SKF_VerifyPIN(happ, USER_TYPE, pin, (void *)&retry);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_VerifyPIN() failed: %#x\n", ret);
    goto end;
}

/**
 * generate keypair
 */
ret = skf->SKF_GenRSAKeyPair(hctn, bits, (void *)&pubkey);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_GenRSAKeyPair() failed: %#x\n", ret);
    goto end;
}

/**
```

```

    * encrypt prikey
    */
    ret = skf->SKF_RSASessionKey(hctn, SGD_SM1_ECB, (void *)pubkey, sessionkey, (void *)&sessionkeylen,
    &hkey);
    if (ret) {
        LOG_ERROR(symbol->mclog, -1, "SKF_RSASessionKey() failed: %#x\n", ret);
        goto end;
    }

    memset(&bp, 0, sizeof(bp));
    bp.PaddingType = 1;
    ret = skf->SKF_EncryptInit(hkey, bp);
    if (ret) {
        LOG_ERROR(symbol->mclog, -1, "SKF_EncryptInit() failed: %#x\n", ret);
        goto end;
    }

    enclen = sizeof(enc_data);
    if (bits == 1024) {
        if (is_pkcs1_keypair) {
            ret = skf->SKF_Encrypt(hkey, pkcs1_keypair_rsa1024, sizeof(pkcs1_keypair_rsa1024), enc_data, (void
            *)&enclen);
        } else {
            ret = skf->SKF_Encrypt(hkey, keypair_1024, sizeof(keypair_1024), enc_data, (void *)&enclen);
        }
    } else {
        ret = skf->SKF_Encrypt(hkey, keypair_2048, sizeof(keypair_2048), enc_data, (void *)&enclen);
    }
    if (ret) {
        LOG_ERROR(symbol->mclog, -1, "SKF_Encrypt() failed: %#x\n", ret);
        goto end;
    }

    /**
    * import rsa keypair
    */
    ret = skf->SKF_ImportRSAKeyPair(hctn, SGD_SM1_ECB, sessionkey, sessionkeylen, enc_data, enclen);
    if (ret) {
        LOG_ERROR(symbol->mclog, -1, "SKF_ImportRSAKeyPair() failed: %#x\n", ret);
        goto end;
    }
    LOG_INFO(mclog, -1, "rsa %d import keypair [g]succ[n]\n", bits);

end:
    if (hkey) skf->SKF_CloseHandle(hkey);
    if (hctn) skf->SKF_CloseContainer(hctn);
    if (happ) skf->SKF_CloseApplication(happ);
    skf->SKF_UnlockDev(hdev);
    return ret;
}

```

非对称加解密

相关接口

7. 6. 16 ECC 外来公钥加密

函数原型	ULONG DEVAPI SKF_ExtECCEncrypt (DEVHANDLE hDev, ECCPUBLICKEYBLOB* pECCPubKeyBlob, BYTE* pbPlainText, ULONG ulPlainTextLen, PECCCIPHERBLOB pCipherText)		
功能描述	使用外部传入的 ECC 公钥对输入数据做加密运算并输出结果。		
参数	hDev	[IN] 设备句柄。	
	pECCPubKeyBlob	[IN] ECC 公钥数据结构。	
	pbPlainText	[IN] 待加密的明文数据。	
	ulPlainTextLen	[IN] 待加密明文数据的长度。	
	pCipherText	[OUT] 密文数据。	
返回值	SAR_OK:	成功。	
	其他:	错误码。	

函数原型	ULONG DEVAPI SKF_ExtECCDecrypt (DEVHANDLE hDev, ECCPRIVATEKEYBLOB* pECCPriKeyBlob, PECCCIPHERBLOB pCipherText, BYTE* pbPlainText, ULONG* pulPlainTextLen)		
功能描述	使用外部传入的 ECC 私钥对输入数据做解密运算并输出结果。		
参数	hDev	[IN] 设备句柄。	
	pECCPriKeyBlob	[IN] ECC 私钥数据结构。	
	pCipherText	[IN] 待解密的密文数据。	
	pbPlainText	[OUT] 返回明文数据，如果该参数为 NULL，则由 pulPlainTextLen 返回明文数据的实际长度。	
	pulPlainTextLen	[IN, OUT] 输入时表示 pbPlainText 缓冲区的长度，输出时表示明文数据的实际长度。	
返回值	SAR_OK:	成功。	
	其他:	错误码。	

7.6.9 RSA 外来公钥运算

函数原型	ULONG DEVAPI SKF_ExtRSAPubKeyOperation (DEVHANDLE hDev, RSAPUBLICKEYBLOB* pRSAPubKeyBlob, BYTE* pbInput, ULONG ulInputLen, BYTE* pbOutput, ULONG* pulOutputLen)	
功能描述	使用外部传入的 RSA 公钥对输入数据做公钥运算并输出结果。	
参数	hDev	[IN] 设备句柄。
	pRSAPubKeyBlob	[IN] RSA 公钥数据结构。
	pbInput	[IN] 指向待运算的原始数据缓冲区。
	ulInputLen	[IN] 待运算原始数据的长度，必须为公钥模长。
	pbOutput	[OUT] 指向 RSA 公钥运算结果缓冲区，如果该参数为 NULL，则由 pulOutputLen 返回运算结果的实际长度。
	pulOutputLen	[IN, OUT] 输入时表示 pbOutput 缓冲区的长度，输出时表示 RSA 公钥运算结果的实际长度。
返回值	SAR_OK:	成功。
	其他:	错误码。

7.6.10 RSA 外来私钥运算

函数原型	ULONG DEVAPI SKF_ExtRSAPriKeyOperation (DEVHANDLE hDev, RSAPRIVATEKEYBLOB* pRSAPriKeyBlob, BYTE* pbInput, ULONG ulInputLen, BYTE* pbOutput, ULONG* pulOutputLen)	
功能描述	直接使用外部传入的 RSA 私钥对输入数据做私钥运算并输出结果。	
参数	hDev	[IN] 设备句柄。
	pRSAPriKeyBlob	[IN] RSA 私钥数据结构。
	pbInput	[IN] 指向待运算数据缓冲区。
	ulInputLen	[IN] 待运算数据的长度，必须为公钥模长。
	pbOutput	[OUT] RSA 私钥运算结果，如果该参数为 NULL，则由 pulOutputLen 返回运算结果的实际长度。
	pulOutputLen	[IN, OUT] 输入时表示 pbOutput 缓冲区的长度，输出时表示 RSA 私钥运算结果的实际长度。
返回值	SAR_OK:	成功。
	其他:	错误码。

参考代码

```
#define RSA_PKCS1_PADDING_SIZE 11
static int RSA_padding_add_PKCS1_type_2(unsigned char *to, int tlen, const unsigned char *from, int flen)
{
    int j;
    unsigned char *p;

    if (flen > (tlen-RSA_PKCS1_PADDING_SIZE))
        return(-1);
```

```

p=(unsigned char *)to;

*(p++)=0;
*(p++)=2; /* Public Key BT (Block Type) */

/* pad out with non-zero random data */
j=tlen-3-flen;
memset(p,0xff,j);
p+=j;
*(p++)='\0';
memcpy(p,from,(unsigned int)flen);

return(0);
}

static int asym_enc_dec(core_t *core, settings_t *settings, menu_t *menu)
{
    int ret = 0;
    int retry = 0;
    int rsa_bits = 0;
    int data_len = 0;
    char *app = "DEFAULT";
    char *ctn = "DEFAULT";
    char *type = "sm2";
    int bits = 1024;
    int algo = get_asym_algo(type);
    api_t *api = core->get_api(core);
    api_symbol_t *symbol = api->get_symbol(api);
    api_instance_t *instance = api->get_instance(api);
    menu_controller_t *controller = menu->get_controller(menu);
    settings_t *property = controller->get_property(controller);
    unsigned char data[256] = {0};
    unsigned char rsa_data[256] = {0};
    unsigned char pubkey[2048] = {0};
    unsigned char prikey[2048] = {0};
    unsigned int pubkeylen = sizeof(pubkey);
    unsigned char enc_data[512] = {0};
    unsigned int enc_data_len = sizeof(enc_data);
    unsigned char dec_data[512] = {0};
    unsigned int dec_data_len = sizeof(dec_data);
    mclog_api mclog = symbol->mclog;
    SKF_FUNC_LIST *skf = instance->skf;
    DEVHANDLE hdev = (DEVHANDLE)NULL;
    HAPPLICATION happ = (HAPPLICATION)NULL;
    HCONTAINER hctn = (HCONTAINER)NULL;
    RSAPUBLICKEYBLOB *prsapubkey = NULL;
    RSAPRIVATEKEYBLOB *prsaprikey = NULL;
    ECCPRIVATEKEYBLOB sm2_pri;
    unsigned char sm2_keypair[] = {
        //x
        0x19, 0x79, 0x5d, 0xf7, 0x01, 0xf3, 0x9d, 0x1f, 0xb2, 0x20, 0xc4, 0x5f, 0xa7, 0xfa, 0x4e, 0xbf,
        0xad, 0xd1, 0x70, 0x25, 0x37, 0xb9, 0x46, 0xcd, 0x3d, 0x48, 0x04, 0xb3, 0x7f, 0xbc, 0x3e, 0xa5,
        //y
        0x2b, 0x2c, 0xee, 0xd6, 0xcc, 0x04, 0x2b, 0x5b, 0xbb, 0x56, 0x8d, 0xed, 0x3b, 0x36, 0x73, 0xf2,
        0x88, 0xe1, 0x9c, 0xc4, 0x9a, 0xe3, 0xc3, 0x50, 0xd2, 0xb8, 0x09, 0x03, 0xd8, 0x6d, 0x91, 0x2c,
        //d
        0x3f, 0x91, 0x68, 0xe8, 0x6d, 0x2a, 0xac, 0xaa, 0x2c, 0x81, 0xd8, 0xba, 0x24, 0x9b, 0xc9, 0x5a,
        0x60, 0xe0, 0x47, 0x50, 0xa2, 0xee, 0xaa, 0x63, 0x26, 0x2b, 0x54, 0xc4, 0x75, 0x51, 0xb8, 0xdc
    };
};

```

```

/**
 * algo bits
 */
if (algo == SGD_SM2_1) bits = 256;

/**
 * get device handle
 */
hdev = (DEVHANDLE)instance->values->get(instance->values, SKF_DEVICE_HANDLE);
if (!hdev) {
    ret = SAR_INVALIDHANDLEERR;
    LOG_ERROR(symbol->mclog, -1, "invalid device handle\n");
    goto end;
}
skf->SKF_LockDev(hdev, -1);

/**
 * open container
 */
ret = skf->SKF_OpenContainer(happ, ctn, &hctn);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_OpenContainer() failed: %#x\n", ret);
    goto end;
}

/**
 * generate keypair
 */
memset(data, 'a', sizeof(data));
if (get_asym_algo(type) == SGD_RSA) {
    rsa_bits = (bits == 1024) ? GENERATE_KEY_BIT_1024 : GENERATE_KEY_BIT_2048;

    prsaprikey = (void *)prikey;
    ret = skf->SKF_GenExtRSAKey(hdev, bits, prsaprikey);
    if (ret) {
        LOG_ERROR(symbol->mclog, -1, "SKF_GenExtRSAKey() failed: %#x\n", ret);
        goto end;
    }

    prsapubkey->AlgID = prsaprikey->AlgoID;
    prsapubkey->BitLen = prsaprikey->BitLen;
    memcpy(prsapubkey->Modulus, prsaprikey->Modulus, sizeof(prsapubkey->Modulus));
    memcpy(prsapubkey->PublicExponent, prsaprikey->PublicExponent, sizeof(prsapubkey->PublicExponent));

    ret = RSA_padding_add_PKCS1_type_2(rsa_data, bits / 8, data, bits / 8 - RSA_PKCS1_PADDING_SIZE);
    if (ret) goto end;

    enc_data_len = sizeof(enc_data);
    memset(enc_data, 0, enc_data_len);
    ret = skf->SKF_ExtRSAPubKeyOperation(hdev, prsapubkey, data, prsapubkey->BitLen / 8, enc_data,
    &enc_data_len);
    if (ret) {
        LOG_ERROR(symbol->mclog, -1, "SKF_ExtRSAPubKeyOperation() failed: %#x\n", ret);
        goto end;
    }

    dec_data_len = sizeof(dec_data);
    memset(dec_data, 0, dec_data_len);
    ret = skf->SKF_ExtRSAPriKeyOperation(hdev, prsaprikey, enc_data, enc_data_len, dec_data,

```

```

&dec_data_len);
    if (ret) {
        LOG_ERROR(symbol->mclog, -1, "SKF_ExtRSAPubKeyOperation() failed: %#x\n", ret);
        goto end;
    }

    if (memcmp(data, dec_data, dec_data_len)) {
        ret = SAR_FAIL;
        LOG_ERROR(symbol->mclog, -1, "original and decrypt data is not the same!\n");
        LOG_ERROR(symbol->mclog, -1, "original data(%dbytes)\n%#h", dec_data_len, 16, dec_data_len, data +
RSA_PKCS1_PADDING_SIZE);
        LOG_ERROR(symbol->mclog, -1, "decrypt data(%dbytes)\n%#h", dec_data_len, 16, dec_data_len,
dec_data);
        goto end;
    }

} else {
    ret = skf->SKF_ExportPublicKey(hctn, 0, (void *)pubkey, &pubkeylen);
    if (ret) {
        LOG_ERROR(symbol->mclog, -1, "SKF_ExportPublicKey() failed: %#x\n", ret);
        goto end;
    }

    data_len = 32;
    ret = skf->SKF_ExtECCEncrypt(hdev, (void *)pubkey, data, data_len, (void *)enc_data);
    if (ret) {
        LOG_ERROR(symbol->mclog, -1, "SKF_ExtECCEncrypt() failed: %#x\n", ret);
        goto end;
    }

    memset(&sm2_pri, 0, sizeof(sm2_pri));
    sm2_pri.BitLen = 256;
    memcpy(sm2_pri.PrivateKey + 32, sm2_keypair + 64, 32);

    ret = skf->SKF_ExtECCDecrypt(hdev, &sm2_pri, (void *)enc_data, dec_data, &dec_data_len);
    if (ret) {
        LOG_ERROR(symbol->mclog, -1, "SKF_ExtECCDecrypt() failed: %#x\n", ret);
        goto end;
    }

    if (data_len != dec_data_len ||
        memcmp(data, dec_data, data_len)) {
        ret = SAR_FAIL;
        LOG_ERROR(symbol->mclog, -1, "original and decrypt data is not the same!\n");
        goto end;
    }
}

LOG_INFO(mclog, -1, "asym \\[%s] encrypt and decrypt [g]succ[n]\n", type);

end:
if (hctn) skf->SKF_CloseContainer(hctn);
if (happ) skf->SKF_CloseApplication(happ);
skf->SKF_UnlockDev(hdev);
return ret;
}

```

说明

RSA或ECC外部加解密时，用户可替换为自己需要的公钥或私钥。

证书导入

相关接口

7.5.8 导入数字证书

函数原型	ULONG DEVAPI SKF_ImportCertificate(HCONTAINER hContainer, BOOL bSignFlag, BYTE* pbCert, ULONG ulCertLen)		
功能描述	向容器内导入数字证书。		
参数	hContainer	[IN] 容器句柄。	
	bSignFlag	[IN] TRUE 表示签名证书, FALSE 表示加密证书。	
	pbCert	[IN] 指向证书内容缓冲区。	
	ulCertLen	[IN] 证书长度。	
返回值	SAR_OK:	成功。	
	其他:	错误码。	

导入要求

- **前提**: 保证与证书相匹配的密钥对已导入容器。签名证书导入签名密钥对, 加密证书导入加密密钥对。具体导入流程可参考[导入非对称密钥对](#)章节。
- **证书格式**: 必须符合X509标准。

参考代码

```
int sm2_import_cert(core_t *core, settings_t *settings, menu_t *menu)
{
    int ret = 0;
    char *pin = "111111";
    char *app = "DEFAULT";
    char *ctn = "DEFAULT";
    api_t *api = core->get_api(core);
    api_symbol_t *symbol = api->get_symbol(api);
    api_instance_t *instance = api->get_instance(api);
    menu_controllor_t *controllor = menu->get_controllor(menu);
    settings_t *property = controllor->get_property(controllor);
    mclog_api mclog = symbol->mclog;
    SKF_FUNCLIST *skf = instance->skf;
    DEVHANDLE hdev = (DEVHANDLE)NULL;
    HAPPLICATION happ = (HAPPLICATION)NULL;
    HCONTAINER hctn = (HCONTAINER)NULL;
    u8 cert[] = {
        0x30, 0x82, 0x03, 0x01, 0x30, 0x82, 0x02, 0xA7, 0xA0, 0x03, 0x02, 0x01, 0x02, 0x02, 0x08, 0x12,
        0x34, 0x56, 0x78, 0x9A, 0xBC, 0xDE, 0xF1, 0x30, 0x0A, 0x06, 0x08, 0x2A, 0x81, 0x1C, 0xCF, 0x55,
        0x01, 0x83, 0x75, 0x30, 0x7D, 0x31, 0x0B, 0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x06, 0x13, 0x02,
        0x43, 0x4E, 0x31, 0x0B, 0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x08, 0x13, 0x02, 0x47, 0x44, 0x31,
        0x0B, 0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x07, 0x13, 0x02, 0x53, 0x5A, 0x31, 0x10, 0x30, 0x0E,
        0x06, 0x03, 0x55, 0x04, 0x0A, 0x13, 0x07, 0x53, 0x41, 0x4E, 0x47, 0x46, 0x4F, 0x52, 0x31, 0x0C,
        0x30, 0x0A, 0x06, 0x03, 0x55, 0x04, 0x0B, 0x13, 0x03, 0x53, 0x53, 0x4C, 0x31, 0x10, 0x30, 0x0E,
        0x06, 0x03, 0x55, 0x04, 0x03, 0x13, 0x07, 0x73, 0x61, 0x6E, 0x67, 0x66, 0x6F, 0x72, 0x31, 0x22,
        0x30, 0x20, 0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x09, 0x01, 0x16, 0x13, 0x73,
        0x61, 0x6E, 0x67, 0x66, 0x6F, 0x72, 0x40, 0x73, 0x61, 0x6E, 0x67, 0x66, 0x6F, 0x72, 0x2E, 0x63,
        0x6F, 0x6D, 0x30, 0x1E, 0x17, 0x0D, 0x31, 0x35, 0x31, 0x30, 0x31, 0x32, 0x30, 0x38, 0x35, 0x37,
        0x34, 0x38, 0x5A, 0x17, 0x0D, 0x32, 0x35, 0x31, 0x30, 0x31, 0x31, 0x30, 0x38, 0x35, 0x37, 0x34,
        0x38, 0x5A, 0x30, 0x7D, 0x31, 0x0B, 0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x06, 0x13, 0x02, 0x43,
        0x4E, 0x31, 0x0B, 0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x08, 0x13, 0x02, 0x47, 0x44, 0x31, 0x0B,
        0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x07, 0x13, 0x02, 0x53, 0x5A, 0x31, 0x10, 0x30, 0x0E, 0x06,
```

```

0x03, 0x55, 0x04, 0x0A, 0x13, 0x07, 0x53, 0x41, 0x4E, 0x47, 0x46, 0x4F, 0x52, 0x31, 0x0C, 0x30,
0x0A, 0x06, 0x03, 0x55, 0x04, 0x0B, 0x13, 0x03, 0x53, 0x4C, 0x31, 0x10, 0x30, 0x0E, 0x06,
0x03, 0x55, 0x04, 0x03, 0x13, 0x07, 0x73, 0x61, 0x6E, 0x67, 0x66, 0x6F, 0x72, 0x31, 0x22, 0x30,
0x20, 0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x09, 0x01, 0x16, 0x13, 0x73, 0x61,
0x6E, 0x67, 0x66, 0x6F, 0x72, 0x40, 0x73, 0x61, 0x6E, 0x67, 0x66, 0x6F, 0x72, 0x2E, 0x63, 0x6F,
0x6D, 0x30, 0x59, 0x30, 0x13, 0x06, 0x07, 0x2A, 0x86, 0x48, 0xCE, 0x3D, 0x02, 0x01, 0x06, 0x08,
0x2A, 0x81, 0x1C, 0xCF, 0x55, 0x01, 0x82, 0x2D, 0x03, 0x42, 0x00, 0x04, 0x19, 0x79, 0x5D, 0xF7,
0x01, 0xF3, 0x9D, 0x1F, 0xB2, 0x20, 0xC4, 0x5F, 0xA7, 0xFA, 0x4E, 0xBF, 0xAD, 0xD1, 0x70, 0x25,
0x37, 0xB9, 0x46, 0xCD, 0x3D, 0x48, 0x04, 0xB3, 0x7F, 0xBC, 0x3E, 0xA5, 0x2B, 0x2C, 0xEE, 0xD6,
0xCC, 0x04, 0x2B, 0x5B, 0xBB, 0x56, 0x8D, 0xED, 0x3B, 0x36, 0x73, 0xF2, 0x88, 0xE1, 0x9C, 0xC4,
0x9A, 0xE3, 0xC3, 0x50, 0xD2, 0xB8, 0x09, 0x03, 0xD8, 0x6D, 0x91, 0x2C, 0xA3, 0x82, 0x01, 0x0F,
0x30, 0x82, 0x01, 0x0B, 0x30, 0x09, 0x06, 0x03, 0x55, 0x1D, 0x13, 0x04, 0x02, 0x30, 0x00, 0x30,
0x2C, 0x06, 0x09, 0x60, 0x86, 0x48, 0x01, 0x86, 0xF8, 0x42, 0x01, 0x0D, 0x04, 0x1F, 0x16, 0x1D,
0x4F, 0x70, 0x65, 0x6E, 0x53, 0x53, 0x4C, 0x20, 0x47, 0x65, 0x6E, 0x65, 0x72, 0x61, 0x74, 0x65,
0x64, 0x20, 0x43, 0x65, 0x72, 0x74, 0x69, 0x66, 0x69, 0x63, 0x61, 0x74, 0x65, 0x30, 0x1D, 0x06,
0x03, 0x55, 0x1D, 0x0E, 0x04, 0x16, 0x04, 0x14, 0x50, 0x30, 0xB3, 0x89, 0x44, 0xF0, 0xB5, 0x9B,
0x21, 0xA1, 0xCF, 0xC6, 0xE3, 0x80, 0xF0, 0xD5, 0x4D, 0xE5, 0x22, 0x25, 0x30, 0x81, 0xB0, 0x06,
0x03, 0x55, 0x1D, 0x23, 0x04, 0x81, 0xA8, 0x30, 0x81, 0xA5, 0x80, 0x14, 0xEC, 0x25, 0x01, 0xB6,
0x82, 0x08, 0xE1, 0xC3, 0xDF, 0x2D, 0x8D, 0x0B, 0xD6, 0xB5, 0x49, 0x4C, 0x05, 0x8F, 0x32, 0x14,
0xA1, 0x81, 0x81, 0xA4, 0x7F, 0x30, 0x7D, 0x31, 0x0B, 0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x06, 0x06,
0x13, 0x02, 0x43, 0x4E, 0x31, 0x0B, 0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x08, 0x13, 0x02, 0x47,
0x44, 0x31, 0x0B, 0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x07, 0x13, 0x02, 0x53, 0x5A, 0x31, 0x10,
0x30, 0x0E, 0x06, 0x03, 0x55, 0x04, 0x0A, 0x13, 0x07, 0x53, 0x41, 0x4E, 0x47, 0x46, 0x4F, 0x52,
0x31, 0x0C, 0x30, 0x0A, 0x06, 0x03, 0x55, 0x04, 0x0B, 0x13, 0x03, 0x53, 0x53, 0x4C, 0x31, 0x10,
0x30, 0x0E, 0x06, 0x03, 0x55, 0x04, 0x03, 0x13, 0x07, 0x73, 0x61, 0x6E, 0x67, 0x66, 0x6F, 0x72,
0x31, 0x22, 0x30, 0x20, 0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x09, 0x01, 0x16,
0x13, 0x73, 0x61, 0x6E, 0x67, 0x66, 0x6F, 0x72, 0x40, 0x73, 0x61, 0x6E, 0x67, 0x66, 0x6F, 0x72,
0x2E, 0x63, 0x6F, 0x6D, 0x82, 0x09, 0x00, 0x9D, 0x1D, 0x78, 0xA7, 0x1F, 0xAC, 0xEE, 0xA3, 0x30,
0x0A, 0x06, 0x08, 0x2A, 0x81, 0x1C, 0xCF, 0x55, 0x01, 0x83, 0x75, 0x03, 0x48, 0x00, 0x30, 0x45,
0x02, 0x21, 0x00, 0xAF, 0x31, 0xCD, 0x66, 0xB9, 0x81, 0x85, 0x34, 0x17, 0xC3, 0x75, 0x6B, 0xE6,
0xFA, 0x5F, 0xD6, 0x57, 0x94, 0xC2, 0x2A, 0x31, 0xB3, 0x06, 0xC1, 0x84, 0xB6, 0xDC, 0x64, 0xF1,
0x47, 0x50, 0x42, 0x02, 0x20, 0x3F, 0xE1, 0x92, 0x66, 0x2A, 0xCB, 0xD2, 0x9D, 0xBE, 0x89, 0x99,
0x98, 0x7E, 0xA2, 0x57, 0xB8, 0x65, 0xCB, 0x61, 0xE7, 0xC3, 0xD5, 0x8D, 0x08, 0xC5, 0xB3, 0x9D,
0x3E, 0x59, 0x27, 0xB6, 0x5F
};

/**
 * show info
 */
LOG_INFO(mclog, -1, "\n%p", "sm2 import certificate",
    "app", "%s", app,
    "ctn", "%s", ctn,
    "type", "%s", cert_type,
    NULL);

/**
 * get device handle
 */
// LOG_INFO(symbol->mclog, -1, "%s perform start\n", algo_name);
hdev = (DEVHANDLE)instance->values->get(instance->values, SKF_DEVICE_HANDLE);
if (!hdev) {
    ret = SAR_INVALIDHANDLEERR;
    LOG_ERROR(symbol->mclog, -1, "invalid device handle\n");
    goto end;
}

/**
 * open app
 */
ret = skf->SKF_OpenApplication(hdev, app, &happ);

```

```

if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_OpenApplication() failed: %#x\n", ret);
    goto end;
}

/**
 * open container
 */
ret = skf->SKF_OpenContainer(happ, ctn, &hctn);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_OpenContainer() failed: %#x\n", ret);
    goto end;
}

/**
 * import certificate
 */
ret = skf->SKF_ImportCertificate(hctn, 1, cert, sizeof(cert));
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_ImportCertificate() failed: %#x\n", ret);
    goto end;
}
LOG_INFO(symbol->mclog, -1, "sm2 import certificate [g]succ[n]\n");

end:
if (hctn) skf->SKF_CloseContainer(hctn);
if (happ) skf->SKF_CloseApplication(happ);
return ret;
}

```

签名验签

相关接口

7.6.13 ECC 签名

函数原型 `ULONG DEVAPI SKF_ECCSignData (HCONTAINER hContainer, BYTE *pbData, ULONG ulDataLen, PECCSIGNATUREBLOB pSignature)`

功能描述 ECC 数字签名。采用 ECC 算法和指定私钥 hKey，对指定数据 pbData 进行数字签名。签名后的结果存放到 pSignature 中。

参数

hContainer	[IN] 密钥容器句柄。
pbData	[IN] 待签名的数据。
ulDataLen	[IN] 待签名数据长度，必须小于密钥模长。
pSignature	[OUT] 签名值。

返回值

SAR_OK:	成功。
其他:	错误码。

备注 权限要求: 需要用户权限。

输入数据为待签数据的杂凑值。当使用 SM2 算法时，该输入数据为待签数据经过 SM2 签名预处理的结果，预处理过程遵循《公钥密码基础设施应用技术体系 SM2 算法密码使用规范》。

7. 6. 7 RSA 验签

函数原型 ULONG DEVAPI SKF_RSASVerify (DEVHANDLE hDev , RSAPUBLICKEYBLOB*
 pRSAPubKeyBlob, BYTE *pbData, ULONG ulDataLen, BYTE *pbSignature, ULONG

	ulSignLen)	
功能描述	验证 RSA 签名。用 pRSAPubKeyBlob 内的公钥值对待验签数据进行验签。	
参数	hDev	[IN] 设备句柄。
	pRSAPubKeyBlob	[IN] RSA 公钥数据结构。
	pbData	[IN] 待验证签名的数据。
	ulDataLen	[IN] 数据长度，应不大于公钥模长-11。
	pbSignature	[IN] 待验证的签名值。
	ulSignLen	[IN] 签名值长度，必须为公钥模长。
返回值	SAR OK:	成功。

权限要求

▲ 需要用户权限，设备权限提升使用 SKF_VerifyPIN 接口

参考代码

```
int asym_sign_verify(core_t *core, settings_t *settings, menu_t *menu)
{
    int ret = 0;
    int retry = 0;
    int ctn_type = 0;
    char *pin = "111111";
    char *app = "DEFAULT";
    char *ctn = "DEFAULT";
    char *type = "sm2";
    int bits = 1024;
    int algo = SGD_RSA;
    int pubkeylen = 2048;
    api_t *api = core->get_api(core);
    api_symbol_t *symbol = api->get_symbol(api);
    api_instance_t *instance = api->get_instance(api);
    menu_controllor_t *controllor = menu->get_controllor(menu);
    settings_t *property = controllor->get_property(controllor);
    unsigned char data[32] = {0};
    unsigned char pubkey[2048] = {0};
    unsigned char sign_data[512] = {0};
    unsigned int sign_data_len = sizeof(sign_data);
    mclog_api mclog = symbol->mclog;
    SKF_FUNCLIST *skf = instance->skf;
```

```

DEVHANDLE hdev = (DEVHANDLE)NULL;
HAPPLICATION happ = (HAPPLICATION)NULL;
HCONTAINER hctn = (HCONTAINER)NULL;

/**
 * show info
 */
if (algo == SGD_SM2_1) bits = 256;
LOG_INFO(symbol->mclog, -1, "\n%p",
          "%s %d sign and verify", type, bits,
          "app", "%s", app,
          "ctn", "%s", ctn,
          "type", "%s", type,
          "bits", "%d", bits,
          NULL);

/**
 * get device handle
 */
hdev = (DEVHANDLE)instance->values->get(instance->values, SKF_DEVICE_HANDLE);
if (!hdev) {
    ret = SAR_INVALIDHANDLEERR;
    LOG_ERROR(symbol->mclog, -1, "invalid device handle\n");
    goto end;
}

/**
 * open app
 */
ret = skf->SKF_OpenApplication(hdev, app, &happ);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_OpenApplication() failed: %#x\n", ret);
    goto end;
}

/**
 * verify pin
 */
ret = skf->SKF_VerifyPIN(happ, USER_TYPE, pin, (void *)&retry);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_VerifyPIN() failed: %#x\n", ret);
    goto end;
}

/**
 * open container
 */
ret = skf->SKF_OpenContainer(happ, ctn, &hctn);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_OpenContainer() failed: %#x\n", ret);
    goto end;
}

/**
 * get controller type
 */
ret = skf->SKF_GetContainerType(hctn, (void *)&ctn_type);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_GetContainerType() failed: %#x\n", ret);
    goto end;
}

```

```
}

/**
 * generate keypair
 */
if (type == SGD_RSA) {
    if (ctn_type == CTNF_ECC) {
        skf->SKF_CloseContainer(hctn);
        memset(&hctn, 0, sizeof(hctn));

        ret = skf->SKF_DeleteContainer(happ, ctn);
        if (ret) {
            LOG_ERROR(symbol->mclog, -1, "SKF_DeleteContainer() failed: %#x\n", ret);
            goto end;
        }

        ret = skf->SKF_CreateContainer(happ, ctn, &hctn);
        if (ret) {
            LOG_ERROR(symbol->mclog, -1, "SKF_CreateContainer() failed: %#x\n", ret);
            goto end;
        }
    }

    ret = skf->SKF_ExportPublicKey(hctn, 1, (void *)pubkey, &pubkeylen);
    if (ret) {
        ret = skf->SKF_GenRSAKeyPair(hctn, bits, (void *)pubkey);
        if (ret) {
            LOG_ERROR(symbol->mclog, -1, "SKF_GenRSAKeyPair() failed: %#x\n", ret);
            goto end;
        }
    }

    ret = skf->SKF_RSASignData(hctn, data, sizeof(data), sign_data, &sign_data_len);
    if (ret) {
        LOG_ERROR(symbol->mclog, -1, "SKF_RSASignData() failed: %#x\n", ret);
        goto end;
    }

    ret = skf->SKF_RSAVerify(hdev, (void *)pubkey, data, sizeof(data), sign_data, sign_data_len);
    if (ret) {
        LOG_ERROR(symbol->mclog, -1, "SKF_RSAVerify() failed: %#x\n", ret);
        goto end;
    }
} else {
    if (ctn_type == CTNF_RSA) {
        skf->SKF_CloseContainer(hctn);
        memset(&hctn, 0, sizeof(hctn));

        ret = skf->SKF_DeleteContainer(happ, ctn);
        if (ret) {
            LOG_ERROR(symbol->mclog, -1, "SKF_DeleteContainer() failed: %#x\n", ret);
            goto end;
        }

        ret = skf->SKF_CreateContainer(happ, ctn, &hctn);
        if (ret) {
            LOG_ERROR(symbol->mclog, -1, "SKF_CreateContainer() failed: %#x\n", ret);
            goto end;
        }
    }
}
```

```
ret = skf->SKF_ExportPublicKey(hctn, 1, (void *)pubkey, &pubkeylen);
if (ret) {
    ret = skf->SKF_GenECCKeyPair(hctn, SGD_SM2_1, (void *)pubkey);
    if (ret) {
        LOG_ERROR(symbol->mclog, -1, "SKF_GenECCKeyPair() failed: %#x\n", ret);
        goto end;
    }
}

ret = skf->SKF_ECCTSignData(hctn, data, sizeof(data), (void *)sign_data);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_ECCTSignData() failed: %#x\n", ret);
    goto end;
}

ret = skf->SKF_ECCTVerify(hdev, (void *)pubkey, data, sizeof(data), (void *)sign_data);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_ECCTVerify() failed: %#x\n", ret);
    goto end;
}
}
LOG_INFO(mclog, -1, "asym \\[%s] sign and verify [g]succ[n]\n", type);

end:
if (hctn) skf->SKF_CloseContainer(hctn);
if (happ) skf->SKF_CloseApplication(happ);
return ret;
}
```

对称算法运算

相关接口

7.6.25 明文导入会话密钥

函数原型	ULONG DEVAPI SKF_SetSymmKey (DEVHANDLE hDev, BYTE* pbKey, ULONG ulAlgID, HANDLE* phKey)
功能描述	设置明文对称密钥，返回密钥句柄。
参数	hDev [IN] 设备句柄。 pbKey [IN] 指向会话密钥值的缓冲区。 ulAlgID [IN] 会话密钥算法标识。 phKey [OUT] 返回会话密钥句柄。
返回值	SAR_OK: 成功。 其他: 错误码。

7.6.26 加密初始化

函数原型	ULONG DEVAPI SKF_EncryptInit (HANDLE hKey, BLOCKCIPHERPARAM EncryptParam)	
功能描述	数据加密初始化。设置数据加密的算法相关参数。	
参数	hKey	[IN] 加密密钥句柄。
	EncryptParam	[IN] 分组密码算法相关参数：初始向量、初始向量长度、填充方法、反馈值的位长度。
返回值	SAR_OK:	成功。
	其他:	错误码。

7.6.27 单组数据加密

函数原型	ULONG DEVAPI SKF_Encrypt (HANDLE hKey, BYTE * pbData, ULONG ulDataLen, BYTE *pbEncryptedData, ULONG *pulEncryptedLen)	
功能描述	单一分组数据的加密操作。用指定加密密钥对指定数据进行加密，被加密的数据只包含一个分组，加密后的密文保存到指定的缓冲区中。SKF_Encrypt 只对单个分组数据进行加密，在调用 SKF_Encrypt 之前，必须调用 SKF_EncryptInit 初始化加密操作。SKF_Encrypt 等价于先调用 SKF_EncryptUpdate 再调用 SKF_EncryptFinal。	
参数	hKey	[IN] 加密密钥句柄。
	pbData	[IN] 待加密数据。
	ulDataLen	[IN] 待加密数据长度。
	pbEncryptedData	[OUT] 加密后的数据缓冲区指针，可以为 NULL，用于获得加密后数据长度。
	pulEncryptedLen	[IN, OUT] 输入时表示结果数据缓冲区长度，输出时表示结果数据实际长度。
返回值	SAR_OK:	成功。
	其他:	错误码。

7.6.30 解密初始化

函数原型	ULONG DEVAPI SKF_DecryptInit (HANDLE hKey, BLOCKCIPHERPARAM DecryptParam)	
功能描述	数据解密初始化，设置解密密钥相关参数。调用 SKF_DecryptInit 之后，可以调用 SKF_Decrypt 对单个分组数据进行解密，也可以多次调用 SKF_DecryptUpdate 之后再调用 SKF_DecryptFinal 完成对多个分组数据的解密。	
参数	hKey	[IN] 解密密钥句柄。
	DecryptParam	[IN] 分组密码算法相关参数：初始向量、初始向量长度、填充方法、反馈值的位长度。
返回值	SAR_OK:	成功。

7. 6. 31 单组数据解密

函数原型	ULONG DEVAPI SKF_Decrypt(HANDLE hKey, BYTE * pbEncryptedData, ULONG ulEncryptedLen, BYTE * pbData, ULONG * pulDataLen)	
功能描述	单个分组数据的解密操作。用指定解密密钥对指定数据进行解密，被解密的数据只包含一个分组，解密后的明文保存到指定的缓冲区中。SKF_Decrypt 只对单个分组数据进行解密，在调用 SKF_Decrypt 之前，必须调用 SKF_DecryptInit 初始化解密操作。SKF_Decrypt 等价于先调用 SKF_DecryptUpdate 再调用 SKF_DecryptFinal。	
参数	hKey	[IN] 解密密钥句柄。
	pbEncryptedData	[IN] 待解密数据。
	ulEncryptedLen	[IN] 待解密数据长度。
	pbData	[OUT] 指向解密后的数据缓冲区指针，当为 NULL 时可获得解密后的数据长度。
	pulDataLen	[IN, OUT] 输入时表示结果数据缓冲区长度，输出时表示结果数据实际长度。
返回值	SAR_OK:	成功。
	其他:	错误码。

使用须知

▲ 加解密结束后，请调用 SKF_CloseHandle 释放 SKF_SetSymmKey 接口所产生的对称句柄

参考代码

```
static int sym_test(core_t *core, settings_t *settings, menu_t *menu)
{
    int ret = 0;
    char *algo_name = "sm1";
    char *mode_name = "ecb";
    int feedbits = 0;
    int data_len = 256;
    int enable_padding = 0;
    int use_setsymmkey = 1;
    int mode = get_sym_mode(mode_name);
    int algoid = get_sym_algoid(algo_name);
    int algo = get_vgeneratekey_algo(algoid, mode);
    api_t *api = core->get_api(core);
    api_symbol_t *symbol = api->get_symbol(api);
    api_instance_t *instance = api->get_instance(api);
    SKF_FUNCLIST *skf = instance->skf;
    unsigned char key[32] = {1};
    unsigned char iv[32] = {2};
    unsigned int ivlen = get_sym_ivlen(algoid, mode);
    unsigned char *data = NULL;
    unsigned char *encrypt_data = NULL;
    unsigned int encrypt_data_len = data_len + 1024;
    unsigned char *decrypt_data = NULL;
    unsigned int decrypt_data_len = data_len + 1024;
    BLOCKCIPHERPARAM bp;
    DEVHANDLE hdev;
    HANDLE hkey;

    /**
```

```

    * get device handle
    */
memset(&hdev, 0, sizeof(hdev));
hdev = (DEVHANDLE)instance->values->get(instance->values, SKF_DEVICE_HANDLE);
if (!hdev) {
    ret = SAR_INVALIDHANDLEERR;
    LOG_ERROR(symbol->mclog, -1, "invalid device handle\n");
    goto end;
}

/**
 * show info
 */
LOG_INFO(symbol->mclog, -1, "\n%p", "symmetric encrypt and decrypt test",
    "algo",      "%s", algo_name,
    "mode",      "%s", mode_name,
    "feedbits",  "%d", feedbits,
    "data_len",  "%d", data_len,
    "padding",   "%s", enable_padding ? "true" : "false",
    NULL);

/**
 * malloc memory
 */
data = malloc(data_len);
encrypt_data = malloc(encrypt_data_len);
decrypt_data = malloc(decrypt_data_len);
if (!data || !encrypt_data || !decrypt_data) {
    ret = SAR_INDATAERR;
    LOG_ERROR(symbol->mclog, -1, "malloc memory for data failed\n", ret);
    goto end;
}
memset(data, 1, 32);

/**
 * set key
 */
memset(&hkey, 0, sizeof(hkey));
if (!use_setsymmkey) {
    ret = skf->V_GenerateKey(hdev, algo, &hkey);
    if (ret) {
        LOG_ERROR(symbol->mclog, -1, "V_GenerateKey() failed(%#x)\n", ret);
        goto end;
    }
} else {
    algo = algoid + mode;
    ret = skf->SKF_SetSymmKey(hdev, key, algo, &hkey);
    if (ret) {
        LOG_ERROR(symbol->mclog, -1, "SKF_SetSymmKey() failed(%#x)\n", ret);
        goto end;
    }
}

/**
 * encrtypt init
 */
if (mode != MODE_CFB && mode != MODE_OFB) feedbits = 0;
memset(&bp, 0, sizeof(bp));
bp.FeedBitLen = feedbits;
if (enable_padding) {

```

```

    bp.PaddingType = PKCS5_PADDING;
}
if (mode != MODE_ECB) {
    bp.IVLen = ivlen;
    memcpy(bp.IV, iv, ivlen);
}
ret = skf->SKF_EncryptInit(hkey, bp);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_EncryptInit() failed(%#x)\n", ret);
    goto end;
}

/**
 * encrypt init
 */
ret = skf->SKF_Encrypt(hkey, data, data_len, encrypt_data, (void *)&encrypt_data_len);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_EncryptInit() failed(%#x)\n", ret);
    goto end;
}

/**
 * decrypt init
 */
ret = skf->SKF_DecryptInit(hkey, bp);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_DecryptInit() failed(%#x)\n", ret);
    goto end;
}

/**
 * decrypt
 */
ret = skf->SKF_Decrypt(hkey, encrypt_data, encrypt_data_len, decrypt_data, (void *)&decrypt_data_len);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_Decrypt() failed(%#x)\n", ret);
    goto end;
}

/**
 * compare to standard data
 */
if (data_len != decrypt_data_len ||
    memcmp(data, decrypt_data, decrypt_data_len)) {
    ret = SAR_INDATAERR;
    LOG_ERROR(symbol->mclog, -1, "decrypt data and original data is not the same!\n");
    goto end;
}

LOG_INFO(symbol->mclog, -1, "%s %s encrypt and decrypt test [g]succ[n]!\n", algo_name, mode_name);
end:
if (hkey) skf->SKF_CloseHandle(hkey);
FREE_IF(data);
FREE_IF(encrypt_data);
FREE_IF(decrypt_data);
return ret;
}

```

杂凑算法运算

相关接口

7.6.34 密码杂凑初始化

函数原型	ULONG DEVAPI SKF_DigestInit(DEVHANDLE hDev, ULONG ulAlgID, ECCPUBLICKEYBLOB *pPubKey, unsigned char *pucID, ULONG ulIDLen, HANDLE *phHash)
功能描述	初始化密码杂凑计算操作，指定计算密码杂凑的算法。
参数	<div>hDev [IN] 连接设备时返回的设备句柄。</div> <div>ulAlgID [IN] 密码杂凑算法标识。</div> <div>pPubKey [IN] 签名者公钥。当 alAlgID 为 SGD_SM3 时有效。</div> <div>pucID [IN] 签名者的 ID 值，当 alAlgID 为 SGD_SM3 时有效。</div> <div>ulIDLen [IN] 签名者 ID 的长度，当 alAlgID 为 SGD_SM3 时有效。</div> <div>phHash [OUT] 密码杂凑对象句柄。</div>
返回值	<div>SAR_OK: 成功。</div> <div>其他: 错误码。</div>
备注	当 ulAlgID 为 SGD_SM3 且 ulIDLen 不为 0 的情况下 pPubKey、pucID 有效，执行 SM2 算法签名预处理 1 操作。计算过程遵循《公钥密码基础设施应用技术体系 SM2 算法密码使用规范》。

7.6.35 单组数据密码杂凑

函数原型	ULONG DEVAPI SKF_Digest (HANDLE hHash, BYTE *pbData, ULONG ulDataLen, BYTE *pbHashData, ULONG *pulHashLen)
功能描述	对单一分组的消息进行密码杂凑计算。调用 SKF_Digest 之前，必须调用 SKF_DigestInit 初始化密码杂凑计算操作。SKF_Digest 等价于多次调用 SKF_DigestUpdate 之后再调用 SKF_DigestFinal。
参数	<div>hHash [IN] 密码杂凑对象句柄。</div> <div>pbData [IN] 指向消息数据的缓冲区。</div> <div>ulDataLen [IN] 消息数据的长度。</div> <div>pbHashData [OUT] 密码杂凑数据缓冲区指针，当此参数为 NULL 时，由 pulHashLen 返回密码杂凑结果的长度。</div> <div>pulHashLen [IN, OUT] 输入时表示结果数据缓冲区长度，输出时表示结果数据实际长度。</div>
返回值	<div>SAR_OK: 成功。</div> <div>其他: 错误码。</div>

7.6.36 多组数据密码杂凑

函数原型	ULONG DEVAPI SKF_DigestUpdate (HANDLE hHash, BYTE *pbData, ULONG ulDataLen)
功能描述	对多个分组的消息进行密码杂凑计算。调用 SKF_DigestUpdate 之前，必须调用 SKF_DigestInit 初始化密码杂凑计算操作；调用 SKF_DigestUpdate 之后，必须调用 SKF_DigestFinal 结束密码杂凑计算操作。
参数	hHash [IN] 密码杂凑对象句柄。 pbData [IN] 指向消息数据的缓冲区。 ulDataLen [IN] 消息数据的长度。
返回值	SAR_OK: 成功。 其他: 错误码。

7.6.37 结束密码杂凑

函数原型	ULONG DEVAPI SKF_DigestFinal (HANDLE hHash, BYTE *pHashData, ULONG *pulHashLen)
功能描述	结束多个分组消息的密码杂凑计算操作，将密码杂凑结果保存到指定的缓冲区。
参数	hHash [IN] 密码杂凑对象句柄。 pHashData [OUT] 返回的密码杂凑结果缓冲区指针，如果此参数 NULL 时，由 pulHashLen 返回杂凑结果的长度。 pulHashLen [IN, OUT] 输入时表示杂凑结果缓冲区的长度，输出时表示密码杂凑结果的长度。
返回值	SAR_OK: 成功。 其他: 错误码。
备注	SKF_DigestFinal 必须用于 SKF_DigestUpdate 之后。

参考代码

```
static int hash_test(core_t *core, settings_t *settings, menu_t *menu)
{
    int ret = 0;
    int twice_len = 0;
    int file_size = 0;
    int data_len = 1025;
    int enable_log = 1;
    int enable_sm3_check = 1;
    char *algo_name = "sm3";
    char *id = NULL;
    int algo = get_hash_algoid(algo_name);
    api_t *api = core->get_api(core);
    api_symbol_t *symbol = api->get_symbol(api);
    api_instance_t *instance = api->get_instance(api);
    SKF_FUNC_LIST *skf = instance->skf;
    unsigned char *data = NULL;
    unsigned char hash_data[512] = {0};
    unsigned char twice_hash_data[512] = {0};
    unsigned char soft_hash_data[512] = {0};
    unsigned int hash_data_len = sizeof(hash_data);
    unsigned int twice_hash_data_len = sizeof(twice_hash_data);
    DEVHANDLE hdev = (DEVHANDLE) NULL;
```

```

HANDLE hhash = (HANDLE)NULL;
HANDLE hhash2 = (HANDLE)NULL;
FILE *fp = NULL;
FILE *result_fp = NULL;
u8 keypair[] = {
    /* pub x */
    0xB4, 0xD2, 0xE9, 0xF8, 0xFF, 0xE8, 0x67, 0xCF, 0xA6, 0x6D, 0x7E, 0xDE, 0xDE, 0xD1, 0x3E, 0x3E,
    0xE3, 0x3B, 0x7A, 0xF6, 0xD8, 0xED, 0xDD, 0x21, 0xB0, 0xF6, 0xC4, 0xBC, 0x4D, 0x4A, 0x05, 0xF7,
    /* pub y */
    0x99, 0xC5, 0x28, 0x3A, 0xD1, 0xAF, 0x78, 0x77, 0x4F, 0xFF, 0x5F, 0x57, 0xE0, 0x24, 0xE1, 0x4A,
    0x77, 0xF5, 0x1E, 0x93, 0x8D, 0x39, 0xD5, 0xC7, 0x1E, 0x67, 0x12, 0x87, 0x93, 0x65, 0x5D, 0xBA,
    /* priv */
    0xF0, 0xE7, 0x9B, 0x09, 0x3F, 0x8E, 0x64, 0x12, 0x79, 0x02, 0x90, 0x2C, 0x65, 0xC5, 0x64, 0xB8,
    0xBC, 0xFB, 0xD4, 0xD4, 0xD0, 0x26, 0x55, 0x3A, 0x5C, 0x7A, 0x64, 0x6B, 0xB1, 0x84, 0x0B, 0x6F
};
ECCPUBLICKEYBLOB sm2_pubkey = {0};

/**
 * get device handle
 */
memset(&hdev, 0, sizeof(hdev));
hdev = (DEVHANDLE)instance->values->get(instance->values, SKF_DEVICE_HANDLE);
if (!hdev) {
    ret = SAR_INVALIDHANDLEERR;
    LOG_ERROR(symbol->mclog, -1, "invalid device handle\n");
    goto end;
}

/**
 * show info
 */
if (enable_log) {
    LOG_INFO(symbol->mclog, -1, "\n%p", "hash calc test",
        "algo", "%s", algo_name,
        "data_len", "%d", data_len,
        NULL);
}

/**
 * get data
 */
data = malloc(data_len);
if (!data) {
    ret = SAR_NO_ROOM;
    LOG_ERROR(symbol->mclog, -1, "malloc memory for data failed!\n", ret);
    goto end;
}

/**
 * digest init
 */
memset(&sm2_pubkey, 0, sizeof(sm2_pubkey));
sm2_pubkey.BitLen = 256;
memcpy(sm2_pubkey.XCoordinate + 32, keypair, 32);
memcpy(sm2_pubkey.YCoordinate + 32, keypair + 32, 32);
if (id && strlen(id)) {
    ret = skf->SKF_DigestInit(hdev, algo, &sm2_pubkey, (void *)id, strlen(id), &hhash);
} else {
    ret = skf->SKF_DigestInit(hdev, algo, NULL, NULL, 0, &hhash);
}

```

```
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_DigestInit() failed(%#x)\n", ret);
    goto end;
}

/**
 * hash calc
 */
ret = skf->SKF_Digest(hhash, data, data_len, hash_data, (void *)&hash_data_len);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_Digest() failed(%#x)\n", ret);
    goto end;
}
skf->SKF_CloseHandle(hhash);
hhash = NULL;
if (enable_log) {
    LOG_INFO(symbol->mclog, -1, "hash once result\n%#h", 16, hash_data_len, hash_data);
}

/**
 * soft calc check
 */
if (algo == SGD_SM3 && enable_sm3_check && !id) {
    ret = hash_soft_calc(algo, data, data_len, soft_hash_data);
} else {
    ret = -1;
}
if (!ret) {
    if (enable_log) {
        LOG_INFO(symbol->mclog, -1, "hash soft result\n%#h", 16, hash_data_len, soft_hash_data);
    }
    if (memcmp(hash_data, soft_hash_data, hash_data_len)) {
        ret = SAR_FAIL;
        LOG_ERROR(symbol->mclog, -1, "sm3 result is not same with soft algo calc result!\n");
        goto end;
    }
}
ret = 0;

/**
 * twice calc
 */
if (data_len > 16) {
    twice_len = gen_random_index();
    twice_len = twice_len % (data_len - 16);
}
if (twice_len) {
    /**
     * digest init
     */
    if (id && strlen(id)) {
        ret = skf->SKF_DigestInit(hdev, algo, &sm2_pubkey, (void *)id, strlen(id), &hhash2);
    } else {
        ret = skf->SKF_DigestInit(hdev, algo, NULL, NULL, 0, &hhash2);
    }
    if (ret) {
        LOG_ERROR(symbol->mclog, -1, "SKF_DigestInit() failed(%#x)\n", ret);
        goto end;
    }
}
```

```

/**
 * update
 */
ret = skf->SKF_DigestUpdate(hhash2, data, twice_len);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_DigestUpdate() failed(%#x)\n", ret);
    goto end;
}

/**
 * update
 */
ret = skf->SKF_DigestUpdate(hhash2, data + twice_len, data_len - twice_len);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_DigestUpdate() failed(%#x)\n", ret);
    goto end;
}

/**
 * final
 */
ret = skf->SKF_DigestFinal(hhash2, twice_hash_data, &twice_hash_data_len);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_DigestFinal() failed(%#x)\n", ret);
    goto end;
}
skf->SKF_CloseHandle(hhash2);
hhash2 = NULL;
if (enable_log) {
    LOG_INFO(symbol->mclog, -1, "hash twice result\n%#h", 16, hash_data_len, hash_data);
}

if (twice_hash_data_len != hash_data_len ||
    memcmp(hash_data, twice_hash_data, hash_data_len)) {
    ret = SAR_FAIL;
    LOG_DEBUG(symbol->mclog, -1, "hash once data\n%#h", 16, hash_data_len, hash_data);
    LOG_DEBUG(symbol->mclog, -1, "hash twice data\n%#h", 16, twice_hash_data_len, twice_hash_data);
    LOG_ERROR(symbol->mclog, -1, "one and twice hash data is not the same!\n", ret);
    goto end;
}
if (enable_log) {
    LOG_INFO(symbol->mclog, -1, "hash once and twice result is the same! right!\n");
}
}

if (enable_log) {
    LOG_INFO(symbol->mclog, -1, "%s calc [g]succ[n]!\n", algo_name);
}

end:
if (hhash) skf->SKF_CloseHandle(hhash);
if (hhash2) skf->SKF_CloseHandle(hhash2);
FREE_IF(data);
if (fp) fclose(fp);
if (result_fp) fclose(result_fp);
return ret;
}

```

对应接口

7.3.2 创建应用

函数原型	ULONG WINAPI SKF_CreateApplication(DEVHANDLE hDev, LPSTR szAppName, LPSTR szAdminPin, DWORD dwAdminPinRetryCount, LPSTR szUserPin, DWORD dwUserPinRetryCount, DWORD dwCreateFileRights, HAPPLICATION *phApplication)		
功能描述	创建一个应用。		
参数	hDev	[IN]	连接设备时返回的设备句柄。
	szAppName	[IN]	应用名称。
	szAdminPin	[IN]	管理员 PIN。
	dwAdminPinRetryCount	[IN]	管理员 PIN 最大重试次数。
	szUserPin	[IN]	用户 PIN。
	dwUserPinRetryCount	[IN]	用户 PIN 最大重试次数。
	dwCreateFileRights	[IN]	在该应用下创建文件和容器的权限，参见 6.4.9 权限类型。为各种权限的或值。
	phApplication	[OUT]	应用的句柄。
返回值	SAR_OK:	成功。	
	其他:	错误码。	
备注	权限要求：需要设备权限。		

权限要求

▲ 需要设备权限，设备权限提升参考设备认证章节

参考代码

```
static int app_create(core_t *core, settings_t *settings, menu_t *menu)
{
    int ret = 0;
    char *app = "test_app";
    char *so_pin = "so_pin", "111111";
    char *user_pin = "111111";
    int retry = 6;
    api_t *api = core->get_api(core);
    api_symbol_t *symbol = api->get_symbol(api);
    api_instance_t *instance = api->get_instance(api);
    mclog_api mclog = symbol->mclog;
    SKF_FUNCLIST *skf = instance->skf;
    HAPPLICATION happ;
    DEVHANDLE hdev;

    /**
     * get device handle
     */
    memset(&hdev, 0, sizeof(hdev));
    memset(&happ, 0, sizeof(happ));
    hdev = (DEVHANDLE)instance->values->get(instance->values, SKF_DEVICE_HANDLE);
    if (!hdev) {
        ret = SAR_INVALIDHANDLEERR;
        LOG_ERROR(symbol->mclog, -1, "invalid device handle\n");
        goto end;
    }
}
```

```

}

/**
 * show info
 */
LOG_INFO(mclog, -1, "\n%p", "create application",
         "app", "%s", app,
         "len", "%d", strlen(app),
         "so_pin", "%s", so_pin,
         "user_pin", "%s", user_pin,
         NULL);

/**
 * create app
 */
ret = skf->SKF_CreateApplication(hdev, app, so_pin, retry, user_pin, retry, 0, &happ);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "SKF_CreateApplication() failed(%#x)\n", ret);
    goto end;
}

end:
if (happ) {
    LOG_INFO(mclog, -1, "create application \[%s] [g]succ[n]\n", app);
    skf->SKF_CloseApplication(happ);
}
return ret;
}

```

创建容器

相关接口

7.5.2 创建容器

函数原型	ULONG DEVAPI SKF_CreateContainer (HAPPLICATION hApplication, LPSTR szContainerName, HCONTAINER *phContainer)
功能描述	在应用下建立指定名称的容器并返回容器句柄。
参数	hApplication [IN] 应用句柄。 szContainerName [IN] ASCII 字符串，表示所建立容器的名称，容器名称的最大长度不能超过 64 字节。 phContainer [OUT] 返回所建立容器的容器句柄。
返回值	SAR_OK: 成功。 其他: 错误码。

备注

权限要求：需要用户权限。

权限要求

▲ 需要用户权限，设备权限提升使用 SKF_VerifyPIN 接口

参考代码

```

static int ctn_create(core_t *core, settings_t *settings, menu_t *menu)
{
    int ret = 0;
    int retry = 0;
    char *pin = "111111";
    char *app = "DEFAULT";
    char *ctn = "test_ctn";
    api_t *api = core->get_api(core);
    api_symbol_t *symbol = api->get_symbol(api);
    api_instance_t *instance = api->get_instance(api);
    menu_controllor_t *controllor = menu->get_controllor(menu);
    settings_t *property = controllor->get_property(controllor);
    mclog_api mclog = symbol->mclog;
    SKF_FUNCLIST *skf = instance->skf;
    DEVHANDLE hdev = (DEVHANDLE)NULL;
    HAPPLICATION happ = (HAPPLICATION)NULL;
    HCONTAINER hctn = (HCONTAINER)NULL;

    /**
     * get device handle
     */
    hdev = (DEVHANDLE)instance->values->get(instance->values, SKF_DEVICE_HANDLE);
    if (!hdev) {
        ret = SAR_INVALIDHANDLEERR;
        LOG_ERROR(symbol->mclog, -1, "invalid device handle\n");
        goto end;
    }

    /**
     * show info
     */
    LOG_INFO(mclog, -1, "\n%p", "create container",
             "application", "%s", app,
             "container", "%s", ctn,
             "len", "%d", strlen(ctn),
             NULL);

    /**
     * open app
     */
    ret = skf->SKF_OpenApplication(hdev, app, &happ);
    if (ret) {
        LOG_ERROR(symbol->mclog, -1, "SKF_OpenApplication() failed: %#x\n", ret);
        goto end;
    }

    /**
     * verify pin
     */
    ret = skf->SKF_VerifyPIN(happ, USER_TYPE, pin, (void *)&retry);
    if (ret) {
        LOG_ERROR(symbol->mclog, -1, "SKF_VerifyPIN() failed: %#x\n", ret);
        goto end;
    }

    /**
     * create container
     */
    ret = skf->SKF_CreateContainer(happ, ctn, &hctn);
    if (ret) {

```

```
LOG_ERROR(symbol->mclog, -1, "SKF_CreateContainer() failed: %#x\n", ret);
goto end;
}
LOG_INFO(mclog, -1, "create containor \\[%s] [g]succ[n]\n", ctn);

end:
if (hctn) skf->SKF_CloseContainer(hctn);
if (happ) skf->SKF_CloseApplication(happ);
return ret;
}
```

固件升级

说明

固件升级使用 `V_Control()` 接口，接口需要 `FirmWare` 结构体作为参数，该结构体定义如下：

```
typedef struct {
    /**
     * @brief 当有多设备时，可直接指定设备名称；
     *         当该值为NULL时，接口内部会自动枚举设备，并使用第一个设备节点；
     */
    char *szDevName;

    /**
     * @brief 固件升级时所需的AES密钥（长度必须为16字节），如该值为NULL，则接口内部使用默认值；
     */
    u8 *pbAesKey;

    /**
     * @brief 固件升级时所需的发布密钥（长度必须为16字节），如该值为NULL，则接口内部使用默认值；
     */
    u8 *pbReleaseKey;

    /**
     * @brief 固件数据，不能为空；
     */
    u8 *pbFirmWare;

    /**
     * @brief 固件数据长度，长度必须大于0；
     */
    u32 cbFirmWare;

    /**
     * @brief 固件下载过程中需要设备掉电和上电的回调函数，不能为空；
     */
    int (*repower) (void *arg);

    /**
     * @brief 固件下载进度条回调函数，可以为空；
     */
    void (*download_progress_cb) (int progress, void *arg);

    /**
     * @brief 掉电和上电、进度条回调函数所需的参数，可以为NULL；
     */
    void *arg;
} FirmWare;
```

结构体参数说明:

参数	说明
szDevName	<ul style="list-style-type: none">不为NULL时，接口内使用该指定设备名称；为NULL时，接口内部会自动枚举设备，并使用第一个设备节点；
pbAesKey	<ul style="list-style-type: none">固件升级时所需的AES密钥（长度必须为16字节）；如该值为NULL，则接口内部会自动使用默认值；
pbReleaseKey	<ul style="list-style-type: none">固件升级时所需的发布密钥（长度必须为16字节）；如该值为NULL，则接口内部会自动使用默认值；
pbFirmWare	<ul style="list-style-type: none">固件数据，不能为空；
cbFirmWare	<ul style="list-style-type: none">固件数据长度（必须大于0）；
repower	<ul style="list-style-type: none">固件下载过程中需要设备掉电和上电的回调函数，不能为空；
download_progress_cb	<ul style="list-style-type: none">固件下载进度条回调函数，可以为NULL；
arg	<ul style="list-style-type: none">掉电和上电、进度条回调函数所需的参数，可以为NULL；

参考代码

```
static int repower(void *arg)
{
    printf("please shut down device\n");
    sleep(10);
    printf("please repower device\n");
    sleep(10);
    return 0;
}

int download_firmware(core_t *core, settings_t *settings, menu_t *menu)
{
    int ret = 0;
    int size = 0;
    char *bin = "sd_v1.80/CCM3310S-T40_SD_COS_V1.80_20190916_release_with_sha.bin";
    api_t *api = core->get_api(core);
    api_symbol_t *symbol = api->get_symbol(api);
    api_instance_t *instance = api->get_instance(api);
    mclog_api mclog = symbol->mclog;
    SKF_FUNCLIST *skf = instance->skf;
    FirmWare firmware = {NULL};
    FILE *fp = NULL;

    /**
     * show info
     */
    LOG_INFO(mclog, -1, "\n%p", "auto download firmware",
             "name", "%s", name,
```

```
    "bin", "%s", bin,
    NULL);

/**
 * check bin
 */
if (!bin) {
    ret = SAR_FAIL;
    goto end;
}

/**
 * open bin
 */
fp = fopen(bin, "rb");
if (!fp) {
    LOG_ERROR(symbol->mclog, -1, "fopen() failed: %s\n", strerror(errno));
    ret = SAR_FAIL;
    goto end;
}
fseek(fp, 0, SEEK_END);
size = ftell(fp);
fseek(fp, 0, SEEK_SET);

/**
 * read bin
 */
memset(&firmware, 0, sizeof(firmware));
firmware.cbFirmWare = size;
firmware.repower = repower;
firmware.pbFirmWare = malloc(size + 16);
if (!firmware.pbFirmWare) {
    ret = SAR_NO_ROOM;
    LOG_ERROR(symbol->mclog, -1, "malloc() failed: %s\n", strerror(errno));
    goto end;
}
ret = fread(firmware.pbFirmWare, 1, size, fp);
if (ret < 0) {
    ret = SAR_FAIL;
    LOG_ERROR(symbol->mclog, -1, "malloc() failed: %s\n", strerror(errno));
    goto end;
}

/**
 * firmware download
 */
ret = skf->V_Control(0, V_CTL_AUTODOWNLOAD_FIRMWARE, &firmware);
if (ret) {
    LOG_ERROR(symbol->mclog, -1, "V_Control() failed: %#x\n", ret);
    goto end;
}
LOG_INFO(symbol->mclog, -1, "update firmware \\[%s] [g]succ[n]\n", bin);

/**
 * close handle
 */
end:
if (fp) fclose(fp);
return ret;
}
```

SKF使用demo

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

#ifdef _WIN32
#include <dlfcn.h>
#include <unistd.h>
#else
#include <windows.h>
#endif

PSKF_FUNCLIST FunctionList;

int load_library()
{
    int ret = 0;
    void * lib_handle = NULL;
    char path[128] = {0};

#ifdef _WIN32
    P_SKF_GetFuncList GetFunction = NULL;
    GetCurrentDirectory(MAX_PATH, path);
    strcat(path, "\\SKF.dll");
    printf("Load Dll %s\n", path);
    lib_handle = LoadLibrary(path);
    if (lib_handle==NULL)
    {
        ret = GetLastError();
        printf("Load Dll Fail:%d\n", ret);
        return ret;
    }
    else
    {
        GetFunction = (P_SKF_GetFuncList)GetProcAddress(lib_handle, "SKF_GetFuncList");
        if (GetFunction == NULL)
        {
            ret = GetLastError();
            return ret;
        }
        printf("Load Dll OK\n");
        ret = GetFunction(&FunctionList);
        if (ret) return ret;
    }
#else
    P_SKF_GetFuncList get_func_list;

    getcwd(path, sizeof(path));
    strcat(path, "/libskf.so");
    lib_handle = dlopen(path, RTLD_LAZY );
    if (!lib_handle)
    {
        printf("Open Error:%s.\n", dlerror());
        return 0;
    }

    get_func_list = dlsym(lib_handle, "SKF_GetFuncList");
    if (get_func_list == NULL)
```

```

{
    printf("Dlsym Error:%s.\n", dlerror());
    return 0;
}

ret = get_func_list(&FunctionList);
if (ret)
{
    printf("fnGetList ERROR 0x%x", ret);
    return ret;
}
#endif

return ret;
}

void print_data(char *info, unsigned char *data, unsigned int len)
{
    int i = 0;
    if (info) printf("%s\n", info);

    for (i = 0; i < len; i++) {
        if (i && i % 16 == 0) printf("\n");
        printf("0x%02x ", data[i]);
    }
    printf("\n");
}

int sym_gdb_test(HANDLE hdev, unsigned int algo, unsigned char *key, unsigned char *iv, unsigned int iv_len,
unsigned char *data, unsigned int data_len, unsigned char *gdb_data, unsigned int gdb_data_len)
{
    int ret = 0;
    HANDLE hkey;
    BLOCKCIPHERPARAM bp;
    u8 enc_data[1024] = {0};
    u32 enc_len = 0, enc_final_len = 0;
    u8 dec_data[1024] = {0};
    u32 dec_len = 0, dec_final_len = 0;

    ret = FunctionList->SKF_SetSymmKey(hdev, key, algo, &hkey);
    if (ret) {
        printf("SKF_SetSymmKey() failed: %#x\n", ret);
        return ret;
    }

    memset(&bp, 0, sizeof(bp));
    bp.IVLen = iv_len;
    if (bp.IVLen > 0) {
        if (!iv) {
            printf("sym cbc mode, but iv is null\n");
            ret = -1;
            goto end;
        }

        memcpy(bp.IV, iv, bp.IVLen);
    }

    ret = FunctionList->SKF_EncryptInit(hkey, bp);
    if (ret) {
        printf("SKF_EncryptInit() failed: %#x\n", ret);
    }
}

```

```

    goto end;
}

ret = FunctionList->SKF_EncryptUpdate(hkey, data, data_len, enc_data, &enc_len);
if (ret) {
    printf("SKF_EncryptInit() failed: %#x\n", ret);
    goto end;
}

ret = FunctionList->SKF_EncryptFinal(hkey, enc_data + enc_len, &enc_final_len);
if (ret) {
    printf("SKF_EncryptInit() failed: %#x\n", ret);
    goto end;
}
enc_len += enc_final_len;

/**
 * show original data and encrypted data
 */
print_data("original data: ", data, data_len);
print_data("encrypted data: ", enc_data, enc_len);

if (gdb_data && gdb_data_len > 0 && (enc_len != gdb_data_len || memcmp(gdb_data, enc_data, enc_len))) {
    printf("enc and gdb data is not same!\n");
    ret = -1;
    goto end;
}

ret = FunctionList->SKF_DecryptInit(hkey, bp);
if (ret) {
    printf("SKF_DecryptInit() failed: %#x\n", ret);
    goto end;
}

ret = FunctionList->SKF_DecryptUpdate(hkey, enc_data, enc_len, dec_data, &dec_len);
if (ret) {
    printf("SKF_EncryptInit() failed: %#x\n", ret);
    goto end;
}

ret = FunctionList->SKF_DecryptFinal(hkey, dec_data + dec_len, &dec_final_len);
if (ret) {
    printf("SKF_EncryptInit() failed: %#x\n", ret);
    goto end;
}
dec_len += dec_final_len;

/**
 * show encrypted data and decrypted data
 */
print_data("decrypted data: ", dec_data, dec_len);

if (enc_len != dec_len || memcmp(data, dec_data, enc_len)) {
    printf("enc and dec data is not same!\n");
    ret = -1;
} else {
    printf("sym enc and dec succ!\n");
}

end:

```

```
FunctionList->SKF_CloseHandle(hkey);
return ret;
}

int sm4_ecb_gdb_data_test(HANDLE hdev)
{
    int ret = 0;
    unsigned char key[] = {0x77,0x7f,0x23,0xc6,0xfe,0x7b,0x48,0x73,0xdd,0x59,0x5c,0xff,0xf6,0x5f,0x58,0xec};
    unsigned char data[] = {0x5f,0xe9,0x7c,0xcd,0x58,0xfe,0xd7,0xab,0x41,0xf7,0x1e,0xfb,0xfd,0xe7,0xe1,0x46};
    unsigned char gdb_data[] =
{0x56,0xda,0x23,0xe2,0x5f,0xa7,0xcd,0x82,0x5d,0x51,0xc2,0x20,0xf5,0x98,0x09,0x0b};

    ret = sym_gdb_test(hdev, SGD_SMS4_ECB, key, NULL, 0, data, sizeof(data), gdb_data, sizeof(gdb_data));
    return ret;
}

int get_device_info(HANDLE hdev)
{
    int ret = 0;
    DEVINFO info;

    memset(&info, 0, sizeof(info));
    ret = FunctionList->SKF_GetDevInfo(hdev, &info);
    if (ret) {
        printf("SKF_GetDevInfo() failed: %#x\n", ret);
        return ret;
    }

    printf("Manufacturer: %s\n", info.Manufacturer);
    printf("Issuer: %s\n", info.Issuer);
    printf("Label: %s\n", info.Label);
    printf("SerialNumber: %s\n", info.SerialNumber);

    return ret;
}

int main(int argc, char *argv[])
{
    int ret = 0;
    char devices[128] = {0};
    u32 devices_size = sizeof(devices);
    HANDLE hdev;

    ret = load_library();
    if (ret) {
        printf("load_library() failed: %#x\n", ret);
        return ret;
    }

    ret = FunctionList->SKF_EnumDev(1, devices, &devices_size);
    if (ret) {
        printf("SKF_EnumDev() failed: %#x\n", ret);
        goto end;
    }

    ret = FunctionList->SKF_ConnectDev(devices, &hdev);
    if (ret) {
        printf("SKF_ConnectDev() failed: %#x\n", ret);
        goto end;
    }
}
```

```
ret = get_device_info(hdev);
if (ret) {
    printf("get device info failed: %#x\n", ret);
    goto end;
}

ret = sm4_ecb_gdb_data_test(hdev);
if (ret) {
    printf("sym_test() failed: %#x\n", ret);
}

end:
if (hdev) ret = FunctionList->SKF_DisConnectDev(hdev);

#ifdef _WIN32
    system("pause");
#endif
return 0;
}
```

开发问题集锦

SKF_EnumDev()枚举设备时返回0x0A000023错误，无法枚举到设备怎么办？

当发出这种错误时，通过以下几种方法解决：

- usb_ms接口
 - windows平台
 1. windows平台无法枚举到设备一般是没有插入设备造成；
 - linux平台
 1. 确保程序以root权限运行；
 2. 在/dev目录下是否有sd*设备节点；
- sd接口
 - windows平台
 1. windows平台无法枚举到设备一般是没有插入设备造成；
 - linux平台
 1. 确保程序以root权限运行；
 2. sd卡是否已成功挂载，可用mount命令查看挂载情况。挂载方法可参考《[TF卡设备使用须知（非TF卡设备忽略）](#)》章节；

怎么创建应用？

- 首先，设备认证，流程及代码可参考[设备认证](#)章节；
- 然后，调用 `SKF_CreateApplication` 接口创建应用；

应用创建流程可参考[创建应用](#)章节。

怎么删除应用？

- 首先，作设备认证，流程及代码可参考[设备认证](#)章节；
- 然后，调用 `SKF_DeleteApplication` 接口删除应用；

怎么创建容器？

- 首先，使用 `SKF_VerifyPIN` 接口提升至[用户权限](#)；
- 然后，使用 `SKF_CreateContainer` 接口创建容器；

怎么删除容器？

- 首先，使用 `SKF_VerifyPIN` 接口提升至[用户权限](#)；
- 然后，使用 `SKF_DeleteContainer` 接口删除容器；

创建应用失败？

- 首先，确认创建应用之前是否已做过[设备认证](#)，并且保证设备认证后未调用除`SKF_CreateApplication`外的其他任何接口（详情请参考[设备认证-使用须知](#)章节）；
- 其次，确认APP名称是否是字符串，[不能为二进制数组](#)；
- 再者，确认 `szUserPin` 和 `szAdminPin` 参数[是不是字符串](#)，[不能为二进制数组](#)；
- 最后，确保其他参数正确；

应用创建流程可参考[创建应用](#)章节。

删除应用失败？

- 首先，确认删除应用之前是否已做过[设备认证](#)，并且保证设备认证后未调用除`SKF_DeleteApplication`外的其他任何接口（详情请参考[设备认证-使用须知](#)章节）；
- 其次，确认APP名称是否是字符串，[不能为二进制数组](#)；

SKF调用流程是怎么样的？

调用流程参考[SKF使用demo](#)章节。

SKF_SetSymmKey返回SKF_NO_ROOM怎么办？

如果`SKF_SetSymmKey`函数返回`SKF_NO_ROOM`错误，[一定要将设备断电后重新上电](#)才能再次使用。

这是因为对称算法相关资源已耗尽，代码中有多处[对称句柄没有关闭](#)导致。

因此，为避免该问题出现，[所有对称句柄使用完后一定要关闭句柄](#)，[释放对称算法资源](#)。

怎么开启SKF国密库日志？

开启方法可参考《[开启国密库日志的方法](#)》章节。

✎ 由 **anton** 更新于 Fri, Jan 17, 2020 6:57 AM

苏州国芯科技

苏州国芯科技

苏州国芯科技