银河麒麟高级服务器操作系统 V10 SP3 2403

安全SDK用户手册

麒麟软件有限公司
2024年04月

版本说明

日期	版本号	发布说明	编写者	签批人		
				角色	人员	签字
2024/03/06	V0.1	整合完成初稿	赵庆友			
2024/04/01	V0.2	根据评审意见修改，合入飞腾机密计算部分	赵庆友			
2024/04/12	V1.0	部分描述修正	赵庆友			

目录

1. 概述

银河麒麟高级服务器操作系统 V10 SP3 2403 版本现在已广泛发布，并加入了麒麟安全开发套件 的 V1.1 版本（简称 KYSDK）。对于银河麒麟操作系统开发人员来说，这将是一个激动人心的时刻，为安全生态建设与软件开发提供安全、可靠、稳定、快捷的开发接口，新的接口、工具和框架也正在积极开发中。KYSDK 聚焦于身份认证、密码服务、安全管控、应用防护、网络防护、可信计算、安全审计、接口授权等内容，旨在为尽可能多的受众提供对最新银河麒麟操作系统安全功能的支持，帮助解决安全应用与银河麒麟操作系统的兼容、适配、移植、优化等方面的问题。

本文档介绍了如何使用麒麟安全开发套件 SDK 编写应用程序。麒麟安全开发套件SDK 提供的CAPI可以在银河麒麟高级服务器操作系统 V10 SP3 2403 版本下面使用。

2. 安全SDK分类

麒麟安全开发套件 SDK 主要包含以下 API：

● LSM 安全模块 API：提供安全模块的注册和卸载操作

● 可信计算 API：提供可信计算基础应用所需通用操作

● 密码服务 API：提供密码基础服务所需通用操作

● 安全审计 API：提供系统安全审计所需操作，如：审计开关、记录审计等

● 身份认证 API：提供用户管理、用户认证、域用户统一认证和单点登录等不同能力所需的通用操作

● 安全管控 API：提供系统安全功能的策略配置能力，包括：设备管控策略、应用执行控制策略、防火墙规则、软件包安装黑白名单、以及获取审计日志等

● 应用防护 API：提供应用增强防护能力，包括：文件防篡改，进程防杀死，内核模块防卸载、软件包防卸载等

● 接口授权 API：提供对特定的安全接口提供调用授权，只允许特定的应用进行调用

3. 安全SDK 使用授权

3.1 安全功能开关状态

安全SDK中授权检查机制依赖麒麟安全kysec功能，使用前需确认kysec是否开启:

```
[root@localhost ~]# security-switch --get

当前安全级别 : 推荐(default)
-----------------------------------------------------
  安全模块      |      当前状态      |      默认状态
-----------------------------------------------------
  Kysec          启用(Enforcing)     启用(Enforcing)
```

若kysec未开启，可以使用以下命令开启kysec：

```
[root@localhost ~]# setstatus kysec -s enforcing
```

3.2 接口授权工具

安装包libkysdk-security提供接口授权工具kdk-authorize

安全SDK接口中包含有授权检查逻辑的模块有应用执行控制，应用安装控制，外设管控，软件包防卸载，文件保护，进程防杀，内核模块防卸载等。当一个程序调用某个安全sdk接口时，会查询这个程序的授权信息，是否有这个安全SDK接口的权限，如果找不到授权信息，相关调用就会因为权限不足而失败。所以在调用安全sdk接口之前，必须使用接口授权工具（kdk-authorize）对这个程序进行授权。

3.2.1 查看帮助信息

用“--help”参数可以列出接口授权工具kdk-authorize的帮助信息，具体命令如下：

```
[root@localhost ~]# kdk-authorize --help
Usage: kdk_authorize [ -l ]  [ -a / -d ] package_name [ -f ] program_filepath
        -l: show current authorization kdk program information
        -d: cancel the authorization for the kdk program package
        -a: add authorization for the kdk program package
        -p: set shared and unique permissions. all permissions are included by default. unique permissons only
can be used by one package.it will automatically delete other package's permissions. this used with '-a'
        -f: the kdk program file path, this used with '-a'
        --help: help information

eg:
        kdk-authorize -l
        kdk-authorize -d kdkA-package
        kdk-authorize -a kdkA-package -f /usr/bin/kdkA
        kdk-authorize -a kdkA-package -p appinstsec,appexectlsec,appuninstsec,devicesec -f /usr/bin/kdkA

shared permissions: appinstsec,appexectlsec,appuninstsec,filesec,kmodsec,processsec
unique permissions: devicesec
```

3.2.2 查看当前接口授权

用“-l”参数可以列出当前所有的接口授权信息，具体命令如下：

```
[root@localhost ~]# kdk-authorize -l
KDK package name: kdkA-package
KDK program path:/usr/bin/kdkA(appinstsec,appexectlsec,appuninstsec,devicesec)
enhancement_status: 1
```

3.2.3 添加接口授权

用“-a”参数可以给指定软件包添加授权，主要用来管理授权信息，将同一个包的授权文件进行集中管理。

用“-f”参数可以指定要授权的文件路径，注意要使用全路径，此参数必须和“-a”参数同时使用

用“-p”参数可以指定要授权的接口权限，注意此参数必须和“-a”参数同时使用，当只使用了“-a”参数，而没有使用“-p”参数时，默认会添加所有的接口权限。“-p”参数可以同时指定多个权限，每个权限间用“,”分割。

接口权限目前分两类，一类是共享权限，一类是独占权限，共享权限可以被多个软件包同时授权；独占权限同时只能被一个软件包授权，后面授权的软件包会抢占前面授权的软件包权限。

共享权限有：appinstsec（应用安装安全），appexectlsec（应用执行控制安全），appuninstsec（应用卸载安全），filesec（文件安全），kmodsec（内核模块安全），processsec（进程安全）

独占权限有：devicesec（设备安全）

示例：

1.给软件包kdkA-package的文件/usr/bin/kdkA添加所有接口授权

kdk-authorize -a kdkA-package -f /usr/bin/kdkA

2.给软件包kdkA-package的文件/usr/bin/kdkB添加应用安装安全和应用执行控制安全接口授权

kdk-authorize -a kdkA-package -p appinstsec,appexectlsec -f /usr/bin/kdkB

3.以实际场景为例，现在我们有个第三方软件包apptest，里面有个可执行程序/usr/bin/test，/usr/bin/test程序里调用了安全SDK的文件安全和内核模块安全接口。如果我们装上apptest软件包之后，直接执行/usr/bin/test，就会出现调用安全SDK接口失败的错误。这时候我们可以用如下命令进行授权：

kdk-authorize -a apptest -p filesec,kmodsec -f /usr/bin/test

3.2.4 删除接口授权

用“-d”参数可以删除指定软件包的所有授权信息

示例：删除kdkA-package包的所有授权

kdk-authorize -d kdkA-package

4. 安全 SDK 使用说明

4.1 安全管控接口

4.1.1 系统安全

4.1.1.1 应用执行控制

设置应用包中所有程序均可执行

- 头文件：appexectlsec/libkyappexectlsec.h
- 开发依赖包：libkysdk-appexectlsec-devel
- 运行依赖包：libkysdk-appexectlsec
- 编译参数：-lkyappexectlsec -ldl -lkysdk_module_authorize

原型	int kdk_set_app_can_exec(const char* pkgname)	
描述	设置应用包中所有程序均可执行	
参数	pkgname [in]	软件包名
返回值	0	成功
	-1	失败
备注	调用该接口需要kysec开启，并通过授权工具授予appexectlsec权限	

设置应用包中所有程序均不可执行

- 头文件：appexectlsec/libkyappexectlsec.h
- 开发依赖包：libkysdk-appexectlsec-devel
- 运行依赖包：libkysdk-appexectlsec
- 编译参数： -lkyappexectlsec -ldl -lkysdk_module_authorize

原型	int kdk_clear_app_can_exec(const char* pkgname)	
描述	设置应用包中所有程序均不可执行	
参数	pkgname [in]	软件包名
返回值	0	成功
	-1	失败
备注	调用该接口需要kysec开启，并通过授权工具授予appexectlsec权限	

4.1.1.2 应用安装控制

添加一个包到黑白名单中

- 头文件：appinstsec/libkyappinstsec.h
- 开发依赖包：libkysdk-appinstsec-devel
- 运行依赖包：libkysdk-appinstsec
- 编译参数：-lkyappinstsec -ldl -lkysdk_module_authorize

原型	int kdk_pkg_bwlist_add(int type, const char *pkgname)	
描述	添加一个包到黑白名单中	
参数	type [in]	类型，参考KDK_PKG_BWLIST_TYPE
	pkgname [in]	软件包名

返回值	0	成功
	1	软件包已经在列表
	-1	失败
备注	调用该接口需要kysec开启，并通过授权工具授予appinstsec权限	

从黑白名单中删除一个包

- 头文件：appinstsec/libkyappinstsec.h
- 开发依赖包：libkysdk-appinstsec-devel
- 运行依赖包：libkysdk-appinstsec
- 编译参数：-lkyappinstsec -ldl -lkysdk_module_authorize

原型	int kdk_pkg_bwlist_del(int type, const char *pkgname)	
描述	从黑白名单中删除一个包	
参数	type [in]	类型，参考KDK_PKG_BWLIST_TYPE
	pkgname [in]	软件包名
返回值	0	成功
	1	不存在
	-1	失败
备注	调用该接口需要kysec开启，并通过授权工具授予appinstsec权限	

获取黑白名单开关标识

- 头文件：appinstsec/libkyappinstsec.h
- 开发依赖包：libkysdk-appinstsec-devel
- 运行依赖包：libkysdk-appinstsec
- 编译参数：-lkyappinstsec -ldl -lkysdk_module_authorize

原型	int kdk_pkg_bwlist_get_mode(void)	
描述	获取黑白名单开关标识	
参数		
返回值	>=0	参考KDK_PKG_BWLIST_TYPE
	-1	失败

备注	调用该接口需要kysec开启，并通过授权工具授予appinstsec权限

设置黑白名单开关标识

- 头文件：appinstsec/libkyappinstsec.h
- 开发依赖包：libkysdk-appinstsec-devel
- 运行依赖包：libkysdk-appinstsec
- 编译参数：-lkyappinstsec -ldl -lkysdk_module_authorize

原型	int kdk_pkg_bwlist_set_model(int type)	
描述	设置黑白名单开关标识	
参数	type[in]	参考KDK_PKG_BWLIST_TYPE
返回值	0	成功
	-1	失败
备注	调用该接口需要kysec开启，并通过授权工具授予appinstsec权限	

4.1.2 外设管控接口

4.1.2.1 添加 usb 设备的管控策略

- 头文件：libkydevicesec.hpp
- 开发依赖包：libkysdk-devicesec-devel
- 运行依赖包：libkysdk-devicesec
- 编译参数：无

原型	uint32_t kdk_usbguard_appendRule(const std::string& rule_spec, uint32_t parent_id)	
描述	添加usb设备管控策略	
参数	rule_spec	待添加设备管控策略条目
	parent_id	前一条设备管控策略条目id, 默认值usbguard::Rule::LastID
返回值	>0	新添加管控条目id
	其他	失败
备注	调用该接口需要kysec开启，并通过授权工具授予devicesec权限	

4.1.2.2 删除 usb 设备的管控策略

- 头文件：libkydevicesec.hpp
- 开发依赖包：libkysdk-devicesec-devel
- 运行依赖包：libkysdk-devicesec
- 编译参数：无

原型	int kdk_usbguard_removeRule(uint32_t id)	
描述	删除usb设备管控策略	
参数	id	待删除设备管控策略条目序号
返回值	0:	成功
	-1	失败
备注	调用该接口需要kysec开启，并通过授权工具授予devicesec权限	

4.1.2.3 添加指定usb设备的管控策略

- 头文件：libkydevicesec.hpp
- 开发依赖包：libkysdk-devicesec-devel
- 运行依赖包：libkysdk-devicesec
- 编译参数：无

原型	uint32_t kdk_usbguard_appendRule_specific(std::string& vid, std::string& pid, std::string name, std::string serial, std::string interface, int target)	
描述	添加指定usb设备管控策略	
参数	vid	指定设备vid
	pid	指定设备pid
	name	指定设备名
	serial	指定设备序列号
	interface	指定设备接口
返回值	>0	新添加管控条目id
	其他	失败
备注	target 策略类别: Allow=0, Block=1, Reject=2 调用该接口需要kysec开启，并通过授权工具授予devicesec权限	

4.1.2.4 删除指定usb设备的管控策略

- 头文件：libkydevicesec.hpp
- 开发依赖包：libkysdk-devicesec-devel

- 运行依赖包：libkysdk-devicesec
- 编译参数：无

原型	uint32_t kdk_usbguard_removeRule_specific(std::string& vid, std::string& pid, std::string name, std::string serial)	
描述	删除指定usb设备管控策略	
参数	vid	指定设备vid
	pid	指定设备pid
	name	指定设备名
	serial	指定设备序列号
	interface	指定设备接口
返回值	0	成功
	-1	失败
备注	调用该接口需要kysec开启，并通过授权工具授予devicesec权限	

4.1.2.5 获取usb设备的管控策略

- 头文件：libkydevicesec.hpp
- 开发依赖包：libkysdk-devicesec-devel
- 运行依赖包：libkysdk-devicesec
- 编译参数：无

原型	int kdk_usbguard_getRules(std::vector<Rule>& rules)	
描述	获取usb设备管控策略列表	
参数	rules	当前策略列表
返回值	0:	成功
	-1	失败
备注	调用该接口需要kysec开启，并通过授权工具授予devicesec权限	

4.1.2.6 获取指定usb设备管控策略

- 头文件：libkydevicesec.hpp
- 开发依赖包：libkysdk-devicesec-devel
- 运行依赖包：libkysdk-devicesec
- 编译参数：无

原型	uint32_t kdk_usbguard_getRule_specific(std::string& vid, std::string& pid, std::string name, std::string serial, std::string interface)

描述	获取指定usb设备管控策略	
参数	vid	指定设备vid
	pid	指定设备pid
	name	指定设备名
	serial	指定设备序列号
	interface	指定设备接口
返回值	0	Allow
	1	Block
	2	Reject
	< 0	失败
备注	调用该接口需要kysec开启，并通过授权工具授予devicesec权限	

4.1.2.7 获取当前已连接外设usb设备管控状态

- 头文件：libkydevicesec.hpp
- 开发依赖包：libkysdk-devicesec-devel
- 运行依赖包：libkysdk-devicesec
- 编译参数：无

原型	int kdk_usbguard_getDevices(std::vector<Rule>& devices);	
描述	获取当前已连接外设usb设备管控状态	
参数	devices	当前设备列表
返回值	0:	成功
	-1	失败
备注	调用该接口需要kysec开启，并通过授权工具授予devicesec权限	

4.2 应用防护接口

4.2.1 软件包防卸载

添加一个包到防卸载名单中

- 头文件：appuninstsec/libkyappuninstsec.h
- 开发依赖包：libkysdk-appuninstsec-devel
- 运行依赖包：libkysdk-appuninstsec
- 编译参数： -lkyappuninstsec -ldl -lkysdk_module_authorize

原型	int kdk_pkg_uninstall_add(const char *pkgname)

描述	添加一个包到防卸载名单中	
参数	pkgname [in]	软件包名称
返回值	0	成功
	1	软件包已经在列表
	-1	失败
备注	调用该接口需要kysec开启，并通过授权工具授予appuninstsec权限	

从防卸载名单中删除一个包

- 头文件：appuninstsec/libkyappuninstsec.h
- 开发依赖包：libkysdk-appuninstsec-devel
- 运行依赖包：libkysdk-appuninstsec
- 编译参数： -lkyappuninstsec -ldl -lkysdk_module_authorize

原型	int kdk_pkg_uninstall_del(const char *pkgname)	
描述	从防卸载名单中删除一个包	
参数	pkgname [in]	软件包名称
返回值	0	成功
	1	不存在
	-1	失败
备注	调用该接口需要kysec开启，并通过授权工具授予appuninstsec权限	

4.2.2 文件保护接口

设置文件只读属性

- 头文件：libkyfilesec.h
- 开发依赖包：libkysdk-filesec-devel
- 运行依赖包： libkysdk-filesec
- 编译参数：-lkyfilesec -ldl -lkysdk_module_authorize

原型	int kdk_set_file_readonly(const char* filepath);
描述	设置文件只读属性

参数	filepath [in]	文件路径
返回值	0	设置成功
	非0	设置失败
备注	调用该接口需要kysec开启，并通过授权工具授予filesec权限	

取消文件只读属性

- 头文件：libkyfilesec.h
- 开发依赖包：libkysdk-filesec-devel
- 运行依赖包：libkysdk-filesec
- 编译参数：-lkyfilesec -ldl -lkysdk_module_authorize

原型	int kdk_cancel_file_readonly(const char* filepath);	
描述	取消文件只读属性	
参数	filepath [in]	文件路径
返回值	0	取消成功
	非0	取消失败
备注	调用该接口需要kysec开启，并通过授权工具授予filesec权限	

4.2.3 进程防杀

设置进程不可被杀死

- 头文件：libkyprocesssec.h
- 开发依赖包：libkysdk-processsec-devel
- 运行依赖包： libkysdk-processsec
- 编译参数：-lkyprocesssec -ldl -lkysdk_module_authorize

原型	int kdk_process_set_anti_killed(const char *filepath);	
描述	设置进程不可被杀死	
参数	filepath [in]	文件路径
返回值	0	设置成功
	非0	设置失败
备注	调用该接口需要kysec开启，并通过授权工具授予processsec权限	

取消设置进程不可被杀死

- 头文件：libkyprocesssec.h
- 开发依赖包：libkysdk-processsec-devel
- 运行依赖包：libkysdk-processsec
- 编译参数：-lkyprocesssec -ldl -lkysdk_module_authorize

原型	int kdk_process_cancel_anti_killed(const char *filepath);	
描述	取消进程不可被杀死	
参数	filepath [in]	文件路径
返回值	0	取消成功
	非0	取消失败
备注	调用该接口需要kysec开启，并通过授权工具授予processsec权限	

4.2.4 内核模块防卸载

设置内核模块不可卸载

- 头文件：kmodsec/libkykmodsec.h
- 开发依赖包：libkysdk-kmodsec-devel
- 运行依赖包：libkysdk-kmodsec
- 编译参数： -lkykmodsec -ldl -lkysdk_module_authorize

原型	int kdk_kmod_set_anti_unloaded(const char *filename)	
描述	设置内核模块不可卸载	
参数	filename [in]	文件名
返回值	0	成功
	-1	失败
备注	调用该接口需要kysec开启，并通过授权工具授予kmodsec权限	

取消设置内核模块不可卸载

- 头文件：kmodsec/libkykmodsec.h
- 开发依赖包：libkysdk-kmodsec-devel
- 运行依赖包：libkysdk-kmodsec
- 编译参数： -lkykmodsec -ldl -lkysdk_module_authorize

原型	int kdk_kmod_cancel_anti_unloaded(const char *filename)	
描述	取消设置内核模块不可卸载	
参数	filename [in]	文件名
返回	0	成功

值	-1	失败
备注	调用该接口需要kysec开启，并通过授权工具授予kmodsec权限	

4.3 身份认证接口

4.3.1 pam 认证接口

初始化 pam 认证

- 头文件: security/pam_appl.h, security/pam_misc.h
- 开发依赖包:pam-devel
- 运行依赖包:pam

原型	int pam_start(const char *service_name, const char *user, const struct pam_conv *pam_conversation, pam_handle_t **pamh)	
描述	初始化 pam 认证	
参数	service_name	为发起 pam 认证的程序名。认证时会从 /etc/pam.d/service_name 文件获取认证程序的 pam 配置;
	user	为发起 pam认证的用户,若设为NULL,认证过程中会询问用户输入;
	pam_conversation	指向 pam 认证过程中使用的对话相关的结构体;
	pamh	为 pam 认证相关结构体。其内部结构对程序开发者透明,开发者可调用pam_set_item, pam_get_item 等接口配置认证过程中的相关信息
返回值	PAM_SUCCESS;	成功
	PAM_ABORT PAM_BUF_ERR PAM_SYSTEM_ERR	失败
备注		

pam 身份认证

- 头文件: security/pam_appl.h, security/pam_misc.h
- 开发依赖包:pam-devel
- 运行依赖包:pam

原型	int pam_authenticate(pam_handle_t *pamh, int flags)	
描述	pam 身份认证	
参数	pamh	pam_start 初始化成功的认证相关结构体

	flags	为认证标记变量。若无特殊认证流程可配置为 0
返回值	PAM_SUCCESS;	成功
	PAM_ABORT PAM_AUTH_ERR PAM_AUTHINFO_UNAVAIL PAM_CRED_INSUFFICIENT PAM_MAXTRIES PAM_USER_UNKNOWN	失败
备注		

pam 账户合法性管理

- 头文件: security/pam_appl.h, security/pam_misc.h
- 开发依赖包:pam-devel
- 运行依赖包:pam

原型	int pam_acct_mgmt(pam_handle_t *pamh, int flags)	
描述	pam 账户合法性管理	
参数	pamh	pam_start 初始化成功的认证相关结构体
	flags	为认证标记变量。若无特殊认证流程可配置为 0
返回值	PAM_SUCCESS;	成功
	PAM_ACCT_EXPIRED PAM_AUTH_ERR PAM_NEW_AUTHTOK_REQD PAM_PERM_DENIED PAM_USER_UNKNOWN	失败
备注		

打开用户会话

- 头文件: security/pam_appl.h, security/pam_misc.h
- 开发依赖包:pam-devel
- 运行依赖包:pam

原型	int pam_open_session(pam_handle_t *pamh, int flags)	
描述	打开用户会话	
参数	pamh	pam_start 初始化成功的认证相关结构体
	flags	为认证标记变量。若无特殊认证流程可配置为 0
返回值	PAM_SUCCESS;	成功
	PAM_ABORT PAM_BUF_ERR PAM_SESSION_ERR	失败

备注	

关闭用户会话

- 头文件: security/pam_appl.h, security/pam_misc.h
- 开发依赖包:pam-devel
- 运行依赖包:pam

原型	int pam_close_session(pam_handle_t *pamh, int flags)	
描述	关闭用户会话	
参数	pamh	pam_start 初始化成功的认证相关结构体
	flags	为认证标记变量。若无特殊认证流程可配置为 0
返回值	PAM_SUCCESS;	成功
	PAM_ABORT PAM_BUF_ERR PAM_SESSION_ERR	失败
备注		

结束 pam 认证

- 头文件: security/pam_appl.h, security/pam_misc.h
- 开发依赖包:pam-devel
- 运行依赖包:pam

原型	int pam_end(pam_handle_t *pamh, int pam_status)	
描述	结束 pam 认证	
参数	pamh	pam_start 初始化成功的认证相关结构体
	pam_status	为程序最后一次调用PAM 库中接口的返回值
返回值	PAM_SUCCESS;	成功
	PAM_SYSTEM_ERR	失败
备注		

pam 信息结构体

```
//pam 模块返回信息结构体
struct pam_message {
int msg_style; //信息类型,如 PAM_PROMPT_ECHO_OFF,
PAM_PROMPT_ECHO_ON, PAM_TEXT_INFO,PAM_ERROR_MSG 等
const char *msg; //pam 返回信息内容
};
//程序回复给 pam 模块的信息
```

```
struct pam_response {
char *resp; //认证程序回复信息内容
int resp_retcode; //若回复信息获取正常,置 0
};
//程序认证对话相关结构体
struct pam_conv {
//程序认证对话函数指针
int (*conv)(int num_msg, const struct pam_message **msg,
struct pam_response **resp, void *appdata_ptr);
//程序认证信息指针,若为 NULL,认证信息通过对话函数与用户交互获取
void *appdata_ptr;
};
```

4.3.2 pam 模块接口

pam 身份认证模块接口

- 头文件: security/pam_modules.h
- 开发依赖包:pam-devel
- 运行依赖包:pam

原型	int pam_sm_authenticate(pam_handle_t *pamh, int flags, int argc, const char **argv)	
描述	pam 身份认证模块接口	
参数	pamh	参数由认证程序与配置文件传入
	flags	
	argc	
	argv	
返回值	PAM_SUCCESS;	成功
	PAM_AUTH_ERR, PAM_ABORT, PAM_CRED_INSUFFICIENT, PAM_AUTHINFO_UNAVAIL, PAM_MAXTRIES, PAM_USER_UNKNOWN	失败
备注	该接口由 pam 模块开发人员自行实现,接口名称与参数固定	

pam 账户管理模块接口

- 头文件: security/pam_modules.h
- 开发依赖包:pam-devel
- 运行依赖包:pam

原型	int pam_sm_acct_mgmt(pam_handle_t *pamh, int flags, int argc, const char **argv)	
描述	pam 账户管理模块接口	
参数	pamh	参数由认证程序与配置文件传入
	flags	
	argc	
	argv	
返回值	PAM_SUCCESS;	成功
	PAM_ACCT_EXPIRED, PAM_AUTH_ERR, PAM_NEW_AUTHTOK_REQD, PAM_PERM_DENIED, PAM_USER_UNKNOWN	失败
备注	该接口由 pam 模块开发人员自行实现,接口名称与参数固定	

pam 口令管理模块接口

- 头文件: security/pam_modules.h
- 开发依赖包:pam-devel
- 运行依赖包:pam

原型	int pam_sm_chauthtok(pam_handle_t *pamh, int flags, int argc, const char **argv)	
描述	pam 口令管理模块接口	
参数	pamh	参数由认证程序与配置文件传入
	flags	
	argc	
	argv	
返回值	PAM_SUCCESS;	成功
	PAM_AUTHTOK_ERR, PAM_AUTHTOK_RECOVERY_ERR, PAM_AUTHTOK_LOCK_BUSY, PAM_AUTHTOK_DISABLE_AGING, PAM_PERM_DENIED, PAM_TRY_AGAIN, PAM_USER_UNKNOWN	失败
备注	该接口由 pam 模块开发人员自行实现,接口名称与参数固定	

pam 会话管理模块接口

- 头文件: security/pam_modules.h
- 开发依赖包:pam-devel

- 运行依赖包:pam

原型	int pam_sm_open_session(pam_handle_t *pamh, int flags, int argc, const char **argv)	
描述	pam 会话管理模块接口	
参数	pamh	参数由认证程序与配置文件传入
	flags	
	argc	
	argv	
返回值	PAM_SUCCESS;	成功
	PAM_SESSION_ERR	失败
备注	该接口由 pam 模块开发人员自行实现,接口名称与参数固定	

pam 会话管理模块接口

- 头文件: security/pam_modules.h
- 开发依赖包:pam-devel
- 运行依赖包:pam

原型	int pam_sm_close_session(pam_handle_t *pamh, int flags, int argc, const char **argv)	
描述	pam 会话管理模块接口	
参数	pamh	参数由认证程序与配置文件传入
	flags	
	argc	
	argv	
返回值	PAM_SUCCESS;	成功
	PAM_SESSION_ERR	失败
备注	该接口由 pam 模块开发人员自行实现,接口名称与参数固定	

4.4 openssl国密接口

4.4.1 openssl国密接口

openssl国密API：提供国密SM2加解密、SM2签名验签、SM3哈希、SM4对此加解密通用操作

4.4.1.1 SM2接口说明

- 头文件：openssl/evp.h openssl/ec.h
- 开发依赖包：openssl-devel-1.1.1f-15.p16.ky10
- 运行依赖包：openssl-libs-1.1.1f-15.p16.ky10

功能：创建EC_GROUP对象

原型	EC_GROUP *EC_GROUP_new_by_curve_name(int nid);	
描述	用NID指定的曲线创建一个EC_GROUP对象	
参数	nid[in]	曲线名称的nid
返回值	EC_GROUP对象	成功
	NULL	失败
备注		

功能：创建EC _KEY对象

原型	EC_KEY *EC_KEY_new(void);	
描述	创建一个新的EC _KEY对象。	
参数	void	
返回值	EC_KEY对象	成功
	NULL	失败
备注		

功能：设置EC_KEY对象的EC_GROUP

原型	int EC_KEY_set_group(EC_KEY *key, const EC_GROUP *group);	
描述	设置EC_KEY对象的EC_GROUP。	
参数	Key[out]	EC_KEY对象
	group[in]	EC_KEY使用的EC_GROUP对象
返回值	1	成功
	非1	失败
备注		

功能：创建EC密钥

原型	int EC_KEY_generate_key(EC_KEY *key);	
描述	创建一个新的ec密钥	
参数	Key[out]	EC密钥
返回值	1	成功
	非1	失败
备注		

功能：初始SM2密钥

原型	int EVP_PKEY_set1_SM2(EVP_PKEY *pkey, struct ec_key_st *key);	
描述	初始SM2密钥	
参数	pkey[out]	EVP_PKEY对象
	Key[in]	SM2密钥
返回值	1	成功
	非1	失败

备注	

功能：设置EVP_PKEY的类型

原型	int EVP_PKEY_set_alias_type(EVP_PKEY *pkey, int type);	
描述	设置EVP_PKEY的类型	
参数	pkey[out]	EVP_PKEY对象
	type[in]	EVP_PKEY对象名字
返回值	1	成功
	非1	失败
备注		

功能：创建EVP_PKEY_CTX结构

原型	EVP_PKEY_CTX *EVP_PKEY_CTX_new(EVP_PKEY *pkey, ENGINE *e);	
描述	创建一个新的EVP_PKEY_CTX结构	
参数	pkey[out]	EVP_PKEY对象
	e[in]	算法引擎
返回值	EVP_PKEY_CT对象	成功
	NULL	失败
备注		

功能：初始化加密

原型	int EVP_PKEY_encrypt_init(EVP_PKEY_CTX *ctx);	
描述	初始化加密操作	
参数	ctx[out]	加密上下文
返回值	大于0	成功
	小于0	失败
备注		

功能：非对称加密运算

原型	int EVP_PKEY_encrypt(EVP_PKEY_CTX *ctx, unsigned char *out, size_t *outlen, const unsigned char *in, size_t inlen);	
描述	非对称加密运算	
参数	ctx[in]	加密上下文
	out[out]	输出的密文
	outlen[out]	输出密文长度
	in[in]	输入的明文
	inlen[in]	输入明文长度
返回值	1	成功

	非1	失败
备注		

功能：初始化解密

原型	int EVP_PKEY_decrypt_init(EVP_PKEY_CTX *ctx);	
描述	初始化解密操作	
参数	ctx[out]	解密上下文
返回值	大于0	成功
	小于0	失败
备注		

功能：非对称解密

原型	int EVP_PKEY_decrypt(EVP_PKEY_CTX *ctx, unsigned char *out, size_t *outlen, const unsigned char *in, size_t inlen);	
描述	非对称解密	
参数	ctx[in]	解密上下文
	out[out]	解密得到的明文
	outlen[out]	输出明文长度
	in[in]	输入的密文
	inlen[in]	输入密文长度
返回值	1	成功
	非1	失败
备注		

功能：签名运算

原型	int EVP_PKEY_sign(EVP_PKEY_CTX *ctx, unsigned char *sig, size_t *siglen, const unsigned char *tbs, size_t tbslen);	
描述	签名运算	
参数	ctx[in]	上下文数据
	sig[out]	生成的签名
	siglen[out]	签名的长度
	tbs[in]	待签名数据
	tbslen[in]	待签名数据长度
返回值	1	成功
	非1	失败
备注		

功能：初始验签上下文

原型	int EVP_PKEY_verify_init(EVP_PKEY_CTX *ctx);	
描述	初始验签上下文	
参数	ctx[out]	验签上下文
返回值	1	成功
	非1	失败
备注		

功能：验签操作

原型	int EVP_PKEY_verify(EVP_PKEY_CTX *ctx, const unsigned char *sig, size_t siglen, const unsigned char *tbs, size_t tbslen)	
描述	验签操作	
参数	ctx[in]	上下文数据
	sig[in]	生成的签名
	siglen[in]	签名的长度
	tbs[in]	待签名数据
	tbslen[in]	待签名数据长度
返回值	1	成功
	非1	失败
备注		

4.4.1.2 SM3接口说明

- 头文件：openssl/evp.h
- 开发依赖包：openssl-devel-1.1.1f-15.p16.ky10
- 运行依赖包：openssl-libs-1.1.1f-15.p16.ky10

功能：初始EVP结构为SM3

原型	const EVP_MD *EVP_sm3(void);	
描述	初始evp为sm3算法结构	
参数	void	
	evp算法结构	成功
	NULL	失败
备注		

功能：初始EVP_MD_CTX上下文数据

原型	EVP_MD_CTX *EVP_MD_CTX_new(void)	
描述	初始EVP_MD_CTX上下文数据	
参数	void	
返回值	EVP_MD_CTX上下文	成功

	结构数据	
	NULL	失败
备注		

功能：哈希计算第一步

原型	int EVP_DigestInit_ex(EVP_MD_CTX *ctx, const EVP_MD *type, ENGINE *impl);	
描述	哈希计算第一步	
参数	ctx[in]	上下文数据
	type[in]	初始好的evp算法结构
	impl[in]	算法引擎
返回值	1	成功
	非1	失败
备注		

功能：哈希计算第二步

原型	int EVP_DigestUpdate(EVP_MD_CTX *ctx, const void *d, size_t cnt);	
描述	传入计算的哈希数据，进行散列运算	
参数	ctx[in]	上下文数据
	d[in]	传入计算数据
	cnt[in]	传入计算数据的长度
返回值	1	成功
	非1	失败
备注		

功能：哈希计算第三步

原型	int EVP_DigestFinal_ex(EVP_MD_CTX *ctx, unsigned char *md, unsigned int *size)	
描述	哈希计算结束，得到哈希值	
参数	ctx[in]	上下文数据
	md[out]	计算输出的哈希值
	size[out]	输出哈希值数据的长度
返回值	1	成功
	非1	失败
备注		

4.4.1.3 SM4接口说明

- 头文件：openssl/evp.h
- 开发依赖包：openssl-devel-1.1.1f-15.p16.ky10
- 运行依赖包：openssl-libs-1.1.1f-15.p16.ky10

功能：初始上下文

原型	EVP_CIPHER_CTX *EVP_CIPHER_CTX_new(void);	
描述	初始上下文	
参数	void	
返回值	EVP_CIPHER_CTX	成功
	NULL	失败
备注		

功能：初始sm4算法ecb模式evp算法结构

原型	const EVP_CIPHER *EVP_sm4_ecb(void);	
描述	初始sm4算法ecb模式evp算法结构	
参数	void	
返回值	sm4_ecb模式 EVP_CIPHER结构	成功
	NULL	失败
备注		

功能：初始sm4算法cbc模式evp算法结构

原型	const EVP_CIPHER *EVP_sm4_cbc(void);	
描述	初始sm4算法cbc模式evp算法结构	
参数	void	
返回值	sm4_cbc模式 EVP_CIPHER结构	成功
	NULL	失败
备注		

功能：初始sm4算法cfb128模式evp算法结构

原型	const EVP_CIPHER *EVP_sm4_cfb128(void);	
描述	初始sm4算法cfb128模式evp算法结构	
参数	void	
返回值	sm4_cfb128模式 EVP_CIPHER结构	成功
	NULL	失败
备注		

功能：初始sm4算法ofb模式evp算法结构

原型	const EVP_CIPHER *EVP_sm4_ofb(void);	
描述	初始sm4算法ofb模式evp算法结构	
参数	void	
返回值	sm4_ofb模式 EVP_CIPHER结构	成功
	NULL	失败
备注		

功能：初始sm4算法ctr模式evp算法结构

原型	const EVP_CIPHER *EVP_sm4_ctr(void);	
描述	初始sm4算法ctr模式evp算法结构	
参数	void	
返回值	sm4_ctr模式 EVP_CIPHER结构	成功
	NULL	失败
备注		

功能：EVP加密算法初始化

原型	int EVP_EncryptInit(EVP_CIPHER_CTX *ctx, const EVP_CIPHER *cipher, const unsigned char *key, const unsigned char *iv);	
描述	EVP加密算法初始化	
参数	ctx[in]	加密操作的上下文数据
	cipher[in]	初始好的evp算法结构
	key[in]	密钥
	iv[in]	加密使用的iv值
返回值	1	成功
	非1	失败
备注		

功能：对明文数据进行加密

原型	int EVP_EncryptUpdate(EVP_CIPHER_CTX *ctx, unsigned char *out, int *outl, const unsigned char *in, int inl);	
描述	对明文数据进行加密	
参数	ctx[in]	加密操作上下文数据
	out[out]	加密后输出的结果
	in[in]	明文数据
	inl[in]	明文数据长度
返回值	1	成功
	非1	失败

备注	

功能：完成加密操作输出最后的密文

原型	int EVP_EncryptFinal(EVP_CIPHER_CTX *ctx, unsigned char *out, int *outl);	
描述	完成加密操作输出最后的密文	
参数	ctx[in]	加密操作的上下文数据
	out[out]	密文数据
	outl[out]	密文数据的长度
返回值	1	成功
	非1	失败
备注		

功能：EVP解密算法初始化

原型	int EVP_DecryptInit(EVP_CIPHER_CTX *ctx, const EVP_CIPHER *cipher, const unsigned char *key, const unsigned char *iv);	
描述	EVP解密算法初始化	
参数	ctx[in]	解密操作的上下文数据
	cipher[in]	初始好的evp算法结构
	Key[in]	密钥
	iv[in]	解密使用的iv值
返回值	1	成功
	非1	失败
备注		

功能：对明文数据进行解密

原型	int EVP_DecryptUpdate(EVP_CIPHER_CTX *ctx, unsigned char *out, int *outl, const unsigned char *in, int inl);	
描述	对明文数据进行解密	
参数	ctx[in]	解密操作上下文数据
	out[out]	解密后输出的结果
	in[in]	明文数据
	inl[in]	明文数据长度
返回值	1	成功
	非1	失败
备注		

功能：完成解密操作输出最后的明文

原型	int EVP_DecryptFinal(EVP_CIPHER_CTX *ctx, unsigned char *outm, int *outl);	
描述	完成解密操作输出最后的明文	
参数	ctx[in]	解密操作的上下文数据
	out[out]	明文数据
	outl[out]	明文数据的长度
返回值	1	成功
	非1	失败
备注		

4.4.2 海光密码技术

海光密码技术HCT提供“密码算法加速引擎”和“密钥管理”两个功能。海光密码技术引擎API: 提供国密SM2加解密、SM2签名验签、SM3哈希、SM4对称加解密通用操作

功能依赖包：hct-1.0.2-1.ky10.x86_64

使用如下命令查看当前环境状态：/opt/hygon/hct/hct/script/hctconfig status

```
[root@localhost ~]# rpm -qa hct
hct-1.0.2-1.ky10.x86_64
[root@localhost ~]# /opt/hygon/hct/hct/script/hctconfig status
------------>>>  H C T  I N F O D U M P  <<<--------------

HCT Version:                    1.0.2-2023-0512-rc
HCT engine Dir:                 /opt/hygon/hct/lib64/engines-1.1/
HCT lib Dir:                    /opt/hygon/hct/lib64/
HCT script Dir:                 /opt/hygon/hct/hct/script
HCT function test tool Dir:     /opt/hygon/hct/bin/
HCT performance tool Dir:       /opt/hygon/hct/bin/
HCT created 1921 mdev devices to support max 128 progress

HCT CCP effective Binding info :
0000:05:00.1 'NTBCCP 1468' drv=hct unused=ccp,vfio-pci
0000:24:00.2 'PSPCCP Command DMA Processor 1456' drv=hct unused=ccp,vfio-pci
0000:25:00.1 'NTBCCP 1468' drv=hct unused=ccp,vfio-pci
0000:41:00.2 'PSPCCP Command DMA Processor 1456' drv=hct unused=ccp,vfio-pci
0000:42:00.1 'NTBCCP 1468' drv=hct unused=ccp,vfio-pci
0000:65:00.2 'PSPCCP Command DMA Processor 1456' drv=hct unused=ccp,vfio-pci
0000:66:00.1 'NTBCCP 1468' drv=hct unused=ccp,vfio-pci
0000:85:00.2 'PSPCCP Command DMA Processor 1456' drv=hct unused=ccp,vfio-pci
0000:86:00.1 'NTBCCP 1468' drv=hct unused=ccp,vfio-pci
0000:a3:00.2 'PSPCCP Command DMA Processor 1456' drv=hct unused=ccp,vfio-pci
0000:a4:00.1 'NTBCCP 1468' drv=hct unused=ccp,vfio-pci
0000:c5:00.2 'PSPCCP Command DMA Processor 1456' drv=hct unused=ccp,vfio-pci
0000:c6:00.1 'NTBCCP 1468' drv=hct unused=ccp,vfio-pci
0000:e1:00.2 'PSPCCP Command DMA Processor 1456' drv=hct unused=ccp,vfio-pci
0000:e2:00.1 'NTBCCP 1468' drv=hct unused=ccp,vfio-pci
0000:04:00.2 'PSPCCP Command DMA Processor 1456' drv=ccp unused=vfio-pci,hct

-------------------------------------------------------------------------------



[root@localhost ~]#
```

海光密码技术密钥管理API：提供符合《GMT 0018-2012 密码设备应用接口规范》的SDF接口。

功能依赖如下：

依赖名称	要求	备注
BIOS	海光BIOS PI2.1.0.4（对应安全固件版本1837及以上）	BIOS PI2.1.0.4不支持内部RSA密钥
Hag	1868或以上	
glibc	2.17或以上	

4.4.2.1 海光密码技术引擎接口使用

海光密码技术引擎是基于海光内置ccp硬件加速单元，封装的一套对接openssl标准EVP接口的引擎库。开发者可通过引擎名”hct”加载HCT引擎，并将该引擎应用到各个算法接口中，替换默认的openssl软件引擎。

以下列出了各个算法中涉及到的接口详情，并在附录A中列出了完整的算法调用示例代码。

4.4.2.1.1 加载引擎

- 头文件：openssl/engine.h
- 开发依赖包：openssl-devel-1.1.1f-15.p16.ky10
- 运行依赖包：openssl-libs-1.1.1f-15.p16.ky10

功能：加载openssl内置引擎

原型	void ENGINE_load_builtin_engines(void);	
描述	加载openssl内置引擎	
参数	void	
返回值	void	
备注		

功能：检索引擎

原型	ENGINE *ENGINE_by_id(const char *id);	
描述	依据唯一的id值从列表中检索引擎	
参数	id[in]	加速引擎的id值，填入”hct”
返回值	0	成功
	非0	失败
备注		

4.4.2.1.2 SM2接口说明

- 头文件：openssl/evp.h openssl/ec.h
- 开发依赖包：openssl-devel-1.1.1f-15.p16.ky10
- 运行依赖包：openssl-libs-1.1.1f-15.p16.ky10

功能：创建EC_GROUP对象

原型	EC_GROUP *EC_GROUP_new_by_curve_name(int nid);	
描述	用NID指定的曲线创建一个EC_GROUP对象	
参数	nid[in]	曲线名称的nid，填入 NID_sm2
返回值	EC_GROUP对象	成功
	NULL	失败
备注		

功能：创建EC _KEY对象

原型	EC_KEY *EC_KEY_new(void);	
描述	创建一个新的EC _KEY对象。	
参数	void	
返回值	EC_KEY对象	成功
	NULL	失败
备注		

功能：设置EC_KEY对象的EC_GROUP

原型	int EC_KEY_set_group(EC_KEY *key, const EC_GROUP *group);	
描述	设置EC_KEY对象的EC_GROUP。	
参数	Key[out]	EC_KEY对象
	group[in]	EC_KEY使用的EC_GROUP对象
返回值	1	成功
	非1	失败
备注		

功能：创建EC密钥

原型	int EC_KEY_generate_key(EC_KEY *key);	
描述	创建一个新的ec密钥	
参数	Key[out]	EC密钥
返回值	1	成功
	非1	失败
备注		

功能：初始SM2密钥

原型	int EVP_PKEY_set1_SM2(EVP_PKEY *pkey, struct ec_key_st *key);	
描述	初始SM2密钥	
参数	pkey[out]	EVP_PKEY对象
	Key[in]	SM2密钥
返回值	1	成功
	非1	失败
备注		

功能：设置EVP_PKEY的类型

原型	int EVP_PKEY_set_alias_type(EVP_PKEY *pkey, int type);	
描述	设置EVP_PKEY的类型	
参数	pkey[out]	EVP_PKEY对象
	type[in]	EVP_PKEY对象名字
返回值	1	成功
	非1	失败
备注		

功能：创建EVP_PKEY_CTX结构

原型	EVP_PKEY_CTX *EVP_PKEY_CTX_new(EVP_PKEY *pkey, ENGINE *e);

描述	创建一个新的EVP_PKEY_CTX结构	
参数	pkey[out]	EVP_PKEY对象
	e[in]	算法引擎
返回值	EVP_PKEY_CT对象	成功
	NULL	失败
备注	引擎参数填入ENGINE_by_id获取的引擎指针	

功能：初始化加密

原型	int EVP_PKEY_encrypt_init(EVP_PKEY_CTX *ctx);	
描述	初始化加密操作	
参数	ctx[out]	加密上下文
返回值	大于0	成功
	小于0	失败
备注		

功能：非对称加密运算

原型	int EVP_PKEY_encrypt(EVP_PKEY_CTX *ctx, unsigned char *out, size_t *outlen, const unsigned char *in, size_t inlen);	
描述	非对称加密运算	
参数	ctx[in]	加密上下文
	out[out]	输出的密文
	outlen[out]	输出密文长度
	in[in]	输入的明文
	inlen[in]	输入明文长度
返回值	1	成功
	非1	失败
备注		

功能：初始化解密

原型	int EVP_PKEY_decrypt_init(EVP_PKEY_CTX *ctx);	
描述	初始化解密操作	
参数	ctx[out]	解密上下文
返回值	大于0	成功
	小于0	失败
备注		

功能：非对称解密

原型	int EVP_PKEY_decrypt(EVP_PKEY_CTX *ctx, unsigned char *out, size_t *outlen, const unsigned char *in,

	size_t inlen);	
描述	非对称解密	
参数	ctx[in]	解密上下文
	out[out]	解密得到的明文
	outlen[out]	输出明文长度
	in[in]	输入的密文
	inlen[in]	输入密文长度
返回值	1	成功
	非1	失败
备注		

功能：签名运算

原型	int EVP_PKEY_sign(EVP_PKEY_CTX *ctx, unsigned char *sig, size_t *siglen, const unsigned char *tbs, size_t tbslen);	
描述	签名运算	
参数	ctx[in]	上下文数据
	sig[out]	生成的签名
	siglen[out]	签名的长度
	tbs[in]	待签名数据
	tbslen[in]	待签名数据长度
返回值	1	成功
	非1	失败
备注		

功能：初始验签上下文

原型	int EVP_PKEY_verify_init(EVP_PKEY_CTX *ctx);	
描述	初始验签上下文	
参数	ctx[out]	验签上下文
返回值	1	成功
	非1	失败
备注		

功能：验签操作

原型	int EVP_PKEY_verify(EVP_PKEY_CTX *ctx, const unsigned char *sig, size_t siglen, const unsigned char *tbs, size_t tbslen)	
描述	验签操作	
参数	ctx[in]	上下文数据
	sig[in]	生成的签名
	siglen[in]	签名的长度

	tbs[in]	待签名数据
	tbslen[in]	待签名数据长度
返回值	1	成功
	非1	失败
备注		

4.4.2.1.3 SM3接口说明

- 头文件：openssl/evp.h
- 开发依赖包：openssl-devel-1.1.1f-15.p16.ky10
- 运行依赖包：openssl-libs-1.1.1f-15.p16.ky10

功能：初始EVP结构为SM3

原型	const EVP_MD *EVP_sm3(void);	
描述	初始evp为sm3算法结构	
参数	void	
	evp算法结构	成功
	NULL	失败
备注		

功能：初始EVP_MD_CTX上下文数据

原型	EVP_MD_CTX *EVP_MD_CTX_new(void)	
描述	初始EVP_MD_CTX上下文数据	
参数	void	
返回值	EVP_MD_CTX 上下文结构数据	成功
	NULL	失败
备注		

功能：哈希计算第一步

原型	int EVP_DigestInit_ex(EVP_MD_CTX *ctx, const EVP_MD *type, ENGINE *impl);	
描述	哈希计算第一步	
参数	ctx[in]	上下文数据
	type[in]	初始好的evp算法结构
	impl[in]	算法引擎
返回值	1	成功
	非1	失败
备注	引擎参数填入ENGINE_by_id获取的引擎指针	

功能：哈希计算第二步

原型	int EVP_DigestUpdate(EVP_MD_CTX *ctx, const void *d, size_t cnt);	
描述	传入计算的哈希数据，进行散列运算	
参数	ctx[in]	上下文数据
	d[in]	传入计算数据
	cnt[in]	传入计算数据的长度
返回值	1	成功
	非1	失败
备注	该接口可重复调用，追加数据	

功能：哈希计算第三步

原型	int EVP_DigestFinal_ex(EVP_MD_CTX *ctx, unsigned char *md, unsigned int *size)	
描述	哈希计算结束，得到哈希值	
参数	ctx[in]	上下文数据
	md[out]	计算输出的哈希值
	size[out]	输出哈希值数据的长度
返回值	1	成功
	非1	失败
备注		

4.4.2.1.4 SM4接口说明

- 头文件：openssl/evp.h
- 开发依赖包：openssl-devel-1.1.1f-15.p16.ky10
- 运行依赖包：openssl-libs-1.1.1f-15.p16.ky10

功能：初始上下文

原型	EVP_CIPHER_CTX *EVP_CIPHER_CTX_new(void);	
描述	初始上下文	
参数	void	
返回值	EVP_CIPHER_CTX	成功
	NULL	失败
备注		

功能：初始sm4算法ecb模式evp算法结构

原型	const EVP_CIPHER *EVP_sm4_ecb(void);	
描述	初始sm4算法ecb模式evp算法结构	
参数	void	

返回值	sm4_ecb 模式 EVP_CIPHER结构	成功
	NULL	失败
备注		

功能：初始sm4算法cbc模式evp算法结构

原型	const EVP_CIPHER *EVP_sm4_cbc(void);	
描述	初始sm4算法cbc模式evp算法结构	
参数	void	
返回值	sm4_cbc 模式 EVP_CIPHER结构	成功
	NULL	失败
备注		

功能：初始sm4算法cfb128模式evp算法结构

原型	const EVP_CIPHER *EVP_sm4_cfb128(void);	
描述	初始sm4算法cfb128模式evp算法结构	
参数	void	
返回值	sm4_cfb128 模式 EVP_CIPHER结构	成功
	NULL	失败
备注		

功能：初始sm4算法ofb模式evp算法结构

原型	const EVP_CIPHER *EVP_sm4_ofb(void);	
描述	初始sm4算法ofb模式evp算法结构	
参数	void	
返回值	sm4_ofb 模式 EVP_CIPHER结构	成功
	NULL	失败
备注		

功能：初始sm4算法ctr模式evp算法结构

原型	const EVP_CIPHER *EVP_sm4_ctr(void);	
描述	初始sm4算法ctr模式evp算法结构	
参数	void	
返回值	sm4_ctr 模式 EVP_CIPHER结构	成功
	NULL	失败
备注		

功能：EVP加密算法初始化

原型	int EVP_EncryptInit_ex(

	EVP_CIPHER_CTX *ctx, const EVP_CIPHER *cipher, ENGINE *impl, const unsigned char *key, const unsigned char *iv);	
描述	EVP加密算法初始化	
参数	ctx[in]	加密操作的上下文指针
	cipher[in]	初始好的evp算法结构
	impl[in]	算法引擎
	key[in]	密钥
	iv[in]	加密使用的iv值
返回值	1	成功
	非1	失败
备注	引擎参数填入ENGINE_by_id获取的引擎指针	

功能：对明文数据进行加密

原型	int EVP_EncryptUpdate(EVP_CIPHER_CTX *ctx, unsigned char *out, int *outl, const unsigned char *in, int inl);	
描述	对明文数据进行加密	
参数	ctx[in]	加密操作上下文指针
	out[out]	加密后输出的结果
	in[in]	明文数据
	inl[in]	明文数据长度
返回值	1	成功
	非1	失败
备注		

功能：完成加密操作输出最后的密文

原型	int EVP_EncryptFinal(EVP_CIPHER_CTX *ctx, unsigned char *out, int *outl);	
描述	完成加密操作输出最后的密文	
参数	ctx[in]	加密操作的上下文指针
	out[out]	密文数据
	outl[out]	密文数据的长度
返回值	1	成功
	非1	失败
备注		

功能：EVP解密算法初始化

原型	int EVP_DecryptInit_ex(EVP_CIPHER_CTX *ctx, const EVP_CIPHER *cipher,

	ENGINE *impl, const unsigned char *key, const unsigned char *iv);	
描述	EVP解密算法初始化	
参数	ctx[in]	解密操作的上下文指针
	cipher[in]	初始好的evp算法结构
	impl[in]	算法引擎
	Key[in]	密钥
	iv[in]	解密使用的iv值
返回值	1	成功
	非1	失败
备注	引擎参数填入ENGINE_by_id获取的引擎指针	

功能：对明文数据进行解密

原型	int EVP_DecryptUpdate(EVP_CIPHER_CTX *ctx, unsigned char *out, int *outl, const unsigned char *in, int inl);	
描述	对明文数据进行解密	
参数	ctx[in]	解密操作上下文指针
	out[out]	解密后输出的结果
	in[in]	明文数据
	inl[in]	明文数据长度
返回值	1	成功
	非1	失败
备注		

功能：完成解密操作输出最后的明文

原型	int EVP_DecryptFinal(EVP_CIPHER_CTX *ctx, unsigned char *outm, int *outl);	
描述	完成解密操作输出最后的明文	
参数	ctx[in]	解密操作的上下文指针
	out[out]	明文数据
	outl[out]	明文数据的长度
返回值	1	成功
	非1	失败
备注		

4.4.2.2 海光密码技术SDF接口使用

4.4.2.2.1 设备管理类接口

- 头文件：hysdf.h
- 开发依赖包：hygon-sdf-2.0.1-1.ky10
- 运行依赖包：hygon-psp-3.1.0-1.ky10、hct-1.0.1-1.ky10.x86_64（@kylin 请确认安装包名字）

功能：打开设备

原型：	int SDF_OpenDevice(void **phDeviceHandle);	
描述：	打开密码设备。	
参数：	phDeviceHandle[out]	返回的设备句柄
返回值：	0	成功
	非0	失败，返回错误代码
注：phDeviceHandle由函数初始化并填写内容。		

功能：关闭设备

原型：	int SDF_CloseDevice(void *hDeviceHandle);	
描述：	关闭密码设备，并释放相关资源。	
参数：	hDeviceHandle[in]	已打开的设备句柄
返回值：	0	成功
	非0	失败，返回错误代码

功能：创建会话

原型：	int SDF_OpenSession(void *hDeviceHandle, void **phSessionHandle);	
描述：	创建与密码设备的会话。	
参数：	hDeviceHandle[in]	已打开的设备句柄
	phSessionHandle[out]	返回与密码设备建立的新会话句柄
返回值：	0	成功
	非0	失败，返回错误代码

功能：关闭会话

原型：	int SDF_CloseSession(void *hSessionHandle);	
描述：	关闭与密码设备已建立的会话，并释放相关资源。	
参数：	hSessionHandle [in]	与密码设备已建立的会话句柄
返回值：	0	成功
	非0	失败，返回错误代码

功能：获取设备信息

原型：	int SDF_GetDeviceInfo (void *hSessionHandle, DEVICEINFO *pstDeviceInfo);	
描述：	获取密码设备能力描述。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	pstDeviceInfo [out]	设备能力描述信息
返回值：	0	成功

	非0	失败，返回错误代码

功能：产生随机数

原型：	int SDF_GenerateRandom (void *hSessionHandle, unsigned int uiLength, unsigned char *pucRandom);	
描述：	获取指定长度的随机数。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	uiLength[in]	获取的随机数长度
	pucRandom[out]	缓冲区指针，用于存放获取的随机数
返回值：	0	成功
	非0	失败，返回错误代码

功能：获取私钥使用权限

原型：	int SDF_GetPrivateKeyAccessRight (void *hSessionHandle, unsigned int uiKeyIndex, unsigned char * pucPassword, unsigned int uiPwdLength);	
描述：	获取密码设备内部存储的指定索引位置私钥的使用权。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	uiKeyIndex[in]	密码设备存储的密钥对的索引值
	pucPassword[in]	使用私钥权限的标识码
	uiPwdLength[in]	私钥访问控制码长度
返回值：	0	成功
	非 0	失败，返回错误代码
注：私钥访问控制码不少于 8 字节。		

功能：释放私钥使用权限

原型：	int SDF_ReleasePrivateKeyAccessRight (void *hSessionHandle, unsigned int uiKeyIndex);	
描述：	释放密码设备存储的指定索引位置私钥的使用授权。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	uiKeyIndex[in]	密码设备存储的密钥对的索引值
返回值：	0	成功
	非 0	失败，返回错误代码

4.4.2.2.2 密钥管理类函数

- 头文件：hysdf.h
- 开发依赖包：hygon-sdf-2.0.1-1.ky10
- 运行依赖包：hygon-psp-3.1.0-1.ky10、hct-1.0.1-1.ky10.x86_64（@kylin 请确认安装包名字）

功能：导出RSA签名公钥

原型：	int SDF_ExportSignPublicKey_RSA(void *hSessionHandle,

	unsigned int uiKeyIndex, RSArefPublicKey *pucPublicKey);	
描述：	导出密码设备内部存储的指定索引位置的 RSA 签名公钥。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	uiKeyIndex[in]	密码设备存储的密钥对的索引值
	pucPublicKey[out]	RSA 公钥结构
返回值：	0	成功
	非 0	失败，返回错误代码

功能：导出RSA加密公钥

原型：	int SDF_ExportEncPublicKey_RSA(void *hSessionHandle, unsigned int uiKeyIndex, RSArefPublicKey *pucPublicKey);	
描述：	导出密码设备内部存储的指定索引位置的 RSA 加密公钥。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	uiKeyIndex[in]	密码设备存储的密钥对的索引值
	pucPublicKey[out]	RSA 公钥结构
返回值：	0	成功
	非 0	失败，返回错误代码

功能：产生RSA密钥对并输出

原型：	int SDF_GenerateKeyPair_RSA(void *hSessionHandle, unsigned int uiKeyBits, RSArefPublicKey *pucPublicKey, RSArefPrivateKey *pucPrivateKey);	
描述：	请求密码设备产生指定模长的 RSA 密钥对。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	uiKeyBits [in]	指定密钥模长
	pucPublicKey[out]	RSA 公钥结构
	pucPrivateKey[out]	RSA 私钥结构
返回值：	0	成功
	非 0	失败，返回错误代码

功能：生成会话密钥并用内部RSA公钥加密输出

原型：	int SDF_GenerateKeyWithIPK_RSA (void *hSessionHandle, unsigned int uiIPKIndex, unsigned int uiKeyBits, unsigned char *pucKey, unsigned int *puiKeyLength, void **phKeyHandle);	
描述：	生成会话密钥并用指定索引的内部RSA加密公钥加密输出，同时返回密钥句柄。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	uiIPKIndex[in]	密码设备存储的密钥对的索引值
	uiKeyBits[in]	指定产生的会话密钥长度
	pucKey[out]	缓冲区指针，用于存放返回的密钥密文
	puiKeyLength[out]	返回的密钥密文长度
	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
注：公钥加密数据时填充方式与 PKCS#1 v1.5 的要求进行。		

功能：生成会话密钥并用外部RSA公钥加密输出

原型：	int SDF_GenerateKeyWithEPK_RSA (void *hSessionHandle, unsigned int uiKeyBits, RSArefPublicKey *pucPublicKey, unsigned char *pucKey, unsigned int *puiKeyLength, void **phKeyHandle);	
描述：	生成会话密钥并用外部 RSA 公钥加密输出。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	uiKeyBits[in]	指定产生的会话密钥长度
	pucPublicKey[in]	输入的外部 RSA 公钥结构
	pucKey[out]	缓冲区指针，用于存放返回的密钥密文
	puiKeyLength[out]	返回的密钥密文长度
	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
注：公钥加密数据时填充方式与 PKCS#1 v1.5 的要求进行。		

功能：导入会话密钥并用内部RSA私钥解密

原型：	int SDF_ImportKeyWithISK_RSA (void *hSessionHandle, unsigned int uiISKIndex, unsigned char *pucKey, unsigned int puiKeyLength, void **phKeyHandle);	
描述：	导入会话密钥并用内部RSA私钥解密，同时返回密钥句柄。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	uiISKIndex[in]	密码设备存储的密钥对的索引值
	pucKey[in]	缓冲区指针，用于存放输入的密钥密文
	puiKeyLength[in]	输入的密钥密文长度
	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
注：填充方式与公钥加密时相同。		

功能：基于RSA算法的数字信封转换

原型：	int SDF_ExchangeDigitEnvelopeBaseOnRSA(void *hSessionHandle, unsigned int uiKeyIndex, RSArefPublicKey *pucPublicKey, unsigned char *pucDEInput, unsigned int uiDELength, unsigned char *pucDEOutput, unsigned int *puiDELength);	
描述：	导出密码设备内部存储的指定索引位置的 ECC 签名公钥。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	uiKeyIndex[in]	密码设备存储的内部 RSA 密钥对的索引值
	pucPublicKey[in]	外部 RSA 公钥结构

	pucDEInput[in]	缓冲区指针，用于存放输入的会话密钥密文
	uiDELength[in]	输入的会话密钥密文长度
	pucDEOutput[out]	缓冲区指针，用于存放输出会话密钥的密文
	puiDELength[out]	输出的会话密钥密文的长度
返回值：	0	成功
	非 0	失败，返回错误代码

功能：导出ECC签名公钥

原型：	int SDF_ExportSignPublicKey_ECC(void *hSessionHandle, unsigned int uiKeyIndex, ECCrefPublicKey *pucPublicKey);	
描述：	导出密码设备内部存储的指定索引位置的 ECC 签名公钥。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	uiKeyIndex[in]	密码设备存储的密钥对的索引值
	pucPublicKey[out]	ECC 公钥结构
返回值：	0	成功
	非 0	失败，返回错误代码

功能：导出ECC加密公钥

原型：	int SDF_ExportEncPublicKey_ECC(void *hSessionHandle, unsigned int uiKeyIndex, ECCrefPublicKey *pucPublicKey);	
描述：	导出密码设备内部存储的指定索引位置的 ECC 加密公钥。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	uiKeyIndex[in]	密码设备存储的密钥对的索引值
	pucPublicKey[out]	ECC 公钥结构
返回值：	0	成功
	非 0	失败，返回错误代码

功能：产生ECC密钥对并输出

原型：	int SDF_GenerateKeyPair_ECC(void *hSessionHandle, unsigned int uiAlgID, unsigned int uiKeyBits, ECCrefPublicKey *pucPublicKey, ECCrefPrivateKey *pucPrivateKey);	
描述：	请求密码设备产生指定模长的 ECC 密钥对。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	uiAlgID[in]	指定算法标识
	uiKeyBits [in]	指定密钥模长
	pucPublicKey[out]	ECC 公钥结构
	pucPrivateKey[out]	ECC 私钥结构
返回值：	0	成功
	非 0	失败，返回错误代码

功能：生成会话密钥并用内部ECC公钥加密输出

原型：	int SDF_GenerateKeyWithIPK_ECC (void *hSessionHandle, unsigned int uiIPKIndex, unsigned int uiKeyBits,

	ECCCipher *pucKey, void **phKeyHandle);	
描述：	生成会话密钥并用指定索引的内部ECC加密公钥加密输出，同时返回密钥句柄。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	uiIPKIndex[in]	密码设备存储的密钥对的索引值
	uiKeyBits[in]	指定产生的会话密钥长度
	pucKey[out]	缓冲区指针，用于存放返回的密钥密文
	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

功能：生成会话密钥并用外部ECC公钥加密输出

原型：	int SDF_GenerateKeyWithEPK_ECC (void *hSessionHandle, unsigned int uiKeyBits, unsigned int uiAlgID, ECCrefPublicKey *pucPublicKey, ECCCipher *pucKey, void **phKeyHandle);	
描述：	生成会话密钥并用外部 ECC 公钥加密输出，同时返回密钥句柄。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	uiKeyBits[in]	指定产生的会话密钥长度
	uiAlgID[in]	外部 ECC 公钥的算法标识
	pucPublicKey[in]	输入的外部 ECC 公钥结构
	pucKey[out]	缓冲区指针，用于存放返回的密钥密文
	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

功能：导入会话密钥并用内部ECC私钥解密

原型：	int SDF_ImportKeyWithISK_ECC (void *hSessionHandle, unsigned int uiISKIndex, ECCCipher *pucKey, void **phKeyHandle);	
描述：	导入会话密钥并用内部ECC加密私钥解密，同时返回密钥句柄。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	uiISKIndex[in]	密码设备存储的密钥对的索引值
	pucKey[in]	缓冲区指针，用于存放输入的密钥密文
	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

功能：生成密钥协商参数并输出

原型：	int SDF_GenerateAgreementDataWithECC (void *hSessionHandle, unsigned int uiISKIndex, unsigned int uiKeyBits, unsigned char *pucSponsorID, unsigned int uiSponsorIDLength, ECCrefPublicKey *pucSponsorPublicKey, ECCrefPublicKey *pucSponsorTmpPublicKey, void **phAgreementHandle);

描述：	使用ECC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ECC公钥、临时ECC密钥对的公钥及协商句柄。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	uiISKIndex[in]	密码设备存储的密钥对的索引值
	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 ECC 公钥结构
	pucSponsorTmpPublicKey[out]	返回的发起方临时 ECC 公钥结构
	phAgreementHandle[out]	返回的协商句柄，用于计算协商密钥
返回值：	0	成功
	非 0	失败，返回错误代码
注：协商会话密钥时，协商的发起方应首先调用本函数。		

功能：计算会话密钥

原型：	int SDF_GenerateKeyWithECC (void *hSessionHandle, unsigned char *pucResponseID, unsigned int uiResponseIDLength, ECCrefPublicKey *pucResponsePublicKey, ECCrefPublicKey *pucResponseTmpPublicKey, void *hAgreementHandle, void **phKeyHandle);	
描述：	使用ECC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：	hSessionHandle[in]	与设备建立的会话句柄
	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 ECC 公钥结构
	pucResponseTmpPublicKey[in]	外部输入的响应方临时 ECC 公钥结构
	hAgreementHandle[in]	协商句柄，用于计算协商密钥
	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
注：协商的发起方在获得响应方的协商参数后调用本函数。计算会话密钥。使用 SM2 算法计算会话密钥的过程参见 GM/T 0009。		

功能：产生协商数据并计算会话密钥

原型：	int SDF_GenerateAgreementDataAndKeyWithECC (void *hSessionHandle, unsigned int uiISKIndex, unsigned int uiKeyBits, unsigned char *pucResponseID, unsigned int uiResponseIDLength, unsigned char *pucSponsorID, unsigned int uiSponsorIDLength, ECCrefPublicKey *pucSponsorPublicKey, ECCrefPublicKey *pucSponsorTmpPublicKey, ECCrefPublicKey *pucResponsePublicKey, ECCrefPublicKey *pucResponseTmpPublicKey,

	void **phKeyHandle);	
描述：	使用ECC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：	hSessionHandle[in]	与设备建立的会话句柄
	uiISKIndex[in]	密码设备存储的密钥对的索引值
	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 ECC 公钥结构
	pucSponsorTmpPublicKey[in]	外部输入的发起方临时 ECC 公钥结构
	pucResponsePublicKey[out]	返回的响应方 ECC 公钥结构
	pucResponseTmpPublicKey[out]	返回的响应方临时 ECC 公钥结构
	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
注：本函数由响应方调用。使用 SM2 算法计算会话密钥的过程参见 GM/T 0009。		

功能：基于ECC算法的数字信封转换

原型：	int SDF_ExchangeDigitEnvelopeBaseOnECC(void *hSessionHandle, unsigned int uiKeyIndex, unsigned int uiAlgID, ECCrefPublicKey *pucPublicKey, ECCCipher *pucEncDataIn, ECCCipher *pucEncDataOut);	
描述：	导出密码设备内部存储的指定索引位置的 ECC 签名公钥。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	uiKeyIndex[in]	密码设备存储的内部 ECC 密钥对的索引值
	uiAlgID [in]	外部 ECC 公钥的算法标识
	pucPublicKey[in]	外部 ECC 公钥结构
	pucEncDataIn[in]	缓冲区指针，用于存放输入的会话密钥密文
	pucEncDataOut[out]	缓冲区指针，用于存放输出会话密钥的密文
返回值：	0	成功
	非 0	失败，返回错误代码

功能：生成会话密钥并用密钥加密密钥加密输出

原型：	int SDF_GenerateKeyWithKEK (void *hSessionHandle, unsigned int uiKeyBits, unsigned int uiAlgID, unsigned int uiKEKIndex, unsigned char *pucKey, unsigned int *puiKeyLength, void **phKeyHandle);	
描述：	生成会话密钥并用密钥加密密钥加密输出，同时返回密钥句柄。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	uiKeyBits[in]	指定产生的会话密钥长度

	uiAlgID[in]	算法标识，指定对称加密算法
	uiKEKIndex[in]	密码设备存储的密钥加密密钥的索引值
	pucKey[out]	缓冲区指针，用于存放返回的密钥密文
	puiKeyLength[out]	返回的密钥密文长度
	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
注：加密模式使用 ECB 模式。		

功能：导入会话密钥并用密钥加密密钥解密

原型：	int SDF_ImportKeyWithKEK (void *hSessionHandle, unsigned int uiAlgID, unsigned int uiKEKIndex, unsigned char *pucKey, unsigned int uiKeyLength, void **phKeyHandle);	
描述：	导入会话密钥并用密钥加密密钥解密，同时返回会话密钥句柄。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	uiAlgID[in]	算法标识，指定对称加密算法
	uiKEKIndex[in]	密码设备存储的密钥加密密钥的索引值
	pucKey[in]	缓冲区指针，用于存放输入的密钥密文
	uiKeyLength[in]	输入的密钥密文长度
	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
注：加密模式使用 ECB 模式。		

功能：销毁会话密钥

原型：	int SDF_DestroyKey (void *hSessionHandle, void *hKeyHandle);	
描述：	销毁会话密钥，并释放为密钥句柄分配的内存等资源。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	hKeyHandle[in]	输入的密钥句柄
返回值：	0	成功
	非 0	失败，返回错误代码
注：在对称算法运算完成后，应调用本函数销毁会话密钥。		

功能：导入会话密钥

原型：	int SDF_ImportKey(void *hSessionHandle, unsigned char *pucKey, unsigned int uiKeyLength, void **phKeyHandle);	
描述：	导入一个会话密钥明文，支持16字节、24字节、32字节三种长度的对称密钥明文。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	pucKey[in]	输入密钥明文
	uiKeyLength[in]	输入密钥明文长度
	phKeyHandle[out]	输出获取的会话密钥句柄

返回值：	0	成功
	非 0	失败，返回错误代码
注：此函数为非 0018 标准的扩展函数。		

4.4.2.2.3 非对称算法运算

- 头文件：hysdf.h
- 开发依赖包：hygon-sdf-2.0.1-1.ky10
- 运行依赖包：hygon-psp-3.1.0-1.ky10、hct-1.0.1-1.ky10.x86_64（@kylin 请确认安装包名字）

功能：外部公钥RSA运算

原型：	int SDF_ExternalPublicKeyOperation_RSA(void *hSessionHandle, RSArefPublicKey *pucPublicKey, unsigned char *pucDataInput, unsigned int uiInputLength, unsigned char *pucDataOutput, unsigned int *puiOutputLength);	
描述：	指定使用外部公钥对数据进行 RSA 运算。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	pucPublicKey [in]	外部 RSA 公钥结构
	pucDataInput [in]	缓冲区指针，用于存放输入的数据
	uiInputLength[in]	输入的数据长度
	pucDataOutput[out]	缓冲区指针，用于存放输出的数据
	puiOutputLength[out]	输出的数据长度
返回值：	0	成功
	非 0	失败，返回错误代码
注：数据格式由应用层封装。		

功能：内部公钥RSA运算

原型：	int SDF_InternalPublicKeyOperation_RSA(void *hSessionHandle, unsigned int uiKeyIndex, unsigned char *pucDataInput, unsigned int uiInputLength, unsigned char *pucDataOutput, unsigned int *puiOutputLength);	
描述：	使用内部指定索引的公钥对数据进行 RSA 运算。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	uiKeyIndex[in]	密码设备存储的密钥对的索引值
	pucDataInput[in]	缓冲区指针，用于存放外部输入的数据
	uiInputLength[in]	输入的数据长度
	pucDataOutput[out]	缓冲区指针，用于存放输出的数据
	puiOutputLength[out]	输出的数据长度
返回值：	0	成功
	非 0	失败，返回错误代码
注：索引范围仅限于内部签名密钥对，数据格式由应用层封装。		

功能：内部私钥RSA运算

原型：	int SDF_InternalPrivateKeyOperation_RSA(void *hSessionHandle, unsigned int uiKeyIndex, unsigned char *pucDataInput, unsigned int uiInputLength, unsigned char *pucDataOutput, unsigned int *puiOutputLength);	
描述：	使用内部指定索引的私钥对数据进行 RSA 运算。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	uiKeyIndex[in]	密码设备存储的密钥对的索引值
	pucDataInput[in]	缓冲区指针，用于存放外部输入的数据
	uiInputLength[in]	输入的数据长度
	pucDataOutput[out]	缓冲区指针，用于存放输出的数据
	puiOutputLength[out]	输出的数据长度
返回值：	0	成功
	非 0	失败，返回错误代码
注：索引范围仅限于内部签名密钥对，数据格式由应用层封装。		

功能：外部密钥ECC验证

原型：	int SDF_ExternalVerify_ECC(void *hSessionHandle, unsigned int uiAlgID, ECCrefPublicKey *pucPublicKey, unsigned char *pucDataInput, unsigned int uiInputLength, ECCSignature *pucSignature);	
描述：	使用外部公钥对 ECC 签名值进行验证运算。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	uiAlgID[in]	算法标识，指定使用的 ECC 算法
	pucPublicKey[in]	外部 ECC 公钥结构
	pucData[in]	缓冲区指针，用于存放外部输入的数据
	uiDataLength[in]	输入的数据长度
	pucSignature[in]	缓冲区指针，用于存放输入的签名值数据
返回值：	0	成功
	非 0	失败，返回错误代码
注：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，预处理过程见GM/T 0009。		

功能：内部密钥ECC签名

原型：	int SDF_InternalSign_ECC(void *hSessionHandle, unsigned int uiISKIndex, unsigned char *pucData, unsigned int uiDataLength, ECCSignature *pucSignature);	
描述：	使用内部私钥对数据进行 ECC 签名运算。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	uiISKIndex [in]	密码设备存储的密钥对的索引值
	pucData[in]	缓冲区指针，用于存放外部输入的数据
	uiDataLength[in]	输入的数据长度
	pucSignature [out]	缓冲区指针，用于存放输出的签名值数据

返回值：	0	成功
	非 0	失败，返回错误代码
注：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，预处理过程见 GM/T 0009。		

功能：内部密钥ECC验证

原型：	int SDF_InternalVerify_ECC(void *hSessionHandle, unsigned int uiISKIndex, unsigned char *pucData, unsigned int uiDataLength, ECCSignature *pucSignature);	
描述：	使用内部公钥对 ECC 签名值进行验证运算。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	uiISKIndex [in]	密码设备存储的密钥对的索引值
	pucData[in]	缓冲区指针，用于存放外部输入的数据
	uiDataLength[in]	输入的数据长度
	pucSignature[in]	缓冲区指针，用于存放输入的签名值数据
返回值：	0	成功
	非 0	失败，返回错误代码
注：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，预处理过程见GM/T 0009。		

功能：外部密钥ECC加密

原型：	int SDF_ExternalEncrypt_ECC(void *hSessionHandle, unsigned int uiAlgID, ECCrefPublicKey *pucPublicKey, unsigned char *pucData, unsigned int uiDataLength, ECCCipher *pucEncData);	
描述：	使用外部公钥对数据进行 ECC 加密运算。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	uiAlgID[in]	算法标识，指定使用的 ECC 算法
	pucPublicKey[in]	外部 ECC 公钥结构
	pucData[in]	缓冲区指针，用于存放外部输入的数据
	uiDataLength[in]	输入的数据长度
	pucEncData[out]	缓冲区指针，用于存放输出的数据密文
返回值：	0	成功
	非 0	失败，返回错误代码

功能：内部ECC私钥解密

原型：	int SDF_InternalDecrypt_ECC (void *hSessionHandle, unsigned int uiISKIndex, ECCCipher *pucEncData, unsigned char *pucData, unsigned int *uiDataLength);	
描述：	使用内部私钥对数据进行 ECC 解密运算。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	uiISKIndex[in]	密码设备存储的密钥对的索引值
	pucEncData[in]	缓冲区指针，用于存放外部输入的数据密文

	pucData[out]	缓冲区指针，用于存放输出的数据
	uiDataLength[out]	输出的数据长度
返回值：	0	成功
	非 0	失败，返回错误代码
注：此函数为非 0018 标准的扩展函数。		

功能：外部ECC私钥签名

原型：	int SDF_ExternalSign_ECC(void *hSessionHandle, unsigned int uiAlgID, ECCrefPrivateKey *pucPrivateKey, unsigned char *pucData, unsigned int uiDataLength, ECCSignature *pucSignature);	
描述：	使用外部私钥对数据进行 ECC 签名运算。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	uiAlgID[in]	算法标识，指定使用的 ECC 算法
	pucPrivateKey[in]	外部 ECC 私钥结构
	pucData[in]	缓冲区指针，用于存放输入的数据
	uiDataLength[in]	输入的数据长度
	pucSignature[out]	缓冲区指针，用于存放输出的签名值
返回值：	0	成功
	非 0	失败，返回错误代码
注：此函数为非 0018 标准的扩展函数，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，预处理过程见 GM/T 0009。		

功能：外部ECC私钥解密

原型：	int SDF_ExternalDecrypt_ECC(void *hSessionHandle, unsigned int uiAlgID, ECCrefPrivateKey *pucPrivateKey, ECCCipher *pucEncData, unsigned char *pucData, unsigned int *uiDataLength);	
描述：	使用外部私钥对数据进行 ECC 解密运算。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	uiAlgID[in]	算法标识，指定使用的 ECC 算法
	pucPrivateKey[in]	外部 ECC 私钥结构
	pucEncData[in]	缓冲区指针，用于存放输入的数据密文
	pucData[out]	缓冲区指针，用于存放输出的数据
	uiDataLength[out]	输出的数据长度
返回值：	0	成功
	非 0	失败，返回错误代码
注：此函数为非 0018 标准的扩展函数。		

功能：内部ECC公钥加密

原型：	int SDF_InternalEncrypt_ECC(void *hSessionHandle, unsigned int uiIPKIndex, unsigned char *pucData, unsigned int uiDataLength, ECCCipher *pucEncData);

描述：	使用内部公钥对数据进行 ECC 加密运算。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	uiIPKIndex [in]	密码设备存储的密钥对的索引值
	pucData [in]	缓冲区指针，用于存放输入的数据
	uiDataLength [in]	输入的数据长度
	pucEncData [out]	缓冲区指针，用于存放输出的数据密文
返回值：	0	成功
	非 0	失败，返回错误代码
注：此函数为非 0018 标准的扩展函数。		

功能：外部私钥RSA运算

原型：	int SDF_ExternalPrivateKeyOperation_RSA(void *hSessionHandle, RSArefPrivateKey *pucPrivateKey, unsigned char *pucDataInput, unsigned int uiInputLength, unsigned char *pucDataOutput, unsigned int *puiOutputLength);	
描述：	指定使用外部私钥对数据进行 RSA 运算。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	pucPrivateKey[in]	外部 RSA 私钥结构
	pucDataInput[in]	缓冲区指针，用于存放输入的数据
	uiInputLength[in]	输入的数据长度
	pucDataOutput[out]	缓冲区指针，用于存放输出的数据
	puiOutputLength[out]	输出的数据长度
返回值：	0	成功
	非 0	失败，返回错误代码
注：此函数为非0018标准的扩展函数，数据格式由应用层封装。		

4.4.2.2.4 对称算法运算

- 头文件：hysdf.h
- 开发依赖包：hygon-sdf-2.0.1-1.ky10
- 运行依赖包：hygon-psp-3.1.0-1.ky10、hct-1.0.1-1.ky10.x86_64（@kylin 请确认安装包名字）

功能：对称加密

原型：	int SDF_Encrypt(void *hSessionHandle, void *hKeyHandle, unsigned int uiAlgID, unsigned char *pucIV, unsigned char *pucData, unsigned int uiDataLength, unsigned char *pucEncData, unsigned int *puiEncDataLength);	
描述：	使用指定的密钥句柄和 IV 对数据进行对称加密运算。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	hKeyHandle[in]	指定的密钥句柄

	uiAlgID[in]	算法标识，指定对称加密算法
	pucIV[in|out]	缓冲区指针，用于存放输入和返回的 IV 数据
	pucData[in]	缓冲区指针，用于存放输入的数据明文
	uiDataLength[in]	输入的数据明文长度
	pucEncData[out]	缓冲区指针，用于存放输出的数据密文
	puiEncDataLength[out]	输出的数据密文长度
返回值：	0	成功
	非 0	失败，返回错误代码
注：此函数不对数据进行分包处理，输入的数据必须是指定算法分组长度的整数倍。		

功能：对称解密

原型：	int SDF_Decrypt(void *hSessionHandle, void *hKeyHandle, unsigned int uiAlgID, unsigned char *pucIV, unsigned char *pucEncData, unsigned int uiEncDataLength, unsigned char *pucData, unsigned int *puiDataLength);	
描述：	使用指定的密钥句柄和 IV 对数据进行对称解密运算。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	hKeyHandle[in]	指定的密钥句柄
	uiAlgID[in]	算法标识，指定对称加密算法
	pucIV[in|out]	缓冲区指针，用于存放输入和返回的 IV 数据
	pucEncData[in]	缓冲区指针，用于存放输入的数据密文
	uiEncDataLength[in]	输入的数据密文长度
	pucData[out]	缓冲区指针，用于存放输出的数据明文
	puiDataLength[out]	输出的数据明文长度
返回值：	0	成功
	非 0	失败，返回错误代码
注：此函数不对数据进行分包处理，输入的数据必须是指定算法分组长度的整数倍。		

功能：计算MAC

原型：	int SDF_CalculateMAC(void *hSessionHandle, void *hKeyHandle, unsigned int uiAlgID, unsigned char *pucIV, unsigned char *pucData, unsigned int uiDataLength, unsigned char *pucMAC, unsigned int *puiMACLength);	
描述：	使用指定的密钥句柄和 IV 对数据进行 MAC 运算。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	hKeyHandle[in]	指定的密钥句柄
	uiAlgID[in]	算法标识，指定 MAC 加密算法
	pucIV[in|out]	缓冲区指针，用于存放输入和返回的 IV 数据
	pucData[in]	缓冲区指针，用于存放输入的数据明文
	uiDataLength[in]	输入的数据明文长度
	pucMAC[out]	缓冲区指针，用于存放输出的 MAC 值

	puiMACLength[out]	输出的 MAC 值长度
返回值：	0	成功
	非 0	失败，返回错误代码
注：此函数不对数据进行分包处理，多包 MAC 运算由 IV 控制最后的 MAC 值。		

功能：外部对称密钥加密

原型：	int SDF_Encrypt_Ex(void *hSessionHandle, unsigned char *pucKey, unsigned int uiKeyLength, unsigned int uiAlgID, unsigned char *pucIV, unsigned char *pucData, unsigned int uiDataLength, unsigned char *pucEncData, unsigned int *puiEncDataLength);	
描述：	使用外部对称密钥进行加密运算	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	pucKey[in]	缓冲区指针，用于存放输入的密钥明文
	uiKeyLength[in]	输入的密钥明文长度
	uiAlgID [in]	算法标识，指定对称加密算法
	pucIV [in|out]	缓冲区指针，用于存放输入和返回的 IV 数据
	pucData [in]	缓冲区指针，用于存放输入的数据明文
	uiDataLength [in]	输入的数据明文长度
	pucEncData [out]	缓冲区指针，用于存放输出的数据密文
	puiEncDataLength [out]	输出的数据密文长度
返回值：	0	成功
	非 0	失败，返回错误代码
注：此函数为非 0018 标准的扩展函数，不对数据进行分包处理，输入的数据必须是指定算法分组长度的整数倍。		

功能：外部对称密钥解密

原型：	int SDF_Decrypt_Ex(void *hSessionHandle, unsigned char *pucKey, unsigned int uiKeyLength, unsigned int uiAlgID, unsigned char *pucIV, unsigned char *pucEncData, unsigned int uiEncDataLength, unsigned char *pucData, unsigned int *puiDataLength);	
描述：	使用外部对称密钥进行加密运算	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	pucKey[in]	缓冲区指针，用于存放输入的密钥明文
	uiKeyLength[in]	输入的密钥明文长度
	uiAlgID [in]	算法标识，指定对称加密算法
	pucIV [in|out]	缓冲区指针，用于存放输入和返回的 IV 数据
	pucEncData[in]	缓冲区指针，用于存放输入的数据密文

	uiEncDataLength[in]	输入的数据密文长度
	pucData[out]	缓冲区指针，用于存放输出的数据明文
	puiDataLength[out]	输出的数据明文长度
返回值：	0	成功
	非 0	失败，返回错误代码
注：此函数为非 0018 标准的扩展函数，不对数据进行分包处理，输入的数据必须是指定算法分组长度的整数倍。		

4.4.2.2.5 杂凑运算类

- 头文件：hysdf.h
- 开发依赖包：hygon-sdf-2.0.1-1.ky10
- 运行依赖包：hygon-psp-3.1.0-1.ky10、hct-1.0.1-1.ky10.x86_64（@kylin 请确认安装包名字）

功能：杂凑运算初始化

原型：	int SDF_HashInit(void *hSessionHandle, unsigned int uiAlgID, ECCrefPublicKey *pucPublicKey, unsigned char *pucID, unsigned int uiDataLength);	
描述：	三步式数据杂凑运算第一步。	
参数：	hSessionHandle[in]	与设备建立的会话句柄。
	uiAlgID[in]	指定杂凑算法标识。
	pucPublicKey[in]	签名者公钥。当 uiAlgID 为 SGD_SM3 时有效。
	pucID[in]	签名者的 ID 值，当 uiAlgID 为 SGD_SM3 时有效。
	uiDataLength[in]	签名者 ID 的长度，当 uiAlgID 为 SGD_SM3 时有效。
返回值：	0	成功
	非 0	失败，返回错误代码
注：uiDataLength 非零且 uiAlgID 为 SGD_SM3 时，函数执行 SM2 的预处理 1 操作。计算过程见 GM/T 0009。		

功能：多包杂凑运算

原型：	int SDF_HashUpdate(void *hSessionHandle, unsigned char *pucData, unsigned int uiDataLength);	
描述：	三步式数据杂凑运算第二步，对输入的明文进行杂凑运算。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	pucData[in]	缓冲区指针，用于存放输入的数据明文
	uiDataLength[in]	输入的数据明文长度
返回值：	0	成功
	非 0	失败，返回错误代码

功能：杂凑运算结束

原型：	int SDF_HashFinal(void *hSessionHandle, unsigned char *pucHash, unsigned int *puiHashLength);	
描述：	三步式数据杂凑运算第三步，杂凑运算结束返回杂凑数据并清除中间数据。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	pucHash[out]	缓冲区指针，用于存放输出的杂凑数据
	puiHashLength[out]	返回的杂凑数据长度
返回值：	0	成功
	非 0	失败，返回错误代码

4.4.2.2.6 用户文件操作

- 头文件：hysdf.h
- 开发依赖包：hygon-sdf-2.0.1-1.ky10
- 运行依赖包：hygon-psp-3.1.0-1.ky10、hct-1.0.1-1.ky10.x86_64（@kylin 请确认安装包名字）

功能：创建文件

原型：	int SDF_CreateFile(void *hSessionHandle, unsigned char *pucFileName， unsigned int uiNameLen， unsigned int uiFileSize);	
描述：	在密码设备内部创建用于存储用户数据的文件。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	pucFileName[in]	缓冲区指针，用于存放输入的文件名，最大长度 128 字节
	uiNameLen[in]	文件名长度
	uiFileSize[in]	文件所占存储空间的长度
返回值：	0	成功
	非 0	失败，返回错误代码

功能：读取文件

原型：	int SDF_ReadFile(void *hSessionHandle, unsigned char *pucFileName， unsigned int uiNameLen， unsigned int uiOffset, unsigned int *puiFileLength, unsigned char *pucBuffer);	
描述：	读取在密码设备内部存储用户数据的文件的内容。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	pucFileName[in]	缓冲区指针，用于存放输入的文件名，最大长度 128 字节
	uiNameLen[in]	文件名长度
	uiOffset[in]	指定读取文件时的偏移值
	puiFileLength[in|out]	入参时指定读取文件内容的长度；出参时返回实际读取文件内容的长度

	pucBuffer[out]	缓冲区指针，用于存放读取的文件数据
返回值：	0	成功
	非 0	失败，返回错误代码

功能：写文件

原型：	int SDF_WriteFile(void *hSessionHandle, unsigned char *pucFileName， unsigned int uiNameLen， unsigned int uiOffset, unsigned int uiFileLength, unsigned char *pucBuffer);	
描述：	向密码设备内部存储用户数据的文件中写入内容。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	pucFileName[in]	缓冲区指针，用于存放输入的文件名，最大长度 128 字节
	uiNameLen[in]	文件名长度
	uiOffset[in]	指定写入文件时的偏移值
	uiFileLength[in]	指定写入文件内容的长度
	pucBuffer[in]	缓冲区指针，用于存放输入的写文件数据
返回值：	0	成功
	非 0	失败，返回错误代码

功能：删除文件

原型：	int SDF_DeleteFile(void *hSessionHandle, unsigned char *pucFileName， unsigned int uiNameLen);	
描述：	删除指定文件名的密码设备内部存储用户数据的文件。	
参数：	hSessionHandle[in]	与设备建立的会话句柄
	pucFileName[in]	缓冲区指针，用于存放输入的文件名，最大长度 128 字节
	uiNameLen[in]	文件名长度
返回值：	0	成功
	非 0	失败，返回错误代码

4.4.3 鲲鹏KAE加速引擎使用接口

KAE 加解密是鲲鹏加速引擎的加解密模块，使用鲲鹏硬加速模块实现 RSA/SM3/SM4/DH/MD5/AES 算法，结合无损用户态驱动框架，提供高性能对称加解密、非对称加解密算法能力，兼容 OpenSSL 1.1.1a 及其之后版本，支持同步&异步机制。

4.4.3.1 环境要求

- CPU：鲲鹏920处理器
- iBMC版本：V365及以上
- BIOS版本：V105及以上

➢ 获取License

安装鲲鹏 KAE 加速引擎之前需要先安装相应的 License，License 安装成功之后，操作系统才能识别到加速器设备。

TaiShan K 系列服务器硬件 KAE 加速引擎已默认开启，无需申请 License。具体 License 申请使用操作可参考《华为服务器 iBMC 许可证 使用指导》（https://www.hikunpeng.com/document/detail/zh/kunpengaccel/encryp-decryp/devg-kae/kunpengaccel_16_0008.html）。

通过 lspci 命令进行查看操作系统是否有加速器设备，如下所示。不同的操作系统 lspci 查出的加速器描述信息可能不同，除了通过关键字进行过滤，用户还可以查看是否存在如下加速器的 sbdf 号信息。

lspci | grep HPRE

79:00.0 Network and computing encryption device: Huawei Technologies Co., Ltd. HiSilicon HPRE Engine (rev 21)

b9:00.0 Network and computing encryption device: Huawei Technologies Co., Ltd. HiSilicon HPRE Engine (rev 21)

lspci | grep RDE

78:01.0 RAID bus controller: Huawei Technologies Co., Ltd. HiSilicon RDE Engine (rev 21)

b8:01.0 RAID bus controller: Huawei Technologies Co., Ltd. HiSilicon RDE Engine (rev 21)

lspci | grep ZIP

75:00.0 Processing accelerators: Huawei Technologies Co., Ltd. HiSilicon ZIP Engine (rev 21)

b5:00.0 Processing accelerators: Huawei Technologies Co., Ltd. HiSilicon ZIP Engine (rev 21)

lspci | grep SEC

76:00.0 Network and computing encryption device: Huawei Technologies Co., Ltd. HiSilicon SEC Engine (rev 21)

b6:00.0 Network and computing encryption device: Huawei Technologies Co., Ltd. HiSilicon SEC Engine (rev 21)

1) 安装包

hisi_hpre-1.3.10-5.ky10.aarch64.rpm
hisi_rde-1.3.10-5.ky10.aarch64.rpm
hisi_sec2-1.3.10-5.ky10.aarch64.rpm
hisi_zip-1.3.10-5.ky10.aarch64.rpm
uacce-1.3.10-5.ky10.aarch64.rpm
libwd-1.3.10-4.ky10.aarch64.rpm
libkae-1.3.10-7.p01.ky10.aarch64.rpm

2) 加载驱动

modprobe uacce
modprobe hisi_qm
modprobe hisi_sec2
modprobe hisi_hpre
modprobe hisi_rde
lsmod | grep uacce

```
[root@localhost warpdrive]# modprobe uacce
[root@localhost warpdrive]# modprobe hisi_qm
[root@localhost warpdrive]# modprobe hisi_sec2
[root@localhost warpdrive]# modprobe hisi_hpre
[root@localhost warpdrive]# lsmod | grep uacce
uacce                  262144  1 hisi_qm
```

lsmod | grep hisi_qm

```
[root@localhost KAE-1.3.11]# lsmod | grep hisi_qm
hisi_qm                262144  3 hisi_sec2,hisi_hpre,hisi_rde
uacce                  262144  1 hisi_qm
```

3) 查看虚拟文件系统下对应的加速器设备

终端输入命令:ls –al /sys/class/uacce/

```
[root@localhost ~]# ls -al /sys/class/uacce/
总用量 0
drwxr-xr-x  2 root root 0  3月  3 19:43 .
drwxr-xr-x 75 root root 0  3月  3 19:43 ..
lrwxrwxrwx  1 root root 0  3月  3 19:43 hisi_hpre-2 -> ../../devices/pci0000:78/0000:78:00.0/0000:79:00.0/uacce/hisi_hpre-2
lrwxrwxrwx  1 root root 0  3月  3 19:43 hisi_hpre-3 -> ../../devices/pci0000:b8/0000:b8:00.0/0000:b9:00.0/uacce/hisi_hpre-3
lrwxrwxrwx  1 root root 0  3月  3 19:43 hisi_rde-4 -> ../../devices/pci0000:78/0000:78:01.0/uacce/hisi_rde-4
lrwxrwxrwx  1 root root 0  3月  3 19:43 hisi_rde-5 -> ../../devices/pci0000:b8/0000:b8:01.0/uacce/hisi_rde-5
lrwxrwxrwx  1 root root 0  3月  3 19:43 hisi_sec2-0 -> ../../devices/pci0000:74/0000:74:01.0/0000:76:00.0/uacce/hisi_sec2-0
lrwxrwxrwx  1 root root 0  3月  3 19:43 hisi_sec2-1 -> ../../devices/pci0000:b4/0000:b4:01.0/0000:b6:00.0/uacce/hisi_sec2-1
```

4.4.3.2 检索引擎

- 头文件：openssl/engine.h
- 开发依赖包：openssl-devel-1.1.1f-15.p16.ky10
- 运行依赖包：openssl-libs-1.1.1f-15.p16.ky10

功能：检索引擎

原型	ENGINE *ENGINE_by_id(const char *id);	
描述	依据唯一的id值从列表中检索引擎	
参数	id[in]	加速引擎的id值
返回值	0	成功
	非0	失败
备注		

功能：加载openssl内置引擎

原型	void ENGINE_load_builtin_engines(void);	
描述	加载openssl内置引擎	
参数	void	
返回值	void	
备注		

4.4.3.3 SM3接口说明

- 头文件：openssl/evp.h
- 开发依赖包：openssl-devel-1.1.1f-15.p16.ky10
- 运行依赖包：openssl-libs-1.1.1f-15.p16.ky10

功能：初始EVP结构为SM3

原型	const EVP_MD *EVP_sm3(void);	
描述	初始evp为sm3算法结构	
参数	void	
	evp算法结构	成功
	NULL	失败
备注		

功能：初始EVP_MD_CTX上下文数据

原型	EVP_MD_CTX *EVP_MD_CTX_new(void)	
描述	初始EVP_MD_CTX上下文数据	
参数	void	
返回值	EVP_MD_CTX上下文结构数据	成功
	NULL	失败
备注		

功能：哈希计算第一步

原型	int EVP_DigestInit_ex(EVP_MD_CTX *ctx, const EVP_MD *type, ENGINE *impl);	
描述	哈希计算第一步	
参数	ctx[in]	上下文数据
	type[in]	初始好的evp算法结构
	impl[in]	算法引擎
返回值	1	成功
	非1	失败
备注		

功能：哈希计算第二步

原型	int EVP_DigestUpdate(EVP_MD_CTX *ctx, const void *d, size_t cnt);
描述	传入计算的哈希数据，进行散列运算

参数	ctx[in]	上下文数据
	d[in]	传入计算数据
	cnt[in]	传入计算数据的长度
返回值	1	成功
	非1	失败
备注		

功能：哈希计算第三步

原型	int EVP_DigestFinal_ex(EVP_MD_CTX *ctx, unsigned char *md, unsigned int *size)	
描述	哈希计算结束，得到哈希值	
参数	ctx[in]	上下文数据
	md[out]	计算输出的哈希值
	size[out]	输出哈希值数据的长度
返回值	1	成功
	非1	失败
备注		

4.4.3.4 SM4接口说明

- 头文件：openssl/evp.h
- 开发依赖包：openssl-devel-1.1.1f-15.p16.ky10
- 运行依赖包：openssl-libs-1.1.1f-15.p16.ky10

功能：初始上下文

原型	EVP_CIPHER_CTX *EVP_CIPHER_CTX_new(void);	
描述	初始上下文	
参数	void	
返回值	EVP_CIPHER_CTX	成功
	NULL	失败
备注		

功能：初始sm4算法ecb模式evp算法结构

原型	const EVP_CIPHER *EVP_sm4_ecb(void);	
描述	初始sm4算法ecb模式evp算法结构	
参数	void	
返回值	sm4_ecb模式 EVP_CIPHER结构	成功
	NULL	失败
备注		

功能：初始sm4算法cbc模式evp算法结构

原型	const EVP_CIPHER *EVP_sm4_cbc(void);	
描述	初始sm4算法cbc模式evp算法结构	
参数	void	
返回值	sm4_cbc模式 EVP_CIPHER结构	成功
	NULL	失败
备注		

功能：初始sm4算法cfb128模式evp算法结构

原型	const EVP_CIPHER *EVP_sm4_cfb128(void);	
描述	初始sm4算法cfb128模式evp算法结构	
参数	void	
返回值	sm4_cfb128模式 EVP_CIPHER结构	成功
	NULL	失败
备注		

功能：初始sm4算法ofb模式evp算法结构

原型	const EVP_CIPHER *EVP_sm4_ofb(void);	
描述	初始sm4算法ofb模式evp算法结构	
参数	void	
返回值	sm4_ofb模式 EVP_CIPHER结构	成功
	NULL	失败
备注		

功能：初始sm4算法ctr模式evp算法结构

原型	const EVP_CIPHER *EVP_sm4_ctr(void);	
描述	初始sm4算法ctr模式evp算法结构	
参数	void	
返回值	sm4_ctr模式 EVP_CIPHER结构	成功
	NULL	失败
备注		

功能：EVP加密算法初始化

原型	int EVP_EncryptInit(EVP_CIPHER_CTX *ctx, const EVP_CIPHER *cipher, const unsigned char *key, const unsigned char *iv);

描述	EVP加密算法初始化	
参数	ctx[in]	加密操作的上下文数据
	cipher[in]	初始好的evp算法结构
	key[in]	密钥
	iv[in]	加密使用的iv值
返回值	1	成功
	非1	失败
备注		

功能：对明文数据进行加密

原型	int EVP_EncryptUpdate(EVP_CIPHER_CTX *ctx, unsigned char *out, int *outl, const unsigned char *in, int inl);	
描述	对明文数据进行加密	
参数	ctx[in]	加密操作上下文数据
	out[out]	加密后输出的结果
	in[in]	明文数据
	inl[in]	明文数据长度
返回值	1	成功
	非1	失败
备注		

功能：完成加密操作输出最后的密文

原型	int EVP_EncryptFinal(EVP_CIPHER_CTX *ctx, unsigned char *out, int *outl);	
描述	完成加密操作输出最后的密文	
参数	ctx[in]	加密操作的上下文数据
	out[out]	密文数据
	outl[out]	密文数据的长度
返回值	1	成功
	非1	失败
备注		

功能：EVP解密算法初始化

原型	int EVP_DecryptInit(EVP_CIPHER_CTX *ctx, const EVP_CIPHER *cipher, const unsigned char *key, const unsigned char *iv);

描述	EVP解密算法初始化	
参数	ctx[in]	解密操作的上下文数据
	cipher[in]	初始好的evp算法结构
	Key[in]	密钥
	iv[in]	解密使用的iv值
返回值	1	成功
	非1	失败
备注		

功能：对明文数据进行解密

原型	int EVP_DecryptUpdate(EVP_CIPHER_CTX *ctx, unsigned char *out, int *outl, const unsigned char *in, int inl);	
描述	对明文数据进行解密	
参数	ctx[in]	解密操作上下文数据
	out[out]	解密后输出的结果
	in[in]	明文数据
	inl[in]	明文数据长度
返回值	1	成功
	非1	失败
备注		

功能：完成解密操作输出最后的明文

原型	int EVP_DecryptFinal(EVP_CIPHER_CTX *ctx, unsigned char *outm, int *outl);	
描述	完成解密操作输出最后的明文	
参数	ctx[in]	解密操作的上下文数据
	out[out]	明文数据
	outl[out]	明文数据的长度
返回值	1	成功
	非1	失败
备注		

4.4.4 兆芯加速引擎使用接口

硬件与软件的对应关系

CPU	固件	驱动	开发包	备注
KH-30000	默认固件	无	openssl-1.1.1f-	SM3、SM4

			15.p15及以上 或 zx_gmi-1.0-1.p01	
KH-40000	默认固件	无	openssl-1.1.1f-15.p20及以上 或 zx_gmi-1.0-1.p01	SM2、SM3、SM4

兆芯引擎API: 提供国密SM2加解密、SM2签名验签、SM3哈希、SM4对此加解密通用操作。功能依赖包：openssl-1.1.1f-15.p20.ky10

4.4.3.1 检索引擎

- 头文件：openssl/engine.h
- 开发依赖包：openssl-devel-1.1.1f-15.p16.ky10
- 运行依赖包：openssl-libs-1.1.1f-15.p16.ky10

功能：检索引擎

原型	ENGINE *ENGINE_by_id(const char *id);	
描述	依据唯一的id值从列表中检索引擎	
参数	id[in]	加速引擎的id值
返回值	0	成功
	非0	失败
备注		

功能：加载openssl内置引擎

原型	void ENGINE_load_builtin_engines(void);	
描述	加载openssl内置引擎	
参数	void	
返回值	void	
备注		

4.4.3.2 SM2接口说明

- 头文件：openssl/evp.h openssl/ec.h
- 开发依赖包：openssl-devel-1.1.1f-15.p16.ky10
- 运行依赖包：openssl-libs-1.1.1f-15.p16.ky10

功能：创建EC_GROUP对象

原型	EC_GROUP *EC_GROUP_new_by_curve_name(int nid);	
描述	用NID指定的曲线创建一个EC_GROUP对象	
参数	nid[in]	曲线名称的nid

返回值	EC_GROUP对象	成功
	NULL	失败
备注		

功能：创建EC _KEY对象

原型	EC_KEY *EC_KEY_new(void);	
描述	创建一个新的EC _KEY对象。	
参数	void	
返回值	EC_KEY对象	成功
	NULL	失败
备注		

功能：设置EC_KEY对象的EC_GROUP

原型	int EC_KEY_set_group(EC_KEY *key, const EC_GROUP *group);	
描述	设置EC_KEY对象的EC_GROUP。	
参数	Key[out]	EC_KEY对象
	group[in]	EC_KEY使用的EC_GROUP对象
返回值	1	成功
	非1	失败
备注		

功能：创建EC密钥

原型	int EC_KEY_generate_key(EC_KEY *key);	
描述	创建一个新的ec密钥	
参数	Key[out]	EC密钥
返回值	1	成功
	非1	失败
备注		

功能：初始SM2密钥

原型	int EVP_PKEY_set1_SM2(EVP_PKEY *pkey, struct ec_key_st *key);	
描述	初始SM2密钥	
参数	pkey[out]	EVP_PKEY对象
	Key[in]	SM2密钥
返回值	1	成功
	非1	失败
备注		

功能：设置EVP_PKEY的类型

原型	int EVP_PKEY_set_alias_type(EVP_PKEY *pkey,

	int type);	
描述	设置EVP_PKEY的类型	
参数	pkey[out]	EVP_PKEY对象
	type[in]	EVP_PKEY对象名字
返回值	1	成功
	非1	失败
备注		

功能：创建EVP_PKEY_CTX结构

原型	EVP_PKEY_CTX *EVP_PKEY_CTX_new(EVP_PKEY *pkey, ENGINE *e);	
描述	创建一个新的EVP_PKEY_CTX结构	
参数	pkey[out]	EVP_PKEY对象
	e[in]	算法引擎
返回值	EVP_PKEY_CT对象	成功
	NULL	失败
备注		

功能：初始化加密

原型	int EVP_PKEY_encrypt_init(EVP_PKEY_CTX *ctx);	
描述	初始化加密操作	
参数	ctx[out]	加密上下文
返回值	大于0	成功
	小于0	失败
备注		

功能：非对称加密运算

原型	int EVP_PKEY_encrypt(EVP_PKEY_CTX *ctx, unsigned char *out, size_t *outlen, const unsigned char *in, size_t inlen);	
描述	非对称加密运算	
参数	ctx[in]	加密上下文
	out[out]	输出的密文
	outlen[out]	输出密文长度
	in[in]	输入的明文
	inlen[in]	输入明文长度
返回值	1	成功
	非1	失败
备注		

功能：初始化解密

原型	int EVP_PKEY_decrypt_init(EVP_PKEY_CTX *ctx);	
描述	初始化解密操作	
参数	ctx[out]	解密上下文
返回值	大于0	成功
	小于0	失败
备注		

功能：非对称解密

原型	int EVP_PKEY_decrypt(EVP_PKEY_CTX *ctx, unsigned char *out, size_t *outlen, const unsigned char *in, size_t inlen);	
描述	非对称解密	
参数	ctx[in]	解密上下文
	out[out]	解密得到的明文
	outlen[out]	输出明文长度
	in[in]	输入的密文
	inlen[in]	输入密文长度
返回值	1	成功
	非1	失败
备注		

功能：签名运算

原型	int EVP_PKEY_sign(EVP_PKEY_CTX *ctx, unsigned char *sig, size_t *siglen, const unsigned char *tbs, size_t tbslen);	
描述	签名运算	
参数	ctx[in]	上下文数据
	sig[out]	生成的签名
	siglen[out]	签名的长度
	tbs[in]	待签名数据
	tbslen[in]	待签名数据长度
返回值	1	成功
	非1	失败
备注		

功能：初始验签上下文

原型	int EVP_PKEY_verify_init(EVP_PKEY_CTX *ctx);	
描述	初始验签上下文	
参数	ctx[out]	验签上下文
返回值	1	成功
	非1	失败

备注	

功能：验签操作

原型	int EVP_PKEY_verify(EVP_PKEY_CTX *ctx, const unsigned char *sig, size_t siglen, const unsigned char *tbs, size_t tbslen)	
描述	验签操作	
参数	ctx[in]	上下文数据
	sig[in]	生成的签名
	siglen[in]	签名的长度
	tbs[in]	待签名数据
	tbslen[in]	待签名数据长度
返回值	1	成功
	非1	失败
备注		

4.4.3.3 SM3接口说明

- 头文件：openssl/evp.h
- 开发依赖包：openssl-devel-1.1.1f-15.p16.ky10
- 运行依赖包：openssl-libs-1.1.1f-15.p16.ky10

功能：初始EVP结构为SM3

原型	const EVP_MD *EVP_sm3(void);	
描述	初始evp为sm3算法结构	
参数	void	
	evp算法结构	成功
	NULL	失败
备注		

功能：初始EVP_MD_CTX上下文数据

原型	EVP_MD_CTX *EVP_MD_CTX_new(void)	
描述	初始EVP_MD_CTX上下文数据	
参数	void	
返回值	EVP_MD_CTX上下文结构数据	成功
	NULL	失败
备注		

功能：哈希计算第一步

原型	int EVP_DigestInit_ex(EVP_MD_CTX *ctx, const EVP_MD *type, ENGINE *impl);	
描述	哈希计算第一步	
参数	ctx[in]	上下文数据
	type[in]	初始好的evp算法结构
	impl[in]	算法引擎
返回值	1	成功
	非1	失败
备注		

功能：哈希计算第二步

原型	int EVP_DigestUpdate(EVP_MD_CTX *ctx, const void *d, size_t cnt);	
描述	传入计算的哈希数据，进行散列运算	
参数	ctx[in]	上下文数据
	d[in]	传入计算数据
	cnt[in]	传入计算数据的长度
返回值	1	成功
	非1	失败
备注		

功能：哈希计算第三步

原型	int EVP_DigestFinal_ex(EVP_MD_CTX *ctx, unsigned char *md, unsigned int *size)	
描述	哈希计算结束，得到哈希值	
参数	ctx[in]	上下文数据
	md[out]	计算输出的哈希值
	size[out]	输出哈希值数据的长度
返回值	1	成功
	非1	失败
备注		

4.4.3.4 SM4接口说明

- 头文件：openssl/evp.h
- 开发依赖包：openssl-devel-1.1.1f-15.p16.ky10
- 运行依赖包：openssl-libs-1.1.1f-15.p16.ky10

功能：初始上下文

原型	EVP_CIPHER_CTX *EVP_CIPHER_CTX_new(void);	
描述	初始上下文	
参数	void	
返回值	EVP_CIPHER_CTX	成功
	NULL	失败
备注		

功能：初始sm4算法ecb模式evp算法结构

原型	const EVP_CIPHER *EVP_sm4_ecb(void);	
描述	初始sm4算法ecb模式evp算法结构	
参数	void	
返回值	sm4_ecb模式 EVP_CIPHER结构	成功
	NULL	失败
备注		

功能：初始sm4算法cbc模式evp算法结构

原型	const EVP_CIPHER *EVP_sm4_cbc(void);	
描述	初始sm4算法cbc模式evp算法结构	
参数	void	
返回值	sm4_cbc模式 EVP_CIPHER结构	成功
	NULL	失败
备注		

功能：初始sm4算法cfb128模式evp算法结构

原型	const EVP_CIPHER *EVP_sm4_cfb128(void);	
描述	初始sm4算法cfb128模式evp算法结构	
参数	void	
返回值	sm4_cfb128模式 EVP_CIPHER结构	成功
	NULL	失败
备注		

功能：初始sm4算法ofb模式evp算法结构

原型	const EVP_CIPHER *EVP_sm4_ofb(void);	
描述	初始sm4算法ofb模式evp算法结构	
参数	void	
返回值	sm4_ofb模式 EVP_CIPHER结构	成功
	NULL	失败
备注		

功能：初始sm4算法ctr模式evp算法结构

原型	const EVP_CIPHER *EVP_sm4_ctr(void);	
描述	初始sm4算法ctr模式evp算法结构	
参数	void	
返回值	sm4_ctr模式EVP_CIPHER结构	成功
	NULL	失败
备注		

功能：EVP加密算法初始化

原型	int EVP_EncryptInit(EVP_CIPHER_CTX *ctx, const EVP_CIPHER *cipher, const unsigned char *key, const unsigned char *iv);	
描述	EVP加密算法初始化	
参数	ctx[in]	加密操作的上下文数据
	cipher[in]	初始好的evp算法结构
	key[in]	密钥
	iv[in]	加密使用的iv值
返回值	1	成功
	非1	失败
备注		

功能：对明文数据进行加密

原型	int EVP_EncryptUpdate(EVP_CIPHER_CTX *ctx, unsigned char *out, int *outl, const unsigned char *in, int inl);	
描述	对明文数据进行加密	
参数	ctx[in]	加密操作上下文数据
	out[out]	加密后输出的结果
	in[in]	明文数据
	inl[in]	明文数据长度
返回值	1	成功
	非1	失败
备注		

功能：完成加密操作输出最后的密文

原型	int EVP_EncryptFinal(EVP_CIPHER_CTX *ctx, unsigned char *out, int *outl);
描述	完成加密操作输出最后的密文

参数	ctx[in]	加密操作的上下文数据
	out[out]	密文数据
	outl[out]	密文数据的长度
返回值	1	成功
	非1	失败
备注		

功能：EVP解密算法初始化

原型	int EVP_DecryptInit(EVP_CIPHER_CTX *ctx, const EVP_CIPHER *cipher, const unsigned char *key, const unsigned char *iv);	
描述	EVP解密算法初始化	
参数	ctx[in]	解密操作的上下文数据
	cipher[in]	初始好的evp算法结构
	Key[in]	密钥
	iv[in]	解密使用的iv值
返回值	1	成功
	非1	失败
备注		

功能：对明文数据进行解密

原型	int EVP_DecryptUpdate(EVP_CIPHER_CTX *ctx, unsigned char *out, int *outl, const unsigned char *in, int inl);	
描述	对明文数据进行解密	
参数	ctx[in]	解密操作上下文数据
	out[out]	解密后输出的结果
	in[in]	明文数据
	inl[in]	明文数据长度
返回值	1	成功
	非1	失败
备注		

功能：完成解密操作输出最后的明文

原型	int EVP_DecryptFinal(EVP_CIPHER_CTX *ctx, unsigned char *outm, int *outl);	
描述	完成解密操作输出最后的明文	
参数	ctx[in]	解密操作的上下文数据
	out[out]	明文数据
	outl[out]	明文数据的长度

返回值	1	成功
	非1	失败
备注		

4.4.5 龙芯密码模块使用接口

龙芯遵照GM-T 0018-2012密码设备应用接口规范，提供标准的SDF接口，详情请见如下附件。

龙芯3A5000用户手册—基础SDK使

4.5 安全审计

4.5.1 获取审计状态

- 头文件：libkyauditsec.h
- 开发依赖包：libkysdk-auditsec-devel
- 运行依赖包：audit-status-libs libkysdk-auditsec
- 编译参数：-lkyauditsec -ldl -lkysdk_module_authorize

原型	int kdk_get_audit_status();	
描述	获取审计日志状态	
参数		
返回值	0	关闭
	1	开启
	其他	获取失败
备注		

4.5.2 设置审计状态

- 头文件：libkyauditsec.h
- 开发依赖包：libkysdk-auditsec-deve
- 运行依赖包：audit-status-libs libkysdk-auditsec
- 编译参数：-lkyauditsec -ldl -lkysdk_module_authorize

原型	int kdk_set_audit_status(int status);

描述	设置审计日志状态	
参数	status[in]	状态：0 关闭,1开启
返回值	0	设置成功
	非0	设置失败
备注		

4.6 LSM安全模块

添加自定义 LSM 模块

原型	int security_add_external_ops(struct security_operations *ops)	
描述	添加自定义 LSM 模块	
参数	ops[in]	自定义钩子结构体
返回值	0	添加成功
	-errno	添加失败
备注	参数 ops:LSM 模块的钩子函数集合, security_operations 结构体定义见 linux/security_ops.h	

删除自定义 LSM 模块

原型	void security_del_external_ops(struct security_operations *ops)	
描述	删除自定义 LSM 模块	
参数	ops[in]	自定义钩子结构体
返回值	无	
备注	参数 ops::LSM 模块的钩子函数集合,该参数必须与调用 security_add_external_ops 接口时的 ops 参数值相同,否则删除操作会失败	

4.7 内置国家CA根证书

内置国家CA根证书API: 提供国家电子CA根证书的更新、导入、查询、删除通用操作

4.7.1 更新系统下国家电子CA根证书

- 头文件：gdca/libkygdca.h
- 开发依赖包：rootca-gov-cn-devel-1.1.0-03.ky10
- 运行依赖包：rootca-gov-cn-libs-1.1.0-03.ky10

功能：更新本地国家电子CA根证书

原型	int kdk_gdca_update(const char *cert_name, GDCA_UPDATA_STATUS **updata_status, int *cert_num, char *install_path=(char *)CERT_DEFAULT_INSTALL_PATH);
描述	更新本地国家电子CA根证书

参数	cert_name[in]	进行更新的证书名字，当传入NULL时，进行全部更新
	update_status[in]	证书更新状态
	cert_num[in]	证书更新的数量
	install_path[in]	安装路径，默认/etc/pki/GDCA/
返回值	0	成功
	非0	失败
备注		

证书更新状态：

```
typedef strusct GDCA_UPDATA_STATUS_st {
    char file_name[CERT_NAME_MAX_SIZE]; //证书名字
    int updata_status; //更新状态
} app_type;

typedef enum {
    CERT_NEWEST,        //已是最新的，未作更新
    CERT_UPDATA,        //已进行更新
    CERT_NOTFOUND   //指定更新的证书不属于国电CA证书
} UPDATA_STATUS;
```

4.7.2 导入国家电子CA根证书

- 头文件：gdca/libkygdca.h
- 开发依赖包：rootca-gov-cn-devel-1.1.0-03.ky10
- 运行依赖包：rootca-gov-cn-libs-1.1.0-03.ky10

原型	int kdk_gdca_import(const char *source_path, char *dest_path=(char *)CERT_DEFAULT_INSTALL_PATH);	
描述	导入国家电子CA根证书	
参数	source_path[in]	待导入证书路径
	dest_path[in]	证书导入路径，默认/etc/pki/GDCA/
返回值	0	成功
	非0	失败
备注		

4.7.3 查询国家电子CA根证书安装情况

- 头文件：gdca/libkygdca.h
- 开发依赖包：rootca-gov-cn-devel
- 运行依赖包：rootca-gov-cn-libs

功能：查询国家电子CA根证书安装情况

原型	int kdk_gdca_exist(GDCA_CERT **gdca_info, int *number, char *install_path=(char *)CERT_DEFAULT_INSTALL_PATH);	
描述	查询国家电子CA根证书安装情况	
参数	gdca_info[out]	已安装证书的信息
	number[out]	已安装证书数量
	install_path[in]	安装路径，默认/etc/pki/GDCA/
返回值	0	成功
	非0	失败
备注		

已安装证书信息

```
typedef struct GDCA_CERT_st {
    char file_name[CERT_NAME_MAX_SIZE];  //证书名字
}GDCA_CERT;
```

4.7.4 删除系统下国家电子CA根证书

- 头文件：gdca/libkygdca.h
- 开发依赖包：rootca-gov-cn-devel
- 运行依赖包：rootca-gov-cn-libs

功能：删除系统下国家电子CA根证书

原型	int kdk_gdca_delete(const char *file_name, char *dest_path=(char *)CERT_DEFAULT_INSTALL_PATH);	
描述	删除系统下国家电子CA根证书	
参数	file_name[in]	要删除的证书名字
	dest_path[in]	证书部署路径,默认/etc/pki/GDCA/
返回值	0	成功
	非0	失败
备注		

4.8 TCM1.2可信软件栈

4.8.1 通用接口类

4.8.1.1 Tspi_SetAttribUint32

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_SetAttribUint32(TSS_HOBJECT hObject, /* in */ TSS_FLAG attribFlag, /* in */ TSS_FLAG subFlag, /* in */ UINT32 ulAttrib) /* in */
功能描述	以 32-bit 长度设置软件栈对象的属性
参数说明	hObject 需要设置属性的对象句柄 attribFlag 需要设置的属性标记 subFlag 需要设置的子属性标记 ulAttrib 属性值
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.1.2 Tspi_GetAttribUint32

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容

接口定义	TSS_RESULT Tspi_GetAttribUint32(TSS_HOBJECT hObject, /* in */ TSS_FLAG attribFlag, /* in */ TSS_FLAG subFlag, /* in */ UINT32 * pulAttrib) /* out */
功能描述	以 32-bit 长度获取软件栈对象的属性
参数说明	hObject 需要设置属性的对象句柄 attribFlag 需要设置的属性标记 subFlag 需要设置的子属性标记
	pulAttrib 属性值数据
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.1.3 Tspi_SetAttribData

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_SetAttribData(TSS_HOBJECT hObject, /* in */ TSS_FLAG attribFlag, /* in */ TSS_FLAG subFlag, /* in */ UINT32 ulAttribDataSize, /* in */ BYTE * rgbAttribData) /* in */
功能描述	设置软件栈对象的属性

参数说明	hObject 需要设置属性的对象句柄 attribFlag 需要设置的属性标记 subFlag 需要设置的子属性标记 ulAttribDataSize rgbAttribData 的大小（单位：字节） rgbAttribData 存放设置属性值的数据
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.1.4 Tspi_GetAttribData

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_GetAttribData /* in */ (TSS_HOBJECT hObject, /* in */ TSS_FLAG attribFlag, /* in */ TSS_FLAG subFlag, UINT32 * pulAttribDataSize, /* out */
	BYTE ** prgbAttribData) /* out */
功能描述	获取软件栈对象的属性
参数说明	hObject 需要设置属性的对象句柄 attribFlag 需要设置的属性标记 subFlag 需要设置的子属性标记 ulAttribDataSize rgbAttribData 的大小（单位：字节） rgbAttribData 存放设置属性值的数据
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.2 上下文管理类

4.8.2.1 Tspi_Context_Create

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_Context_Create(TSS_HCONTEXT * phContext /* out */);
功能描述	创建软件栈 tspi 对象的上下文
参数说明	phContext 上下文句柄
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.2.2 Tspi_Context_Connect

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_Context_Connect(/* in */ TSS_HCONTEXT tspContext, /* in */ TSS_UNICODE *wszDestination)
功能描述	软件栈 tspi 上下文对象绑定
参数说明	tspContext 上下文对象句柄 wszDestination 连接设备（通常传入 NULL 即可）

返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.2.3 Tspi_Context_FreeMemory

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_Context_FreeMemory(TSS_HCONTEXT tspContext, /* in */ BYTE * rgbMemory /* in */);
功能描述	释放上下文内存
参数说明	tspContext 上下文对象句柄 rgbMemory 内容指针（一般传 NULL 即可）
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.2.4 Tspi_Context_Close

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_Context_Close(TSS_HCONTEXT * tspContext /* out */);
功能描述	关闭上下文
参数说明	tspContext 上下文对象句柄

返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.2.5 Tspi_Context_CreateObject

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_Context_CreateObject(TSS_HCONTEXT tspContext,/* in */ TSS_FLAG objectType, /* in */ TSS_FLAG initFlags, /* in */ TSS_HOBJECT * phObject) /* out */
功能描述	创建 tspi 层新对象
参数说明	tspContext 上下文对象句柄 objectType 要创建的对象的类型标识 initFlags 要创建的对象的属性值 phObject 返回所创建对象的句柄
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.3 策略管理类

4.8.3.1 Tspi_Context_GetDefaultPolicy

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容

接口定义	TSS_RESULT Tspi_Context_GetDefaultPolicy(TSS_HCONTEXT tspContext, /* in */ TSS_HPOLICY * phPolicy) /* out */
功能描述	获取上下文的默认策略
参数说明	tspContext 软件栈上下文对象 t 策略对象 phPolicy
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.3.2 Tspi_GetPolicyObject

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_GetPolicyObject /* in */ (TSS_HOBJECT /* in */ hObject, /* out */ TSS_FLAG policyType, TSS_HPOLICY * phPolicy)
功能描述	获取对象的策略
参数说明	hObject 软件栈对象 policyType 策略类型 phPolicy 策略对象
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.3.3 Tspi_Policy_AssignToObject

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel

- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_Policy_AssignToObject(TSS_HPOLICY hPolicy, /* in */ TSS_HOBJECT hObject) /* in */
功能描述	为对象设置策略
参数说明	hPolicy 策略对象 hObject 软件栈对象
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.3.4 Tspi_Policy_SetSecret

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_Policy_SetSecret(
	TSS_HPOLICY hPolicy, /* in */ TSS_FLAG secretMode, /* in */ UINT32 ulSecretLength, /* in */ BYTE * rgbSecret) /* in */
功能描述	为策略设置秘密
参数说明	hPolicy 策略对象 secretMode 秘密模式 ulSecretLength 秘密数据长度 rgbSecret 秘密数据
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.4 TCM 管理类

4.8.4.1 Tspi_TCM_SelfTestFull

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_TCM_SelfTestFull(TSS_HTPM hTPM) /* in */
功能描述	完全自检
参数说明	hTPM TCM 对象
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.4.2 Tspi_TCM_GetTestResult

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_TCM_GetTestResult(TSS_HTPM hTPM, /* in */
	UINT32 * pulTestResultLength, /* out */ BYTE ** prgbTestResult) /* out */
功能描述	获取自检结果
参数说明	hTPM TCM 对象 pulTestResultLength prgbTestResult
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.4.3 Tspi_TCM_SetStatus（所有者可创建模式）

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_TCM_SetStatus(TSS_HTPM hTPM, /* in */ TSS_FLAG statusFlag, /* in */ TSS_BOOL fTpmState) /* in */
功能描述	所有者可创建模式
参数说明	hTPM 软件栈的 TCM 对象句柄 statusFlag 传入 TSS_TPMSTATUS_SETOWNERINSTALL fTPMState 传入 true
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.4.4 Tspi_TCM_SetStatus（所有者设置 TCM 模式）

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_TCM_SetStatus(TSS_HTPM hTPM, /* in */ TSS_FLAG statusFlag, /* in */ TSS_BOOL fTpmState) /* in */
功能描述	所有者设置 TCM 模式
参数说明	hTPM 软件栈的 TCM 对象句柄 statusFlag 传入 TSS_TPMSTATUS_OWNERSETDISABLE fTPMState 传入 true

返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.4.5 Tspi_TCM_SetStatus（物理现场设置可用模式）

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_TCM_SetStatus(TSS_HTPM hTPM, /* in */ TSS_FLAG statusFlag, /* in */ TSS_BOOL fTpmState) /* in */
功能描述	物理现场设置可用模式
参数说明	hTPM 软件栈的 TCM 对象句柄 statusFlag 传入 TSS_TPMSTATUS_PHYSICALDISABLE fTPMState 传入 false
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.4.6 Tspi_TCM_SetStatus（物理现场设置禁用模式）

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_TCM_SetStatus(TSS_HTPM hTPM, /* in */ TSS_FLAG statusFlag, /* in */ TSS_BOOL fTpmState) /* in */
功能描述	物理现场设置禁用模式

参数说明	hTPM 软件栈的 TCM 对象句柄
	statusFlag 传入 TSS_TPMSTATUS_PHYSICALDISABLE fTPMState 传入 true
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.4.7 Tspi_TCM_SetStatus（临时无效模式）

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_TCM_SetStatus(TSS_HTPM hTPM, /* in */ TSS_FLAG statusFlag, /* in */ TSS_BOOL fTpmState) /* in */
功能描述	
参数说明	hTPM 软件栈的 TCM 对象句柄 statusFlag 传入 TSS_TPMSTATUS_SETTEMPDEACTIVATED fTPMState 传入 true
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.4.8 Tspi_TCM_SetStatus（物理现场授权设置无效模式）

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容

接口定义	TSS_RESULT Tspi_TCM_SetStatus(TSS_HTPM hTPM, /* in */ TSS_FLAG statusFlag, /* in */ TSS_BOOL fTpmState) /* in */
功能描述	物理现场授权设置无效模式
参数说明	hTPM 软件栈的 TCM 对象句柄 statusFlag 传入 TSS_TPMSTATUS_DEACTIVATED fTPMState 传入 true，表明可以使用物理现场作为授权方式
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.4.9 Tspi_TCM_SetOperatorAuth

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_TCM_SetOperatorAuth(TSS_HTPM hTpm, /* in */ TSS_HPOLICY hOperatorPolicy) /* in */
功能描述	设置操作员的授权值
参数说明	hTpm 软件栈的 TCM 对象句柄 hOperatorPolicy 新的操作员授权策略对句柄
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.4.10 Tspi_TCM_TakeOwnership

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容

接口定义	TSS_RESULT Tspi_TCM_TakeOwnership(TSS_HTPM hTPM, /* in */ TSS_HKEY hKeySRK, /* in */ TSS_HKEY hEndorsementPubKey) /* in */
功能描述	获取 TCM 的所有者权限， 并设置 OWNER 及 SRK 的授权数据
参数说明	hTPM 软件栈的 TCM 对象句柄 hKeySRK 存储根密钥（SRK）的密钥对象句柄 hEndorsementPubKey 背书密钥（EK）的密钥对象句柄
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.4.11 Tspi_TCM_ClearOwner（所有者授权清除）

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_TCM_ClearOwner(TSS_HTPM hTPM, /* in */ TSS_BOOL fForcedClear) /* in */
功能描述	所有者授权清除
参数说明	hTPM 软件栈的 TCM 对象句柄 fForcedClear 传入 false
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.4.12 Tspi_TCM_ClearOwner（强制清除）

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_TCM_ClearOwner(TSS_HTPM hTPM, /* in */ TSS_BOOL fForcedClear) /* in */
功能描述	强制清除
参数说明	hTPM 软件栈的 TCM 对象句柄 fForcedClear 传入 true
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.4.13 Tspi_TCM_SetStatus（禁用所有者授权清除权）

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_TCM_SetStatus(TSS_HTPM hTPM, /* in */ TSS_FLAG statusFlag, /* in */
	TSS_BOOL fTpmState) /* in */
功能描述	禁用所有者授权清除权
参数说明	hTPM 软件栈的 TCM 对象句柄 statusFlag 传入 TSS_TPMSTATUS_DISABLEOWNERCLEAR fTPMState 传入 false
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.4.14 Tspi_TCM_SetStatus（禁用强制清除权）

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_TCM_SetStatus(TSS_HTPM hTPM, /* in */ TSS_FLAG statusFlag, /* in */ TSS_BOOL fTpmState) /* in */
功能描述	禁用强制清除权
参数说明	hTPM 软件栈的 TCM 对象句柄 statusFlag 传入 TSS_TPMSTATUS_DISABLEFORCECLEAR fTPMState 传入 false
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.4.15 Tspi_TCM_GetCapability

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_TCM_GetCapability(TSS_HTPM hTPM, /* in */ TSS_FLAG capArea, /* in */ UINT32 ulSubCapLength, /* in */ BYTE * rgbSubCap, /* in */
	UINT32 * pulRespDataLength, /* out */ BYTE ** prgbRespData) /* out */
功能描述	获取属性

参数说明	hTPM 软件栈的 TCM 对象句柄 capArea 属性大类 ulSubCapLength 属性子类长度 rgbSubCap 属性子类 pulRespDataLength 返回数据长度 prgbRespData) 返回数据
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.4.16 Tspi_TCM_SetStatus（设置属性）

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_TCM_SetStatus(TSS_HTPM hTPM, /* in */ TSS_FLAG statusFlag, /* in */ TSS_BOOL fTpmState) /* in */
功能描述	设置属性
参数说明	hTPM 软件栈的 TCM 对象句柄 TSS_FLAG 属性标签 statusFlag 属性状态 TSS_BOOL fTpmState
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.4.17 Tspi_TCM_GetPubEndorsementKey（不授权读秘码模块密钥公钥）

- 头文件：tss_tcm/tspi.h

- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_TCM_GetPubEndorsementKey(TSS_HTPM hTPM, /* in */ TSS_BOOL fOwnerAuthorized, /* in */ TSS_VALIDATION *pValidationData, /* in, out */ TSS_HKEY *phEndorsementPubKey) /* out */
功能描述	不授权读秘码模块密钥公钥
参数说明	hTPM 软件栈的 TCM 对象句柄 fOwnerAuthorized 所有者是否已授权，传 false pValidationData 验证信息 phEndorsementPubKey 读取的 EK 信息
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.4.18 Tspi_TCM_GetPubEndorsementKey（授权读秘码模块密钥公钥）

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_TCM_GetPubEndorsementKey(TSS_HTPM hTPM, /* in */ TSS_BOOL fOwnerAuthorized, /* in */ TSS_VALIDATION *pValidationData, /* in, out */ TSS_HKEY *phEndorsementPubKey) /* out */
功能描述	不授权读秘码模块密钥公钥

参数说明	hTPM 软件栈的 TCM 对象句柄 fOwnerAuthorized 所有者是否已授权，传 true pValidationData 验证信息 phEndorsementPubKey 读取的 EK 信息
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.4.19 Tspi_ChangeAuth

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_ChangeAuth(TSS_HOBJECT hObjectToChange, /* in */ TSS_HOBJECT hParentObject, /* in */ TSS_HPOLICY hNewPolicy) /* in */
功能描述	改变对象的授权数据（秘密）
参数说明	hObjectToChange 需要修改授权数据的对象句柄 hParentObject 有 hObjectToChange 所指向对象的父对象句柄 hNewPolicy 更新的授权信息策略对象句柄
返回值说明	成功： TSS_SUCCESS 失败： TSS_E_INVALID_HANDLE TSS_E_BAD_PARAMETER 其它 TCM 错误码等

4.8.5 数据保护操作类

4.8.5.1 Tspi_Data_Seal

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_Data_Seal(TSS_HENCDATA /* in */ hEncData, /* in */ TSS_HKEY hEncKey, /* in */ UINT32 ulDataLength, /* in */ BYTE * rgbDataToSeal, /* in */ TSS_HPCRS hPcrComposite)
功能描述	用来对一个数据块进行加密。如果要对该加密的数据进行解密， 则必须使用同一平台的 Tspi_Data_Unseal()函数
参数说明	hEncData 数据对象句柄， 当命令操作成功后， 该数据对象包含被加密数据 hEncKey 对数据进行加密的密钥对象句柄 ulDataLength 为参数 rgbDataToSeal 的数据长度，以字节为单位 rgbDataToSeal 指向将要被加密的数据 hPcrComposite PCR Composite 对象句柄，包含有 TCM_PCR_INFO 信息
返回值说明	成功： TSS_SUCCESS 失败： TSS_E_INVALID_HANDLE TSS_E_BAD_PARAMETER 其它 TCM 错误码等

4.8.5.2 Tspi_Data_Unseal

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_Data_Unseal(TSS_HENCDATA hEncData, /* in */ TSS_HKEY hKey /* in */ UINT32 * pulUnsealedDataLength, /* out */ BYTE ** prgbUnsealedData) /* out */
功能描述	对指定的数据块进行解密
参数说明	hEncData 包含有加密数据的数据对象句柄 hKey 进行数据解密的密钥对象句柄 pulUnsealedDataLength（单位为字节） 指向 prgbUnsealedData 的数据长度 prgbUnsealedData 若操作成功，指向包含明文信息的指针
返回值说明	成功： TSS_SUCCESS 失败： TSS_E_INVALID_HANDLE
	TSS_E_BAD_PARAMETER TSS_E_OUTOFMEMORY 其它 TCM 错误码等

4.8.6 PCR 管理类

4.8.6.1 Tspi_PcrComposite_GetPcrValue

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_PcrComposite_GetPcrValue(TSS_HPCRS hPcrComposite, /* in */ UINT32 ulPcrIndex, /* in */ UINT32 * /* out */ UINT32 pulPcrValueLength, /* out */ BYTE ** prgbPcrValue)
功能描述	返回在 PCR composite 对象中指定 PCR 索引的摘要值
参数说明	hPcrComposite 软件栈 PCR composite 对象句柄 ulPcrIndex PCR 索引 ulPcrValueLength 参数 rgbPcrValue 的数据长度（单位：字节） rgbPcrValue 返回指定的 PCR 值
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.6.2 Tspi_PcrComposite_SetPcrValue

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_PcrComposite_SetPcrValue(TSS_HPCRS hPcrComposite,/* in */ UINT32 ulPcrIndex, /* in */ UINT32 ulPcrValueLength, /* in */ BYTE * rgbPcrValue) /* in */
功能描述	对一个 PCR composite 对象中给定的 PCR 索引设置摘要值

参数说明	hPcrComposite 软件栈 PCR composite 对象句柄 ulPcrIndex PCR 索引 ulPcrValueLength 参数 rgbPcrValue 的数据长度（单位： 字节） rgbPcrValue 要设置的数据
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.6.3 Tspi_PcrComposite_SetPcrLocality

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_PcrComposite_SetPcrLocality(TSS_HPCRS hPcrComposite,/* in */ UINT32 LocalityValue) /* in */
功能描述	设置 PCR composite 对象中的 LocalityAtRelease 值
参数说明	hPcrComposite 软件栈 PCR composite 对象句柄 LocalityValue 要设置的 LocalityAtRelease 值
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.6.4 Tspi_PcrComposite_SelectPcrIndex

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_PcrComposite_SelectPcrIndex(TSS_HPCRS hPcrComposite, /* in */ UINT32 ulPcrIndex) /* in */

功能描述	在 PCR compiste 对象中选择一个 PCR 索引
参数说明	hPcrComposit 软件栈 PCR composite 对象句柄 e PCR 索引 ulPcrIndex
返回值说明	成功： TSS_SUCCESS
	失败： 非零

4.8.6.5 Tspi_TCM_PcrRead

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_TCM_PcrRead(TSS_HTPM hTPM, /* in */ UINT32 ulPcrIndex, /* in */ UINT32 *pulPcrValueLength, /* out */ BYTE **prgbPcrValue) /* out */
功能描述	写入平台配置寄存器
参数说明	hTPM 软件栈的 TCM 对象句柄 ulPcrIndex PCR 索引值 pulPcrValueLength 读取的 PCR 数据长度 prgbPcrValue 读取的 PCR 数据
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.6.6 Tspi_TCM_PcrReset

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容

接口定义	TSS_RESULT Tspi_TCM_PcrReset(/* in */ TSS_HTPM hTPM, /* in */ TSS_HPCRS hPcrComposite)
功能描述	复位平台配置寄存器
参数说明	hTPM 软件栈的 TCM 对象句柄 hPcrComposite 待复位的 PCR 组合信息
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.6.7 Tspi_TCM_PcrExtend

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_TCM_PcrExtend(TSS_HTPM hTPM, /* in */ UINT32 ulPcrIndex, /* in */ UINT32 ulPcrDataLength, /* in */ BYTE *pbPcrData, /* in */ TSS_PCR_EVENT *pPcrEvent, /* in */ UINT32 * pulPcrValueLength, /* out */ BYTE ** prgbPcrValue) /* out */
功能描述	扩展平台配置寄存器

参数说明	hTPM	软件栈的 TCM 对象句柄
	ulPcrIndex	PCR 索引值
	ulPcrDataLength	扩展值
	pbPcrData	扩展值得长度
	pPcrEvent	PCR 扩展事件
	pulPcrValueLength	扩展后的 PCR 数据长度
	prgbPcrValue	扩展后的 PCR 数据
返回值说明	成功： TSS_SUCCESS 失败： 非零	

4.8.6.8 Tspi_TCM_Quote

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_TCM_Quote(TSS_HTPM hTPM, /* in */ TSS_HKEY hIdentKey, /* in */ TSS_HPCRS hPcrComposite, /* in */ TSS_VALIDATION * pValidationData) /* in, out */
功能描述	该函数调用 TCM_Quote 命令，提供当前平台的完整性信息
参数说明	hTPM TCM 对象句柄 hIdentKey 身份密钥对象句柄，用于 TCM 内部对 PCR 数据签名 hPcrComposite PcrComposite 对象句柄，表示所选择的 PCR 寄存器集合 pValidationData 验证数据结构， 其中：

	1) 输入时，需要提供 externalData（一般用 32 字节的随机数填 充） 2) 输出时，validationData 返回 PCR 的签名数据
返回值说明	成功： TSS_SUCCESS 失败： TSS_E_INVALID_HANDLE TSS_E_BAD_PARAMETER TSS_E_OUTOFMEMORY 其它 TCM 错误码等

4.8.7 密钥管理类

4.8.7.1 Tspi_Key_CreateKey

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_Key_CreateKey(TSS_HKEY hKey, /* in */ TSS_HKEY hWrappingKey, /* in */ TSS_HPCRS hPcrComposite) /* in, may be NULL */
功能描述	在 TCM 内部创建密钥（对称密钥或非对称密钥对）
参数说明	hKey 要创建的密钥对象句柄 hWrappingKey 用来加密新创建的密钥的密钥句柄 hPcrComposite 将新创建的密钥绑定到 PCR（可以传入 NULL）
返回值说明	成功： TSS_SUCCESS
	失败： TSS_E_INVALID_HANDLE TSS_E_BAD_PARAMETER TSS_E_OUTOFMEMORY 其它 TCM 错误码等

4.8.7.2 Tspi_Key_LoadKey

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_Key_LoadKey(TSS_HKEY hKey, /* in */ TSS_HKEY hUnwrappingKey) /* in */
功能描述	将主机的密钥加载到 TCM 中，TCM 负责将密钥解密并缓存加载在 TCM 中； 只有执行本命令后， 才能使用密钥进行加解密、签名等 密钥服务功能
参数说明	hKey 被载入的密钥对象句柄 hUnwrappingKey 用于解开 hKey 的密钥句柄（即 hKey 的父密 钥）
返回值说明	成功： TSS_SUCCESS 失败： TSS_E_INVALID_HANDLE TSS_E_BAD_PARAMETER TSS_E_OUTOFMEMORY 其它 TCM 错误码等

4.8.7.3 Tspi_Key_GetPubKey

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容

接口定义	TSS_RESULT Tspi_Key_GetPubKey(TSS_HKEY hKey, /* in */ UINT32 * pulPubKeyLength, /* out */
	BYTE ** prgbPubKey) /* out */
功能描述	获取已经加载密钥的公钥
参数说明	hKey 密钥对象句柄 pulPubKeyLength 接收 prgbPubKey 参数的数据长度（单位：字节） prgbPubKey 指向内存中 hKey 密钥对象的公钥（采用 TCM_PUBKEY 结构）
返回值说明	成功： TSS_SUCCESS 失败： TSS_E_INVALID_HANDLE TSS_E_BAD_PARAMETER TSS_E_OUTOFMEMORY 其它 TCM 错误码等

4.8.8 密码服务（密码算法）类

4.8.8.1 Tspi_TCM_GetRandom

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_TCM_GetRandom(TSS_HTPM hTPM, /* in */ UINT32 ulRandomDataLength, /* in */ BYTE ** prgbRandomData) /* out */
功能描述	获取随机数

参数说明	hTPM TCM 对象句柄 ulRandomDataLength 随机数长度（单位：字节） prgbRandomData 产生的随机数数据
返回值说明	成功： TSS_SUCCESS 失败： 非零

4.8.8.2 Tspi_Data_Encrypt

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_Data_Encrypt(TSS_HENCDATA hEncData, /* in */ TSS_HKEY hEncKey, /* in */ TSS_BOOL bFinal, /* in */ BYTE* rgbDataIV, /* in */ BYTE* rgbDataToEncrypt, /* in */ UINT32 ulDataLength) /* in */
功能描述	对明文数据进行加密。 采用何种密码算法取决于密钥属性。目前已支持 SM4。
参数说明	hEncData 含有被加密数据的对象句柄 hEncKey 用于加密数据的密钥对象句柄 bFinal 用于标识是否是应用加密的明文最后一个分组 rgbDataIV 用于对称加密使用的 IV，固定 16 字节。可以为空。若为空，则使用 TSPI 接口提供的默认 IV 值 rgbDataToEncrypt 明文数据 ulDataLength 明文数据字节长度

返回值说明	成功： TSS_SUCCESS 失败： TSS_E_INVALID_HANDLE TSS_E_BAD_PARAMETER 其它 TCM 错误码等

4.8.8.3 Tspi_Data_Decrypt

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_Data_Decrypt(TSS_HENCDATA hEncData, /* in */ TSS_HKEY hEncKey, /* in */ TSS_BOOL bFinal, /* in */ BYTE* rgbDataIV, /* in */ UINT32* ulDataLength, /* out */ BYTE** rgbDataDecrypted) /* out */
功能描述	对密文数据进行解密。 采用何种算法取决于密钥属性。
参数说明	hEncData 含有被加密数据的对象句柄 hEncKey 用于解密数据的密钥对象句柄 bFinal 用于标识是否是应用解密的密文最后一个分组 rgbDataIV 用于解密使用的 IV，固定 16 字节 ulDataLength 明文数据字节长度 rgbDataDecrypted 一旦命令执行成功，指向包含解密数据的存储区

返回值说明	成功： TSS_SUCCESS 失败： TSS_E_INVALID_HANDLE TSS_E_BAD_PARAMETER 其它 TCM 错误码等

4.8.8.4 Tspi_Sm2_Decrypt

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_Sm2_Decrypt(/* in */ TSS_HKEY hKey, /* in */ UINT32 inDataLen, /* in */ BYTE* inData, /* out */ UINT32 outDataLen, /* out */ * outData) BYTE**
功能描述	对明文数据进行 SM2 加密
参数说明	hKey 用于加密数据的密钥对象句柄 inDataLen 明文数据字节长度 inData 明文数据 outDataLen 加密后的密文数据长度 outData 加密后的密文数据
返回值说明	成功： TSS_SUCCESS 失败： TSS_E_INVALID_HANDLE TSS_E_BAD_PARAMETER 其它 TCM 错误码等

4.8.8.5 Tspi_Hash_Sign

- 头文件：tss_tcm/tspi.h

- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_Hash_Sign(TSS_HHASH hHash, /* in */ TSS_HKEY hKey, /* in */ UINT32 * pulSignatureLength, /* out */ BYTE ** prgbSignature) /* out */
功能描述	对经过 SM3 哈希计算所得的摘要数据进行 SM2 签名
参数说明	hHash 用于签名的哈希对象句柄 hKey 用于签名的密钥对象句柄 pulSignatureLength 签名数据长度 prgbSignature 签名数据
返回值说明	成功： TSS_SUCCESS 失败： TSS_E_INVALID_HANDLE TSS_E_BAD_PARAMETER 其它 TCM 错误码等

4.8.9 NV 存储管理类

4.8.9.1 Tspi_NV_DefineSpace

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_NV_DefineSpace(TSS_HNVSTORE hNvstore, /* in */ TSS_HPCRS hReadPcrComposite, /* in, may be NULL */

	TSS_HPCRS hWritePcrComposite) /* in, may be NULL*/
功能描述	创建 NV 存储空间
参数说明	hNvstore NV 存储对象句柄 hReadPcrComposite 将 PCR 绑定到 NV 读操作（可传入 NULL） hWritePcrComposite 将 PCR 绑定到 NV 写操作（可传入 NULL）
返回值说明	成功： TSS_SUCCESS 失败： TSS_E_INVALID_HANDLE TSS_E_BAD_PARAMETER TSS_E_OUTOFMEMORY 其它 TCM 错误码等

4.8.9.2 Tspi_NV_ReleaseSpace

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_NV_ReleaseSpace(TSS_HNVSTORE hNvstore) /* in */
功能描述	释放已创建的 NV 存储空间
参数说明	hNvstore NV 存储对象句柄
返回值说明	成功： TSS_SUCCESS 失败： TSS_E_INVALID_HANDLE TSS_E_BAD_PARAMETER TSS_E_OUTOFMEMORY
	其它 TCM 错误码等

4.8.9.3 Tspi_NV_WriteValue

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_NV_WriteValue(TSS_HNVSTORE hNvstore, /* in */ UINT32 offset, /* in */ UINT32 ulDataLength, /* in */ BYTE* rgbDataToWrite) /* in */
功能描述	向指定的 NV 存储空间写入数据
参数说明	hNvstore NV 存储对象句柄 offset 在 NV 区域的写入起始地址 ulDataLength 写入数据长度 rgbDataToWrite 写入数据
返回值说明	成功： TSS_SUCCESS 失败： TSS_E_INVALID_HANDLE TSS_E_BAD_PARAMETER TSS_E_OUTOFMEMORY 其它 TCM 错误码等

4.8.9.4 Tspi_NV_ReadValue

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容

接口定义	TSS_RESULT Tspi_NV_ReadValue(TSS_HNVSTORE hNvstore, /* in */ UINT32 offset, /* in */ UINT32* ulDataLength, /* in, out */ BYTE** rgbDataRead) /* out */
功能描述	从指定的 NV 存储空间读取数据
参数说明	hNvstore NV 存储对象句柄 offset 从 NV 区域的读取起始地址 ulDataLength 读取数据长度 rgbDataToWrite 读取数据
返回值说明	成功： TSS_SUCCESS 失败： TSS_E_INVALID_HANDLE TSS_E_BAD_PARAMETER TSS_E_OUTOFMEMORY 其它 TCM 错误码等

4.8.10 杂凑计算类

4.8.10.1 Tspi_Hash_SM3Start

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_Hash_SM3Start(TSS_HCONTEXT /* in */ tspContext, /* out */ UINT32* maxSM3UpdateBits)
功能描述	SM3 哈希初始化
参数说明	tspContext 软件栈上下文句柄 maxSM3UpdateBits SM3 最大哈希长度（单位：比特）

返回值说明	成功： TSS_SUCCESS 失败： TCM 错误码等

4.8.10.2 Tspi_Hash_SM3Update

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_Hash_SM3Update(/* in */ TSS_HCONTEXT tspContext, /* in */ BYTE* rgbData, /* in */ UINT32 ulDataLength)
功能描述	SM3 哈希多包运算
参数说明	tspContext 软件栈上下文 rgbData 待哈希数据 ulDataLength 待哈希数据长度
返回值说明	成功： TSS_SUCCESS 失败： TCM 错误码等

4.8.10.3 Tspi_Hash_SM3Complete

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_Hash_SM3Complete(TSS_HCONTEXT tspContext, /* in */ BYTE* rgbData, /* in */ UINT32 ulDataLength, /* in */ BYTE* rgbHashResult) /* out */

功能描述	SM3 完成哈希运算
参数说明	tspContext 软件栈上下文 rgbData 待哈希数据 ulDataLength 待哈希数据长度
返回值说明	成功： TSS_SUCCESS 失败： TCM 错误码等

4.8.10.4 Tspi_Hash_SM3CompleteExtend

- 头文件：tss_tcm/tspi.h
- 开发依赖包： trousers-tcm-devel
- 运行依赖包：trousers-tcm

条目	详细内容
接口定义	TSS_RESULT Tspi_Hash_SM3CompleteExtend(TSS_HCONTEXT tspContext, /* in */ UINT32 ulPCRIndex, /* in */ BYTE* rgbData, /* in */ UINT32 ulDataLength, /* in */
	BYTE* rgbHashResult, /* out */ BYTE* rgbPCRResult) /* out */
功能描述	SM3 完成哈希运算并写入平台寄存器
参数说明	tspContext 软件栈上下文 ulPCRIndex 结果写入的 PCR 寄存器索引 rgbData 待哈希数据 ulDataLength 待哈希数据长度 rgbHashResult 哈希计算结果 rgbPCRResult PCR 扩展后的值
返回值说明	成功： TSS_SUCCESS 失败： TCM 错误码等

4.9 飞腾机密计算

硬件与软件的对应关系

CPU	固件	驱动	开发包	备注
2000+	601FBS34GW	内核中内置	libkysec-tee	未充分适配
S2500	G625FBS57GW_RELEASE.bin	内核中内置	libkysec-tee	
S5000	进行中	进行中	进行中	

通用数据结构

KYSEC_TEE_DATA：

```
typedef struct KYSEC_TEE_DATA {
  uint8_t *dptr; //TEE 数据数据，数据 buf 最大长度 4096 字节
  uint32_t dsize; //载荷数据长度
} KYSEC_TEE_DATA;
```

KYSEC TEE通用数据结构，当其作为接收TEE数据参数时，调用者获取数据并使用结束后，dptr内存需调用者释放

ID定义：

```
#define SOS_ID_MAX_LEN 32 //安全存储 ID 最大长度
typedef unsigned char SOS_ID[SOS_ID_MAX_LEN + 1]; //安全存储对象 ID
#define KM_ID_MAX_LEN 32 //密钥管理对象 ID 最大长度
typedef unsigned char KM_ID[KM_ID_MAX_LEN + 1]; //密钥存储 ID
```

4.9.1 安全存储模块

- 头文件：kysec_tee_def.h、kysec_tee_sos.h
- 开发包：libkysec-tee-devel
- 接口包：libkysec-tee
- 接口库：libkysec_tee.so

安全存储对象存储

声明	描述	参数说明	返回值说明

extern KT_Result kysec_tee_sos_store(const SOS_ID sos_id, const KYSEC_TEE_DATA data);	将安全存储对象存入TEE，若安全存储对象key已存在，不会更新其数据	sos_id: 传入参数，安全存储对象id，最大长度32字节，包含Field和Description两部分，格式为 Field:Description，通过":"分隔 data: 传入参数， 需存储TEE的安全存储对象数据载荷	0: 调用成功 其他值: 失败

安全存储对象数据更新

声明	描述	参数说明	返回值说明
extern KT_Result kysec_tee_sos_update(const SOS_ID sos_id, const KYSEC_TEE_DATA data);	更新TEE中对应id的数据，若安全存储对象key不存在，不会新建数据存入	sos_id: 传入参数，安全存储对象id，最大长度32字节，包含Field和Description两部分， 格式为 Field:Description，通过":"分隔 data: 传入参数， 需更新TEE的安全存储对象数据载荷	0: 调用成功 其他值: 失败

删除TEE中安全存储对象

声明	描述	参数说明	返回值说明
extern KT_Result kysec_tee_sos_delete(const SOS_ID sos_id);	删除TEE中对应id的安全存储对象	sos_id: 传入参数，安全存储对象id，最大长度32字节，包含Field和Description两部分， 格式为 Field:Description，通过":"分隔	0: 调用成功 其他值: 失败

获取TEE中安全存储对象

声明	描述	参数说明	返回值说明

extern KT_Result kysec_tee_sos_fetch(const SOS_ID sos_id, KYSEC_TEE_DATA *data);	获取TEE中对应id的安全存储对象	sos_id: 传入参数，安全存储对象id，最大长度32字节，包含Field和Description两部分， 格式为 Field:Description，通过":"分隔 data: 获取数据存放地址，内存核外申请。	0: 调用成功 其他值: 失败

4.9.2 密钥服务模块

- 头文件：kysec_tee_def.h、kysec_tee_km.h
- 开发包：libkysec-tee-dev
- 接口包：libkysec-tee
- 接口库：libkysec_tee

在TEE中生成SM2公私钥对

声明	描述	参数说明	返回值说明
extern KT_Result kysec_tee_km_sm2_keygen(const KM_ID km_id, uint8_t pub_key[64]);	在TEE中生成SM2公私钥对，数据不可读取，可指定私钥进行相关计算	km_id: 传入参数，安全存储对象id，最大长度32字节，包含Field和Description两部分， 格式为 Field:Description，通过":"分隔 pub_key: 传出参数，内部生成的sm2公钥	0: 调用成功 其他值: 失败

指定TEE中SM2私钥签名

声明	描述	参数说明	返回值说明
extern KT_Result kysec_tee_km_sm2_sign(const KM_ID km_id, KYSEC_TEE_DATA data, uint8_t sign_data[64]);	通过在TEE中生成SM2公私钥中的私钥对数据签名	km_id: 传入参数，生成私钥时id，最大长度32字节，包含Field和Description两部分，格式为 Field:Description，通过":"分隔 data: 需签名数	0: 调用成功 其他值: 失败

		据 sign_data: 签名数据	

指定TEE中SM2验签

声明	描述	参数说明	返回值说明
extern KT_Result kysec_tee_km_sm2_verify(const KM_ID km_id, KYSEC_TEE_DATA data, uint8_t sign_data[64]);	通过在TEE中生成SM2公私钥对数据验签	km_id: 传入参数，生成私钥时id，最大长度32字节，包含Field和Description两部分，格 式为 Field:Description，通过":"分隔 data: 需验签数据 sign_data: 签名数据	0: 验签通过 1: 验签失败 其他值: 失败

4.9.3 算法服务模块

- 头文件：kysec_tee_def.h、kysec_tee_algo.h
- 开发包：libkysec-tee-dev
- 接口包：libkysec-tee
- 接口库：libkysec_tee

SM2算法

生成SM2公私钥对

声明	描述	参数说明	返回值说明
int kysec_tee_algo_sm2_keygen(uint8_t pub_key[64], uint8_t pri_key[32])	生成sm2公私钥对	pub_key: 存放生成的公钥 pri_key:存放生成的私钥	0: 调用成功 其他值: 失败

sm2数据签名

声明	描述	参数说明	返回值说明

int kysec_tee_algo_sm2_sign(KYSEC_TEE_DATA data, const uint8_t pub_key[64], const uint8_t pri_key[32], uint8_t sign_data[64])	sm2数据签名	data: 待签名数据 pub_key:sm2公钥 pri_key:sm2私钥 sign_data:存放签名	0: 调用成功 其他值: 失败

sm2数据验签

声明	描述	参数说明	返回值说明
int kysec_tee_algo_sm2_verify(KYSEC_TEE_DATA data, const uint8_t pub_key[64], uint8_t sign_data[64])	sm2数据验签	data: 待验签数据 pub_key:sm2公钥 sign_data: 签名	0: 验签成功 其他值: 验签失败

sm2数据加密

声明	描述	参数说明	返回值说明
int kysec_tee_algo_sm2_encrypt(KYSEC_TEE_DATA data, const uint8_t pub_key[64], KYSEC_TEE_DATA *out_data)	sm2数据加密	data: 待加密数据 pub_key:公钥 out_data: 加密数据，内存外部申请，内部赋值	0: 调用成功 其他值: 失败

sm2数据解密

声明	描述	参数说明	返回值说明
int kysec_tee_algo_sm2_decrypt(KYSEC_TEE_DATA data, const uint8_t pub_key[64], const uint8_t pri_key[32], KYSEC_TEE_DATA *out_data)	sm2数据解密	data: 待解密数据 pub_key:公钥 pri_key:私钥 out_data: 解密数据，内存外部申请，内部赋值	0: 调用成功 其他值: 失败

SM3算法

KYSEC TEE算法服务sm3算法上下文：

```
typedef struct ALGO_SM3_CTX {
  uint8_t content[128]; //上下文数据
```

```
  uint8_t ctx[128];
}ALGO_SM3_CTX;
```

sm3算法上下文初始化

声明	描述	参数说明	返回值说明
int kysec_tee_algo_sm3_init(ALGO_SM3_CTX **context)	初始化sm3算法上下文	context: sm3算法上下文，内存接口内部申请，外部释放	0: 调用成功 其他值: 失败

sm3数据处理

声明	描述	参数说明	返回值说明
int kysec_tee_algo_sm3_update(ALGO_SM3_CTX *context, KYSEC_TEE_DATA data)	sm3数据处理	context: sm3算法上下文，通过init申请 data: 待处理数据	0: 调用成功 其他值: 失败

sm3计算完成

声明	描述	参数说明	返回值说明
int kysec_tee_algo_sm3_finalize(ALGO_SM3_CTX *context, uint8_t digest[32])	sm3计算完成	context: sm3算法上下文，通过init申请 digest: 传出，摘要值	0: 调用成功 其他值: 失败

SM4算法

KYSEC TEE算法服务sm4算法上下文：

```
typedef struct ALGO_SM4_CTX {
  uint8_t content[128]; //content: 上下文数据
  uint8_t ctx[128];
}ALGO_SM4_CTX;
```

KYSEC TEE算法服务sm4算法操作模式：

```
typedef enum ALGO_SM4_OP_MODE {
 SM4_MODE_ECB = 0, // ECB Mode
 SM4_MODE_CBC, // CBC Mode
```

```
 SM4_MODE_MAX
}ALGO_SM4_OP_MODE;
```

KYSEC TEE算法服务sm4算法操作类型：

```
typedef enum ALGO_SM4_OP_TYPE {
  SM4_CRYPTO_ENCRYPT = 0, // encrypt
  SM4_CRYPTO_DECRYPT , // decrypt
  SM4_CRYPTO_MAX
}ALGO_SM4_OP_TYPE;
```

sm4上下文初始化

声明	描述	参数说明	返回值说明
int kysec_tee_algo_sm4_init(ALGO_SM4_CTX **context, ALGO_SM4_OP_MODE mode, ALGO_SM4_OP_TYPE type, const uint8_t key[16], const uint8_t iv[16])	初始化sm4上下文	context: sm4算法上下文，，内存接口内部申请，外部释放 mode: 加解密模式，ECB或CBC type: 加解密操作类型，加密或解密 key: 秘钥 iv: 向量，CBC模式使用	0: 调用成功 其他值: 失败

sm4算法操作执行

声明	描述	参数说明	返回值说明
int kysec_tee_algo_sm4_update(ALGO_SM4_CTX *context, ALGO_SM4_OP_TYPE type, KYSEC_TEE_DATA data, const uint32_t flag, KYSEC_TEE_DATA *out_data)	sm4算法操作执行	context: sm4上下文，与init时同一对象 type: 加解密操作类型，加密或解密 data: 待处理数据 flag: 处理标记位，0 中间数据包， 1 结束数据包 out_data:输出数据	0: 调用成功 其他值: 失败

随机数

生成随机数

声明	描述	参数说明	返回值说明
int kysec_tee_algo_random(uint32_t len, KYSEC_TEE_DATA *data)	生成随机数	len: 需生成随机数位数 data: 生成随机数结果，外部申请内存，内部赋值	0: 调用成功 其他值: 失败

附录 A 代码示例

A.1 应用执行控制接口示例

编译：gcc appexectlsectest.c -lkyappexectlsec -ldl -lkysdk_module_authorize

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <unistd.h>
#include <string.h>
#include <getopt.h>
#include "libkyappexectlsec.h"
#define WIDTH 4

#include "kysecuritytool.h"
#define PKG_LIST_LEN_MAX    256

static int ret;

static void usage(void)
{
    printf("Usage: appexectltest [ -s | -c ] packagename | [ -h ] \n");
    printf("%*s-s packagename\n",WIDTH,"");
    printf("%*s--set packagename\n",WIDTH,"");
    printf("\t  set control for package\n\n");
    printf("%*s-c packagename\n",WIDTH,"");
    printf("%*s--clear packagename\n",WIDTH,"");
    printf("\t  clear control for package\n\n");
    printf("%*s-h\n",WIDTH,"");
    printf("%*s--help\n",WIDTH,"");
    printf("\t  help information\n\n");
    printf("eg:\n");
    printf("%*sappexectltest -s rpm-test\n",WIDTH,"");
    printf("%*sappexectltest -c rpm-test\n\n",WIDTH,"");
}

int test_kdk_set_app_can_exec(char *pkgname){
    //int ret = -2;
    const char *name = pkgname;
    ret = kdk_set_app_can_exec(name);
    printf("\tpkgname : %s, ret: %d\n",name, ret);
    return ret;
}

int test_kdk_clear_app_can_exec(char *pkgname){
    //int ret = -2;
    const char *name = pkgname;
    ret =  kdk_clear_app_can_exec(name);
    printf("\tpkgname : %s, ret: %d\n",name, ret);
    return ret;
}

int main(int argc, char *argv[])
{
    if (argc == 1)
    {
        usage();
        return -1;
    }

    int opt = 0;
    int option_index = 0;
    unsigned int pkg_list_len = 0;
    char* pkg_list[PKG_LIST_LEN_MAX] = {0};
    int iRet;

    static struct option long_options[] = {
```

```
      {"set", required_argument, 0, 's'},
      {"clear", required_argument, 0, 'c'},
      {"help", no_argument, 0, 'h'},
      {0, 0, 0, 0}
   };

   while ((opt = getopt_long(argc, argv, "s:c:h", long_options, &option_index)) != -1)
   {
      switch (opt)
      {
      case 's':
        ret = test_kdk_set_app_can_exec(optarg);
          break;
      case 'c':
          ret = test_kdk_clear_app_can_exec(optarg);
          break;
      case 'h':
          usage();
          break;
      default:
          usage();
          return -1;
      }
   }

   return 0;
}
```

A.2 软件包安装黑白名单接口示例

编译：gcc appinstsectest.c -lkyappinstsec -ldl -lkysdk_module_authorize

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <getopt.h>
#include "libkyappinstsec.h"
#define WIDTH 4

static int ret;

static void usage(void)
{
   printf("Usage: appinstsectest [ -g | -s type ] | [-a | -d ] packagename type | [ -h ]
\n");
   printf("%*s-s type\n",WIDTH,"");
   printf("%*s--set type\n",WIDTH,"");
   printf("\t type value: KDK_PKG_BWLIST_TYPE_NORMAL(0),
KDK_PKG_BWLIST_TYPE_WHITELIST(1), KDK_PKG_BWLIST_TYPE_BLACKLIST(2)\n");
   printf("\t set package bwlist mode\n\n");
   printf("%*s-g\n",WIDTH,"");
   printf("%*s--get\n",WIDTH,"");
   printf("\t get package bwlist mode\n\n");
   printf("%*s-a packagename type\n",WIDTH,"");
```

```
    printf("%*s--add packagename type\n",WIDTH,"");
    printf("\t add package in bwlist\n\n");
    printf("%*s-d packagename type\n",WIDTH,"");
    printf("%*s--del packagename type\n",WIDTH,"");
    printf("\t delete package from bwlist\n\n");
    printf("%*s-h\n",WIDTH,"");
    printf("%*s--help\n",WIDTH,"");
    printf("\t help information\n");
    printf("eg:\n");
    printf("%*sappinstsectest -s 1\n",WIDTH,"");
    printf("%*sappinstsectest -g\n",WIDTH,"");
    printf("%*sappinstsectest -a python2 1\n",WIDTH,"");
    printf("%*sappinstsectest -d python2 1\n\n",WIDTH,"");
}

int test_kdk_pkg_bwlist_set_model(int type){
    //int ret = -1;
    ret = kdk_pkg_bwlist_set_model(type);
    printf("\ttype : %d, ret: %d\n",type, ret);
    return ret;
}

int test_kdk_pkg_bwlist_get_mode(){
    //int ret = -1;
    ret = kdk_pkg_bwlist_get_mode();
    printf("\tret: %d\n",ret);
    return ret;
}

int test_kdk_pkg_bwlist_add(int type, char *pkgname){
    //int ret = -1;
    const char *name =  pkgname;
    ret = kdk_pkg_bwlist_add(type,name);
    printf("\tpkgname : %s, ret: %d\n",name, ret);
    return ret;
}

int test_kdk_pkg_bwlist_del(int type, char *pkgname){
    //int ret = -1;
    const char *name = pkgname;
    ret =  kdk_pkg_bwlist_del(type, name);
    printf("\tpkgname : %s, ret: %d\n",name, ret);
    return ret;
}

int main(int argc, char *argv[])
{
    if (argc == 1)
    {
        usage();
        return -1;
    }

    int opt = 0;
    int option_index = 0;
    int type = 0;
    //int ret = -1;

    static struct option long_options[] = {
```

```
      {"set", required_argument, 0, 's'},
      {"add", required_argument, 0, 'a'},
      {"del", required_argument, 0, 'd'},
      {"get", no_argument, 0, 'g'},
      {"help", no_argument, 0, 'h'},
      {0, 0, 0, 0}
   };

   while ((opt = getopt_long(argc, argv, "s:a:d:gh", long_options, &option_index)) != -
1)
   {
      switch (opt)
      {
      case 's':
        ret = test_kdk_pkg_bwlist_set_model(atoi(optarg));
        printf("type: %s, %d\n",optarg, atoi(optarg));
          break;
      case 'g':
        ret = test_kdk_pkg_bwlist_get_mode();
          break;
      case 'a':
        if (optind < argc && argc == 4){
          if(argv[optind][0] != '-'){
              printf("package_name:%s, type: %s\n",optarg, argv[optind]);
              type = atoi(argv[optind]);
              ret = test_kdk_pkg_bwlist_add(type, optarg);
          }
        }else{
          usage();
        }
          break;
      case 'd':
        if (optind < argc && argc == 4){
          if(argv[optind][0] != '-'){
              printf("package_name:%s, type: %s\n",optarg, argv[optind]);
              type = atoi(argv[optind]);
              ret = test_kdk_pkg_bwlist_del(type, optarg);
          }
        }else{
          usage();
        }
          break;
      case 'h':
          usage();
          break;
      default:
          usage();
          return -1;
      }
   }

   return 0;
}
```

A.3 软件包防卸载接口示例

编译：gcc appuninstsec.c -lkyappuninstsec -ldl -lkysdk_module_authorize

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <getopt.h>
#include "libkyappuninstsec.h"
#define WIDTH 4

static int ret;

static void usage(void)
{
   printf("Usage: appuninstsectest [ -a | -d ] packagename | [ -h ]\n");
   printf("%*s-a packagename\n",WIDTH,"");
   printf("%*s--add packagename\n",WIDTH,"");
   printf("\t set package uninstall property\n\n");
   printf("%*s-d packagename\n",WIDTH,"");
   printf("%*s--del packagename\n",WIDTH,"");
   printf("\t cancel package uninstall property\n\n");
   printf("%*s-h\n",WIDTH,"");
   printf("%*s--help\n",WIDTH,"");
   printf("\t help information\n\n");
   printf("eg:\n");
   printf("%*sappuninstsectest -a rpm-test\n",WIDTH,"");
   printf("%*sappuninstsectest -d rpm-test\n\n",WIDTH,"");
}

int test_pkg_uninstall_add(char *pkgname){
   //int ret = -1;
   const char *name =  pkgname;
   ret = pkg_uninstall_add(name);
   printf("\tpkgname : %s, ret: %d\n",name, ret);
   return ret;
}

int test_pkg_uninstall_del(char *pkgname){
   //int ret = -1;
   const char *name =  pkgname;
   ret =  pkg_uninstall_del(name);
   printf("\tpkgname : %s, ret: %d\n",name, ret);
   return ret;
}

int main(int argc, char *argv[])
{
   if (argc == 1)
   {
      usage();
      return -1;
   }

   int opt = 0;
```

```
    int option_index = 0;
    //int ret = -1;

    static struct option long_options[] = {
        {"add", required_argument, 0, 'a'},
        {"del", required_argument, 0, 'd'},
        {"help", no_argument, 0, 'h'},
        {0, 0, 0, 0}
    };

    while ((opt = getopt_long(argc, argv, "a:d:h", long_options, &option_index)) != -1)
    {
        switch (opt)
        {
        case 'a':
          ret = test_pkg_uninstall_add(optarg);
            break;
        case 'd':
          ret = test_pkg_uninstall_del(optarg);
            break;
        case 'h':
            usage();
            break;
        default:
            usage();
            return -1;
        }
    }

    return 0;
}
```

A.4 内核模块防卸载接口示例

编译：gcc kmodsectest.c -lkykmodsec -ldl -lkysdk_module_authorize

```
#include <stdio.h>
#include <getopt.h>

#include <libkykmodsec.h>
#define WIDTH 4

static int ret;

static void usage(void)
{
    printf("Usage: kmodsectest [ -s | -c ] filename | [ -h ]\n");
    printf("%*s-s filename\n",WIDTH,"");
    printf("%*s--set filename\n",WIDTH,"");
    printf("\t set kernel module unloaded property\n\n");
    printf("%*s-c filename\n",WIDTH,"");
    printf("%*s--clear filename\n",WIDTH,"");
    printf("\t clear kernel module unloaded property\n\n");
    printf("%*s-h\n",WIDTH,"");
    printf("%*s--help\n",WIDTH,"");
```

```
    printf("\t help information\n\n");
    printf("eg:\n");
    printf("%*skmodsectest -s /opt/qianxin.com/qaxbrowser/qaxbrowser\n",WIDTH,"");
    printf("%*skmodsectest -c
/opt/qianxin.com/qaxbrowser/qaxbrowser\n\n",WIDTH,"");
}

int test_kdk_kmod_set_anti_unloaded(char *filename){
    //int ret = -1;
    const char *file_name =  filename;
    ret = kdk_kmod_set_anti_unloaded(file_name);
    printf("\tfilename: %s, ret: %d\n",file_name, ret);
    return ret;
}

int test_kdk_kmod_cancel_anti_unloaded(char *filename){
    //int ret = -1;
    const char *file_name = filename;
    ret =  kdk_kmod_cancel_anti_unloaded(file_name);
    printf("\tfilename: %s, ret: %d\n",file_name, ret);
    return ret;
}

int main(int argc, char *argv[])
{
    if (argc == 1)
    {
        usage();
        return -1;
    }

    int opt = 0;
    int option_index = 0;
    //int ret = -1;

    static struct option long_options[] = {
        {"set", required_argument, 0, 's'},
        {"clear", required_argument, 0, 'c'},
        {"help", no_argument, 0, 'h'},
        {0, 0, 0, 0}
    };

    while ((opt = getopt_long(argc, argv, "s:c:h", long_options, &option_index)) != -1)
    {
        switch (opt)
        {
        case 's':
          ret = test_kdk_kmod_set_anti_unloaded(optarg);
            break;
        case 'c':
          ret = test_kdk_kmod_cancel_anti_unloaded(optarg);
            break;
        case 'h':
            usage();
            break;
        default:
            usage();
            return -1;
        }
```

```
    }

    return 0;
```

A.5 安全审计接口示例

编译：gcc auditsectest.c -lkyauditsec -ldl -lkysdk_module_authorize

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <getopt.h>
#include "libkyauditsec.h"
#define WIDTH 4

typedef struct{
    int type;
    char* msg;
    char* hostname;
    char* addr;
    char* tty;
    char* result;
}audit_log_t;

static audit_log_t audit_log;

static void usage(void)
{
    printf("Usage: auditsectest [ -g | -s ] status | -a type msg hostname addr
tty result | [ -h ] \n");
    printf("%*s-s status\n",WIDTH,"");
    printf("%*s--set status\n",WIDTH,"");
    printf("\t status: 0,close; 1,open\n");
    printf("\t set audit status\n\n");
    printf("%*s-g\n",WIDTH,"");
    printf("%*s--get\n",WIDTH,"");
    printf("\t get audit status\n\n");
    printf("%*s-h\n",WIDTH,"");
    printf("%*s--help\n",WIDTH,"");
    printf("\t help information\n\n");
    printf("eg:\n");
    printf("%*sauditsectest -s 1\n",WIDTH,"");
    printf("%*sauditsectest -g\n",WIDTH,"");
}

int test_kdk_set_audit_status(char *status){
    int ret = -1;
    if( (0 != strcmp(status,"0")) && (0 != strcmp(status,"1")) ){
```

```
            ret = 1;
        printf("\tret: %d status %s error!\n",ret, status);
        return ret;
    }
    ret = kdk_set_audit_status(atoi(status));
    printf("\tstatus: %s, ret: %d\n",status, ret);
    return ret;
}

int test_kdk_get_audit_status(){
    int ret = -1;
    ret = kdk_get_audit_status();
    printf("\tret: %d\n",ret);
    return ret;
}

int main(int argc, char *argv[])
{
    if (argc == 1)
    {
        usage();
        return -1;
    }

    int opt = 0;
    int option_index = 0;
    int type = 0;
    int ret = -1;

    static struct option long_options[] = {
        {"set", required_argument, 0, 's'},
        {"get", no_argument, 0, 'g'},
        {"help", no_argument, 0, 'h'},
        {0, 0, 0, 0}
    };

    while ((opt = getopt_long(argc, argv, "s:a:gh", long_options,
&option_index)) != -1)
    {
        switch (opt)
        {
        case 's':
                ret = test_kdk_set_audit_status(optarg);
            break;
        case 'g':
                ret = test_kdk_get_audit_status();
            break;
        case 'h':
            usage();
            break;
        default:
            usage();
```

```
        return -1;
      }
    }

    return 0;
}
```

A.6 文件保护接口示例

编译：gcc filesectest.c -lkyfilesec -ldl -lkysdk_module_authorize

```
#include <stdio.h>
#include <getopt.h>

#include <libkyfilesec.h>
#define WIDTH 4

static int ret;

static void usage(void)
{
    printf("Usage: filesectest [ -s | -c ] filepath | [ -h ]\n");
    printf("%*s-s filepath\n",WIDTH,"");
    printf("%*s--set filepath\n",WIDTH,"");
    printf("\t set filepath readonly property\n\n");
    printf("%*s-c filepath\n",WIDTH,"");
    printf("%*s--clear filepath\n",WIDTH,"");
    printf("\t clear filepath readonly property\n\n");
    printf("%*s-h\n",WIDTH,"");
    printf("%*s--help\n",WIDTH,"");
    printf("\t help information\n\n");
    printf("eg:\n");
    printf("%*sfilesectest -s /tmp/test\n",WIDTH,"");
    printf("%*sfilesectest -c /tmp/test\n\n",WIDTH,"");
}

int test_kdk_set_file_readonly(char *filepath){
    //int ret = -1;
    const char *file_path =  filepath;
    ret = kdk_set_file_readonly(file_path);
    printf("\tfilepath: %s, ret: %d\n",filepath, ret);
    return ret;
}

int test_kdk_cancel_file_readonly(char *filepath){
    //int ret = -1;
    const char *file_path = filepath;
    ret =  kdk_cancel_file_readonly(file_path);
    printf("\tfilepath: %s, ret: %d\n",filepath, ret);
    return ret;
```

```
}

int main(int argc, char *argv[])
{
    if (argc == 1)
    {
        usage();
        return -1;
    }

    int opt = 0;
    int option_index = 0;
    //int ret = -1;

    static struct option long_options[] = {
        {"set", required_argument, 0, 's'},
        {"clear", required_argument, 0, 'c'},
        {"help", no_argument, 0, 'h'},
        {0, 0, 0, 0}
    };

    while ((opt = getopt_long(argc, argv, "s:c:h", long_options,
&option_index)) != -1)
    {
        switch (opt)
        {
        case 's':
          ret = test_kdk_set_file_readonly(optarg);
            break;
        case 'c':
          ret = test_kdk_cancel_file_readonly(optarg);
            break;
        case 'h':
            usage();
            break;
        default:
            usage();
            return -1;
        }
    }

      return 0;
}
```

A.7 进程防杀接口示例

编译：gcc processsectest.c -lkyprocesssec -ldl -lkysdk_module_authorize

```
#include <stdio.h>
```

```
#include <getopt.h>

#include <libkyprocesssec.h>
#define WIDTH 4

static int ret;

static void usage(void)
{
    printf("Usage: processsectest [ -s  | -c | -S | -C ] processpath | [ -h ]\n");
    printf("%*s-s processpath\n",WIDTH,"");
    printf("%*s--exe_set processpath\n",WIDTH,"");
    printf("\t set the process executable\n\n");
    printf("%*s-c processpath\n",WIDTH,"");
    printf("%*s--exe_clear processpath\n",WIDTH,"");
    printf("\t clear process executable property\n\n");
    printf("%*s-S processpath\n",WIDTH,"");
    printf("%*s--kill_set processpath\n",WIDTH,"");
    printf("\t set process killed property\n\n");
    printf("%*s-C processpath\n",WIDTH,"");
    printf("%*s--kill_clear processpath\n",WIDTH,"");
    printf("\t clear process killed property\n\n");
    printf("%*s-h\n",WIDTH,"");
    printf("%*s--help\n",WIDTH,"");
    printf("\t help information\n\n");
    printf("eg:\n");
    printf("%*sprocesssectest -s
/opt/qianxin.com/qaxbrowser/qaxbrowser\n",WIDTH,"");
    printf("%*sprocesssectest -c
/opt/qianxin.com/qaxbrowser/qaxbrowser\n",WIDTH,"");
    printf("%*sprocesssectest -S
/opt/qianxin.com/qaxbrowser/qaxbrowser\n",WIDTH,"");
    printf("%*sprocesssectest -C
/opt/qianxin.com/qaxbrowser/qaxbrowser\n\n",WIDTH,"");
}

int test_kdk_process_set_anti_killed(char *processpath){
    //int ret = -1;
    const char *process_path =  processpath;
    ret = kdk_process_set_anti_killed(process_path);
    printf("\tprocesspath: %s, ret: %d\n",processpath, ret);
    return ret;
}

int test_kdk_process_cancel_anti_killed(char *processpath){
    //int ret = -1;
    const char *process_path = processpath;
    ret =  kdk_process_cancel_anti_killed(process_path);
    printf("\tprocesspath: %s, ret: %d\n",processpath, ret);
    return ret;
}
```

```
int main(int argc, char *argv[])
{
    if (argc == 1)
    {
        usage();
        return -1;
    }

    int opt = 0;
    int option_index = 0;

    static struct option long_options[] = {
        {"exe_set", required_argument, 0, 's'},
        {"exe_clear", required_argument, 0, 'c'},
        {"kill_set", required_argument, 0, 'S'},
        {"kill_clear", required_argument, 0, 'C'},
        {"help", no_argument, 0, 'h'},
        {0, 0, 0, 0}
    };

    while ((opt = getopt_long(argc, argv, "s:c:S:C:h", long_options,
&option_index)) != -1)
    {
        switch (opt)
        {
        case 'S':
                ret = test_kdk_process_set_anti_killed(optarg);
            break;
        case 'C':
                ret = test_kdk_process_cancel_anti_killed(optarg);
            break;
        case 'h':
            usage();
            break;
        default:
            usage();
            return -1;
        }
    }

    return 0;
}
```

A.8 openssl sm2算法接口示例

```
//编译：gcc sm2_enc.c -lcrypto -o sm2_enc

//运行：./sm2_enc

//功能：使用openssl进行sm2加密运算
```

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <openssl/ec.h>
#include <openssl/evp.h>

void printECKeyData(const EC_KEY* key)
{
    BIGNUM* gx = BN_new();
    BIGNUM* gy = BN_new();
    printf("private key: %s\n", BN_bn2hex(EC_KEY_get0_private_key(key)));
    const EC_POINT * point  = EC_KEY_get0_public_key(key);
    if(EC_POINT_get_affine_coordinates_GFp(
                EC_KEY_get0_group(key), point, gx, gy , NULL)){
        printf("public key x: %s\n", BN_bn2hex(gx));
        printf("public key y: %s\n", BN_bn2hex(gy));
    }
    BN_free(gx);
    BN_free(gy);
}

int Encrypt(const unsigned char* in_buf, size_t in_buflen, unsigned char*
out_encrypted, size_t* len_encrypted, EC_KEY* pubKey)
{
    printf("----------------------------------------encrypt ----------------------------------------
---\n");
    int ret = -1, i;
    EVP_PKEY_CTX * ectx = NULL;
    EVP_PKEY* pkey = NULL;
    EC_KEY* key_pair = NULL;
    unsigned char* ciphertext = NULL;
    size_t ciphertext_len, plaintext_len;
    pkey = EVP_PKEY_new();

#if 1
    EVP_PKEY_set1_SM2(pkey, pubKey);
#else
    if(!EVP_PKEY_set1_EC_KEY(pkey, pubKey)) {
        printf("Error set public key");
        goto clean_up;
    }
#endif

    /* compute SM2 encryption */
    if ((EVP_PKEY_set_alias_type(pkey, EVP_PKEY_SM2)) != 1)
    {
        printf("EVP_PKEY_set_alias_type failed.");
```

```
        goto clean_up;
    }

    if (!(ectx = EVP_PKEY_CTX_new(pkey, NULL)))
    {
        printf("EVP_PKEY_CTX_new failed.");
        goto clean_up;
    }

    if ((EVP_PKEY_encrypt_init(ectx)) != 1)
    {
        printf("EVP_PKEY_encrypt failed.");
        goto clean_up;
    }

    if ((EVP_PKEY_encrypt(ectx, NULL, &ciphertext_len, in_buf, in_buflen)) != 1)
    {
        printf("EVP_PKEY_set_alias_type failed.");
        goto clean_up;
    }

    if (!(ciphertext = OPENSSL_malloc(ciphertext_len)))
    {
        goto clean_up;
    }

    if ((ret = EVP_PKEY_encrypt(ectx, ciphertext, &ciphertext_len, in_buf,
in_buflen)) != 1)
    {
        printf("EVP_PKEY_encrypt failed.");
        goto clean_up;
    }
    memcpy(out_encrypted, ciphertext, ciphertext_len);
    *len_encrypted = ciphertext_len;
    ret = 0;
clean_up:
    if (pkey)
    {
        EVP_PKEY_free(pkey);
    }

    if (ectx)
    {
        EVP_PKEY_CTX_free(ectx);
    }

    if (ciphertext)
```

```
    {
        free(ciphertext);
    }

    return ret;
}

int Decrypt(const unsigned char* in_buf, size_t in_buflen, unsigned char*
out_plaint, size_t* len_plaint,  EC_KEY* prikey)
{
    printf("----------------------------------------decrypt ----------------------------------------
---\n");
    int ret = -1, i;
    EVP_PKEY_CTX* ectx = NULL;
    EVP_PKEY* pkey = NULL;
    EC_KEY* key_pair = NULL;
    unsigned char * plaintext = NULL;
    size_t ciphertext_len, plaintext_len;

    pkey = EVP_PKEY_new();
#if 1
    EVP_PKEY_set1_SM2(pkey, prikey);
#else
    if(!EVP_PKEY_set1_EC_KEY(pkey, prikey)) {
        printf("Error set public key");
        goto clean_up;
    }
#endif

    if ((EVP_PKEY_set_alias_type(pkey, EVP_PKEY_SM2)) != 1)
    {
        printf("EVP_PKEY_set_alias_type failed.");
        goto clean_up;
    }

    if (!(ectx = EVP_PKEY_CTX_new(pkey, NULL)))
    {
        printf("EVP_PKEY_CTX_new failed.");
        goto clean_up;
    }

    /* compute SM2 decryption */
    if ((EVP_PKEY_decrypt_init(ectx)) != 1)
    {
        printf("EVP_PKEY_decrypt_init failed.");
        goto clean_up;
    }
```

```
    if ((EVP_PKEY_decrypt(ectx, NULL, &plaintext_len, in_buf, in_buflen)) != 1)
    {
        printf("EVP_PKEY_decrypt failed.");
        goto clean_up;
    }

    if (!(plaintext = OPENSSL_malloc(plaintext_len)))
    {
        printf("plaintext is NULL\n");
        goto clean_up;
    }

    if ((ret = EVP_PKEY_decrypt(ectx, plaintext, &plaintext_len, in_buf, in_buflen))
!= 1)
    {
        printf("EVP_PKEY_decrypt failed.");
        goto clean_up;
    }
    memcpy(out_plaint, plaintext, plaintext_len);
    *len_plaint = plaintext_len;
    ret = 0;
clean_up:
    if (pkey)
    {
        EVP_PKEY_free(pkey);
    }
    if (plaintext)
    {
        free(plaintext);
    }
    return ret;
}

int main(int argc ,char* argv[])
{
    EC_KEY* sm2Key = NULL;
    EC_GROUP* group = NULL;
    int count = 1;
    group = EC_GROUP_new_by_curve_name(NID_sm2);
    if (group == NULL) {
        printf("EC_GROUP_new_by_curve_name failed\n");
        return -1;
    }
    sm2Key = EC_KEY_new();
    if (!sm2Key) {
        printf("EC_KEY_new failed!\n");
```

```
        return -1;
    }
    int ret = EC_KEY_set_group(sm2Key, group);
    if (ret != 1) {
        printf("EC_KEY_set_group failed!\n");
        return -1;
    }
    //generate the key
    ret = EC_KEY_generate_key(sm2Key);
    if (ret != 1) {
        printf("EC_KEY_generate_key failed!\n");
        return -1;
    }
    printECKeyData(sm2Key);
    unsigned char* message = "123456";
    const size_t msg_len = 6;
    unsigned char ciphertext[1024] = {0};
    size_t ciphertextLen = 0;
    unsigned char origintext[1024] = {0};
    size_t origintextLen = 0;

    while (count > 0) {
        memset(ciphertext, 0, 1024);
        ciphertextLen = 0;
        Encrypt((unsigned char*)message, msg_len, ciphertext, &ciphertextLen,
sm2Key);
        printf("cipher  text len: %d\n", ciphertextLen);
        for (int i = 0; i < ciphertextLen; ++i) {
            printf("%02x ", ciphertext[i]);
        }
        printf("\n");
        memset(origintext, 0, 1024);
        origintextLen = 0;
        Decrypt(ciphertext, ciphertextLen, origintext, &origintextLen, sm2Key);

        for (int i = 0; i < origintextLen; ++i) {
            printf("%02x ", origintext[i]);
        }
        printf("\n");

        count--;
    }
    printf("decrypt data: %s\n", origintext);
    if (strncasecmp(origintext, message, origintextLen) == 0) {
        printf("success\n");
    }
```

```
    EC_KEY_free(sm2Key);
    EC_GROUP_free(group);
    return 0;
}

//编译：gcc sm2_sig.c -lcrypto -o sm2_sig

//运行： ./sm2_sig
//功能：使用openssl进行sm2签名运算
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <openssl/ec.h>
#include <openssl/evp.h>

int EVP_compute_msg_hash(const EVP_MD *digest, const uint8_t *msg, size_t
msg_len, uint8_t *dgst, size_t *dgst_len)
{
  EVP_MD_CTX *hash = EVP_MD_CTX_new();
  const int md_size = EVP_MD_size(digest);
  int ret = 0;

  if (dgst == NULL) {
    goto done;
  }

  if (md_size < 0) {
    goto done;
  }

  if (hash == NULL) {
    goto done;
  }
  int tmp = 0;

  if (!EVP_DigestInit_ex(hash, digest, NULL) ||
      !EVP_DigestUpdate(hash, msg, msg_len) ||
      !EVP_DigestFinal_ex(hash, dgst, &tmp)) {
    goto done;
  }
  *dgst_len = tmp;
  ret = 1;

done:
  EVP_MD_CTX_free(hash);
  return ret;
```

```
}

int main(int argc ,char* argv[])
{

   EC_KEY* sm2Key = NULL;
   EC_GROUP* group = NULL;
   EVP_PKEY* sm2_pkey = NULL;
   EVP_PKEY_CTX *sm2_pctx = NULL;
   EVP_PKEY_CTX *sm2_vfy_pctx = NULL;

    group = EC_GROUP_new_by_curve_name(NID_sm2);
    if (group == NULL) {
         printf("EC_GROUP_new_by_curve_name failed\n");
         return -1;
    }

    sm2Key = EC_KEY_new();
    if (!sm2Key) {
         printf("EC_KEY_new failed!\n");
         return -1;
    }
    int ret = EC_KEY_set_group(sm2Key, group);
    if (ret != 1) {
         printf("EC_KEY_set_group failed!\n");
         return -1;
    }
    /*generate the key*/
    ret = EC_KEY_generate_key(sm2Key);
    if (ret != 1) {
         printf("EC_KEY_generate_key failed!\n");
         return -1;
    }

    const char* message = "the encryption data";
    const size_t msg_len = strlen(message);

   sm2_pkey = EVP_PKEY_new();
   if (sm2_pkey == NULL) {
    return -1;
   }

   if (!EVP_PKEY_set1_SM2(sm2_pkey, sm2Key)) {
    return -1;
   }

   sm2_pctx = EVP_PKEY_CTX_new(sm2_pkey, NULL);
```

```
    if (sm2_pctx == NULL) {
        return -1;
      }

    unsigned char hash[32] = {0};
    size_t hashLen = 0;
    if (!EVP_compute_msg_hash(EVP_sm3(), message, msg_len, hash, &hashLen))
{
        printf("EVP_compute_msg_hash failed\n");
        return -1;
    }

     if (EVP_PKEY_sign_init(sm2_pctx) <= 0) {
          printf("EVP_PKEY_sign_init failed\n");
          return -1;
     }

     size_t signValueLen = 0;

     if (EVP_PKEY_sign(sm2_pctx, NULL , &signValueLen, hash, hashLen) <= 0) {
          printf("EVP_PKEY_sign first failed\n");
          return -1;
     }
     unsigned char* signValue = malloc(signValueLen+1);
     memset(signValue, 0, signValueLen+1);
     if (!(ret = EVP_PKEY_sign(sm2_pctx, signValue, &signValueLen, hash,
hashLen)) ) {
          printf("EVP_PKEY_sign sec failed\n");
          return -1;
     }

    EVP_PKEY* ver_pkey = EVP_PKEY_new();
    if (ver_pkey == NULL) {
        return -1;
    }

    if (!EVP_PKEY_set1_SM2(ver_pkey, sm2Key)) {
        return -1;
    }

    sm2_vfy_pctx = EVP_PKEY_CTX_new(ver_pkey, NULL);
    if (sm2_vfy_pctx == NULL) {
        return -1;
      }

     if (EVP_PKEY_verify_init(sm2_vfy_pctx) <= 0) {
          printf("EVP_PKEY_verify_init failed\n");
```

```
        return -1;
    }

    if (EVP_PKEY_verify(sm2_vfy_pctx, signValue, signValueLen, hash, hashLen)
!= 1) {
        printf("EVP_PKEY_verify failed\n");
    }
    printf("verify ok\n");

    EVP_PKEY_free(sm2_pkey);
    EVP_PKEY_CTX_free(sm2_pctx);
    EVP_PKEY_CTX_free(sm2_vfy_pctx);
    free(signValue);

    return 0;
}
```

A.9 openssl sm3算法接口示例

```
//编译：gcc sm3.c -lcrypto -o sm3
//运行：./sm3
//功能：使用openssl进行sm3运算
#include <stdio.h>
#include <openssl/evp.h>

int main()
{
    int len, dataLen = 3;
    char source[3]={'a','b','c'};
    const EVP_MD *sm3_md;
    EVP_MD_CTX *sm3_ctx;
    unsigned char hash_value[64];

    sm3_md = EVP_sm3();
    sm3_ctx = EVP_MD_CTX_new();

    EVP_DigestInit_ex(sm3_ctx, sm3_md, NULL);
    EVP_DigestUpdate(sm3_ctx, source, dataLen);
    EVP_DigestFinal_ex(sm3_ctx, hash_value, &len);

    printf("hash_value:");
    for(int i = 0; i < len; i++)
        printf("%02x ",hash_value[i]);
```

```
    printf("\n");

    EVP_MD_CTX_free(sm3_ctx);
    return 0;
}
```

A.10 openssl sm4算法接口示例

```
//编译：gcc -g sm4.c -lcrypto -o sm4
//运行：./sm4
//功能：sm4加解密测试
#include <stdio.h>
#include <assert.h>
#include <openssl/evp.h>

void printHex(char *name, unsigned char *c, int n)
{
    int i;
    printf("%s: ",name);
    for (i = 0; i < n; i++)
    {
        printf("%02X", c[i]);
    }
    printf("\n");
}

int main()
{
    int outdata_len = 0;
    int dataLen = 16;
    int plain_dataLen = 17;
    int enc_dataLen = 16;
    EVP_CIPHER_CTX *ctx;
    const EVP_CIPHER *mode;
    int sm4enStrLen,sm4deStrLen;
    unsigned char *sm4_en = malloc(2*dataLen);
    unsigned char *sm4_de = malloc(2*dataLen);
    unsigned char
key[16]={0x15,0x67,0x28,0xe1,0x5f,0x9a,0xfc,0x01,0xd4,0xb6,0x1
b,0x4e,0x44,0x5d,0xbb,0x26};
    unsigned char
plain[17]={0x15,0x67,0x28,0xe1,0x5f,0x9a,0xfc,0x01,0xd4,0xb6,0x
1b,0x4e,0x44,0x5d,0xbb,0x26,0x1};
unsigned char
iv[16]={0x15,0x67,0x28,0xe1,0x5f,0x9a,0xfc,0x01,0xd4,0xb6,0x1b,
0x4e,0x44,0x5d,0xbb,0x26};
```

```
  ctx = EVP_CIPHER_CTX_new();
   assert(ctx != NULL && "CIPHER_CTX_new");

   mode = EVP_sm4_ecb();
   assert(mode != NULL);

   /* Encrypt */
   EVP_EncryptInit_ex(ctx, mode, NULL, key, NULL);
   if( 0 == plain_dataLen % 16 )
   {
      EVP_CIPHER_CTX_set_padding( ctx, 0 );
   }else
       EVP_CIPHER_CTX_set_padding( ctx, 1 );
   if (!EVP_EncryptUpdate(ctx, sm4_en, &sm4enStrLen, plain,
plain_dataLen))
       return -1;
   outdata_len = sm4enStrLen;
   sm4enStrLen = 0;
   if (!EVP_EncryptFinal_ex(ctx, sm4_en+outdata_len,
&sm4enStrLen))
       return -1;
   sm4enStrLen += outdata_len;
   printHex("Encrypt", sm4_en, sm4enStrLen);


   /* Decrypt */
   EVP_DecryptInit_ex(ctx, mode, NULL, key, NULL);
   if( 0 == sm4enStrLen % 16 )
   {
      EVP_CIPHER_CTX_set_padding( ctx, 0 );
   }else
       EVP_CIPHER_CTX_set_padding( ctx, 1 );
   if (!EVP_DecryptUpdate(ctx, sm4_de, &sm4deStrLen, sm4_en,
sm4enStrLen)) {
       return -1;
   }
   outdata_len = sm4deStrLen;
   sm4deStrLen = 0;
   if (!EVP_DecryptFinal_ex(ctx, sm4_de + outdata_len,
&sm4deStrLen)) {
       return -1;
   }
   sm4deStrLen += outdata_len;
   printHex("Decrypt", sm4_de, sm4deStrLen);

   EVP_CIPHER_CTX_free(ctx);
```

```
    return 0;
}
```

A.11 usbguard接口示例

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <kysdk/kysdk-security/libkydevicesec.hpp>

using namespace usbguard;
using namespace std;

int main(int argc, char* argv[])
{
     int ret;
    int id;

     cout << "usbguard_appendRule:"  << endl;
     std::string rule_spec = "allow id 1234:abcd name \"name1\" serial
\"sn1\"  with-interface f3:ab:cd";

     id = kdk_usbguard_appendRule(rule_spec);
     cout << "append rule id:" << id <<endl;


     cout << "usbguard_removedRule:"  << endl;
     ret = kdk_usbguard_removeRule(id);
     if(ret == 0){
           cout << "removed rule id: " << id << endl;
     }


     cout << "usbguard_appendRule_specific:"  << endl;
     std::string vid = string("12f4");
     std::string pid = string("434a");

     id = kdk_usbguard_appendRule_specific(vid, pid, name, serial,  target);
     cout << "append rule id: " << id <<endl;


     cout << "usbguard_removeRule_specific:"  << endl;
     ret = kdk_usbguard_removeRule_specific(vid, pid, name, serial);
     if(ret == 0){
           cout << "removed rule vid:pid:  " << vid << ":" << pid << endl;
     }


     cout << "usbguard_getRules:"  << endl;
     std::string interface1;
     std::vector<Rule> rules;
     ret =  kdk_usbguard_getRules(rules);
     print_rules_list(rules);

     cout << "usbguard_getRule_specific:"  << endl;
     int decision;
     decision = kdk_usbguard_getRule_specific(vid, pid, name, serial);
     cout << "vid:pid:name:serial" << ":" << endl;
     cout << vid << ":" << pid << ":" << name << ":" << serial << endl;
```

```
        cout << "--------" << " rule taget:" << decision << endl;


        cout << "usbguard_getDevices:"  << endl;
        std::vector<Rule> devices;
        ret = kdk_usbguard_getDevices(devices);
        print_devices_list(devices);

        return 0;
}

int print_rules_list(std::vector<Rule>& rules)
{
        if ( NULL == rules){
            return -1;
        }

        for (auto rule : rules) {
              cout << rule.getRuleID() <<": "<<  rule.toString() << endl;
              cout << "---- ";
              cout << rule.getDeviceID().toString() << " ";
              if(!rule.attributeSerial().empty()){
                      cout << rule.getSerial() << " " ;
              }
              if(!rule.attributeName().empty()){
                      cout << rule.getName() << " ";
              }
              interface1="";
              for (auto interface : rule.attributeWithInterface().values()) {
                      if(string::npos == nterface1.find(interface.toRuleString())){
                            interface1.append(interface.toRuleString());
                      }
              }
              std::cout << interface1 << " ";
              std::cout << endl;
        }
        return 0;
}

int print_devices_list(std::vector<Rule>& devices)
{
        if ( NULL == devices){
            return -1;
        }

        for(auto device : devices){
              std::cout << device.getRuleID() << ":";
              std::cout << Rule::targetToString(device.getTarget()) << " ";
              std::cout << device.getDeviceID().toString() << " ";
              std::cout << device.getName() << " ";
              std::cout << device.getSerial() << " ";
              std::cout << device.getViaPort() << " ";
              interface1="";
              for (auto interface : device.attributeWithInterface().values()) {
                    if(string::npos == interface1.find(interface.toRuleString())){
                          interface1.append(interface.toRuleString());
                    }
              }
              std::cout << interface1 << " ";
              std::cout << endl;
        }
        return 0;
}
```

A.12 更新系统下国家电子CA根证书

```
//编译：g++ kdk_gdca_update.c -lgdca -o kdk_gdca_update

//运行：./kdk_gdca_update
//功能：更新国家电子CA根证书

#include <stdio.h>
#include <stdio.h>

#include <stdlib.h>

#include <gdca/libkygdca.h>


const char ca_update_status[3][64] = {
    "已是最新的",
    "已更新",
    "不属于国家电子CA根证书,未做更新"
};
void show_update_status(kdk_gdca_updata_status *updata_status, int
change_num)
{
        for(int i = 0; i < change_num; i++)
        {
            printf("%s%s\n",updata_status[i].file_name,
      ca_update_status[updata_status[i].updata_status]);
        }
}
int main()
{
    int ret;
    int change_cert_num = 0;
    const char *ca_filename = NULL;
    kdk_gdca_updata_status *cert_updata_status = NULL;
    ret = kdk_gdca_update(ca_filename, &cert_updata_status,
&change_cert_num);
    if( ret != GDCA_OK)
    {
        if(ret == GDCA_POST_EASY_PRTFORM_ERR)
            printf("证书更新失败,请检查网络状态\n");
        else
```

```
            printf("gdca_certs update command fail, errorcode is 0x%08x\n",
ret);
        free((char *)ca_filename);
        return ret;
    }
    show_update_status(cert_updata_status, change_cert_num);
    return 0;
}
```

A.13 导入国家电子CA根证书

```
//编译：g++ kdk_gdca_import.c -lgdca -o kdk_gdca_import
//运行：./kdk_gdca_import 公务人员应用根证书.cer
//功能：导入公务人员应用根证书
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <gdca/libkygdca.h>
void show_import_status(int status,char *source_path)
{
    char *ca_name = strrchr(source_path, '/'); // 查找最后一个'/'的位置
    if(ca_name == NULL)
    {
        ca_name = source_path;
    }else{
        ca_name++;  //跳过'/'字符
    }

    switch(status)
    {
        case GDCA_OK:
            printf("证书已导入至%s\n",CERT_DEFAULT_INSTALL_PATH);
            break;
        case CERT_OVERLAY_IMPORT:
```

```
            printf("%s已覆盖
     %s%s\n",source_path,CERT_DEFAULT_INSTALL_PATH,ca_name);
            break;
        case GDCA_CERT_OPEN_ERR:
        case GDCA_CERT_LOAD_ERR:
        case GDCA_CERT_VERIFY_ERR:
        case GDCA_CERT_OWNER_ERR:
            printf("证书导入失败，导入证书非国家电子CA根证书\n");
            break;
        case GDCA_CERT_VALIDITY_DATE_ERR:
            printf("证书导入失败，导入证书不在有效使用期限内\n");
            break;
        case GDCA_NOT_FILE_ERR:
            printf("导入证书失败,导入证书文件不存在\n");
            break;
        default:
            printf("gdca_certs import command fail, errorcode is
0x%08x\n", status);
            break;
    }
}
int main(int argc,char *argv[])
{
    int ret = 0;
    if(argc !=2 )
    {
        printf("param error\n");
        return 0;
    }
    ret = kdk_gdca_import(argv[1]);
    show_import_status(ret,argv[1]);
    return 0;
}
```

A.14 查询国家电子CA根证书安装情况

```
//编译：g++ kdk_gdca_exist.c -lgdca -o kdk_gdca_exist

//运行：./kdk_gdca_exist

//功能： 查询国家电子CA根证书安装情况

#include <stdio.h>
#include <gdca/libkygdca.h>

int main()

{

    int ret;

    int number = 0;

    kdk_gdca_cert *gdca_info = NULL;

    ret = kdk_gdca_exist(&gdca_info,&number);

if(GDCA_OK != ret)

    {

        printf("gdca_certs exist  fail, errorcode is 0x%08x\n", ret);

        return ret;

    }

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

        printf("%s已安装\n",gdca_info[i].file_name);

    return 0;

}
```

A.15 删除国家电子CA根证书

```
//编译：g++ kdk_gdca_delete.c -lgdca -o kdk_gdca_delete

//运行：./kdk_gdca_delete 公务人员应用根证书.cer

//功能： 删除/etc/pki/GDCA/公务人员应用根证书.cer证书文件

#include <stdio.h>
#include <gdca/libkygdca.h>

int main(int argc,char *argv[])

{

    int ret = 0;

    if(argc !=2 )

    {
```

```
        printf("param error\n");
        return 0;
    }
    ret = kdk_gdca_delete(argv[1]);
    if(ret == GDCA_DELETE_FILE_ERR)
        printf("指定删除的%s%s证书文件不存在
\n",CERT_DEFAULT_INSTALL_PATH,argv[1]);
    else
        printf("%s%s证书文件已删除\n",CERT_DEFAULT_INSTALL_PATH,argv[1]);
    return 0;
}
```

A.16 海光sm2算法接口示例

```
//编译：gcc sm2_enc.c -lcrypto -o _sm2_enc
//运行： ./_sm2_enc
//功能：使用加速引擎进行sm2加密运算
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#include <openssl/ec.h>
#include <openssl/evp.h>
#include <openssl/engine.h>

static char *engine_name = "hct";
ENGINE *e = NULL;

void printECKeyData(const EC_KEY *key)
{
   BIGNUM *gx = BN_new();
   BIGNUM *gy = BN_new();
```

```
    if (!gx || !gy)
    {
        printf("BN_new failed on gx or gy\n");
        return;
    }

    printf("private key: %s\n",
        BN_bn2hex(EC_KEY_get0_private_key(key)));

    const EC_POINT *point = EC_KEY_get0_public_key(key);
    if (!point)
    {
        printf("EC_KEY_get0_public_key failed\n");
        return;
    }

    if (EC_POINT_get_affine_coordinates_GFp(
            EC_KEY_get0_group(key), point, gx, gy, NULL))
    {
        printf("public key x: %s\n", BN_bn2hex(gx));
        printf("public key y: %s\n", BN_bn2hex(gy));
    }
    BN_free(gx);
    BN_free(gy);
}

int Encrypt(const unsigned char *in_buf, size_t in_buflen,
            unsigned char *out_encrypted, size_t *len_encrypted,
            EC_KEY *pubKey)
{
    printf("----------------encrypt -------------------\n");
```

```
int ret = -1, i;
EVP_PKEY_CTX *ectx = NULL;
EVP_PKEY *pkey = NULL;
EC_KEY *key_pair = NULL;
unsigned char *ciphertext = NULL;
size_t ciphertext_len, plaintext_len;

pkey = EVP_PKEY_new();
if (pkey == NULL)
{
    printf("EVP_PKEY_new pkey failed!\n");
    goto clean_up;
}

if (!EVP_PKEY_set1_EC_KEY(pkey, pubKey))
{
    printf("Error set public key");
    goto clean_up;
}
/* compute SM2 encryption */
if ((EVP_PKEY_set_alias_type(pkey, EVP_PKEY_SM2)) != 1)
{
    printf("EVP_PKEY_set_alias_type failed.");
    goto clean_up;
}

if (!(ectx = EVP_PKEY_CTX_new(pkey, e)))
{
    printf("EVP_PKEY_CTX_new failed.");
    goto clean_up;
}
```

```
    if ((EVP_PKEY_encrypt_init(ectx)) != 1)
    {
        printf("EVP_PKEY_encrypt failed.");
        goto clean_up;
    }

    if ((EVP_PKEY_encrypt(ectx, NULL, &ciphertext_len,
            in_buf, in_buflen)) != 1)
    {
        printf("EVP_PKEY_set_alias_type failed.");
        goto clean_up;
    }

    if (!(ciphertext = OPENSSL_malloc(ciphertext_len)))
    {
        goto clean_up;
    }

    if ((ret = EVP_PKEY_encrypt(ectx, ciphertext,
            &ciphertext_len, in_buf, in_buflen)) != 1)
    {
        printf("EVP_PKEY_encrypt failed.");
        goto clean_up;
    }

    memcpy(out_encrypted, ciphertext, ciphertext_len);
    *len_encrypted = ciphertext_len;
    ret = 0;

clean_up:
```

```
    if (pkey)
    {
        EVP_PKEY_free(pkey);
    }

    if (ectx)
    {
        EVP_PKEY_CTX_free(ectx);
    }

    if (ciphertext)
    {
        free(ciphertext);
    }

    return ret;
}

int Decrypt(const unsigned char *in_buf, size_t in_buflen,
            unsigned char *out_plaint, size_t *len_plaint,
            EC_KEY *prikey)
{
    printf("----------------decrypt -------------------\n");
    int ret = -1, i;
    EVP_PKEY_CTX *ectx = NULL;
    EVP_PKEY *pkey = NULL;
    EC_KEY *key_pair = NULL;
    unsigned char *plaintext = NULL;
    size_t ciphertext_len, plaintext_len;

    pkey = EVP_PKEY_new();
```

```
if (pkey == NULL)
{
    printf("EVP_PKEY_new pkey failed!\n");
    goto clean_up;
}

if (!EVP_PKEY_set1_EC_KEY(pkey, prikey))
{
    printf("Error set public key");
    goto clean_up;
}

if ((EVP_PKEY_set_alias_type(pkey, EVP_PKEY_SM2)) != 1)
{
    printf("EVP_PKEY_set_alias_type failed.");
    goto clean_up;
}

if (!(ectx = EVP_PKEY_CTX_new(pkey, e)))
{
    printf("EVP_PKEY_CTX_new failed.");
    goto clean_up;
}

/* compute SM2 decryption */
if ((EVP_PKEY_decrypt_init(ectx)) != 1)
{
    printf("EVP_PKEY_decrypt_init failed.");
    goto clean_up;
}
```

```
    if ((EVP_PKEY_decrypt(ectx, NULL, &plaintext_len, in_buf,
            in_buflen)) != 1)
    {
        printf("EVP_PKEY_decrypt failed.");
        goto clean_up;
    }

    if (!(plaintext = OPENSSL_malloc(plaintext_len)))
    {
        printf("plaintext is NULL\n");
        goto clean_up;
    }

    if ((ret = EVP_PKEY_decrypt(ectx, plaintext, &plaintext_len,
            in_buf, in_buflen)) != 1)
    {
        printf("EVP_PKEY_decrypt failed.");
        goto clean_up;
    }

    memcpy(out_plaint, plaintext, plaintext_len);
    *len_plaint = plaintext_len;
    ret = 0;

clean_up:
    if (pkey)
    {
        EVP_PKEY_free(pkey);
    }
    if (plaintext)
    {
```

```
        free(plaintext);
    }
    return ret;
}

int main(int argc, char *argv[])
{
    EC_KEY *sm2Key = NULL;
    EC_GROUP *group = NULL;
    int count = 1;
    unsigned char *message = "123456";
    const size_t msg_len = 6;
    unsigned char ciphertext[1024] = {0};
    size_t ciphertextLen = 0;
    unsigned char origintext[1024] = {0};
    size_t origintextLen = 0;

    group = EC_GROUP_new_by_curve_name(NID_sm2);
    if (group == NULL)
    {
        printf("EC_GROUP_new_by_curve_name failed\n");
        return -1;
    }

    sm2Key = EC_KEY_new();
    if (!sm2Key)
    {
        printf("EC_KEY_new failed!\n");
        return -1;
    }
```

```
int ret = EC_KEY_set_group(sm2Key, group);
if (ret != 1)
{
    printf("EC_KEY_set_group failed!\n");
    return -1;
}

/*generate the key*/
ret = EC_KEY_generate_key(sm2Key);
if (ret != 1)
{
    printf("EC_KEY_generate_key failed!\n");
    return -1;
}
printECKeyData(sm2Key);

ENGINE_load_builtin_engines();
if ((e = ENGINE_by_id(engine_name)) == NULL)
{
    printf("ENGINE_by_id error\n");
    return -1;
}

while (count > 0)
{
    memset(ciphertext, 0, 1024);
    ciphertextLen = 0;

    Encrypt((unsigned char *)message, msg_len, ciphertext,
            &ciphertextLen, sm2Key);
```

```
        printf("cipher  text len: %ld\n", ciphertextLen);
        for (int i = 0; i < ciphertextLen; ++i)
        {
            printf("%02x ", ciphertext[i]);
        }
        printf("\n");

        memset(origintext, 0, 1024);
        origintextLen = 0;

        Decrypt(ciphertext, ciphertextLen, origintext,
                &origintextLen, sm2Key);

        for (int i = 0; i < origintextLen; ++i)
        {
            printf("%02x ", origintext[i]);
        }
        printf("\n");

        count--;
    }

    printf("decrypt data: %s\n", origintext);
    if (strncasecmp(origintext, message, origintextLen) == 0)
    {
        printf("success\n");
    }

    EC_KEY_free(sm2Key);
    EC_GROUP_free(group);
    return 0;
```

```
}
```

```
//编译：gcc sm2_sig.c -lcrypto -o _sm2_sig
//运行： ./_sm2_sig
//功能：使用加速引擎进行sm2签名运算
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>

#include <openssl/ec.h>
#include <openssl/evp.h>
#include <openssl/engine.h>

static char *engine_name = "hct";
static ENGINE *e = NULL;

int EVP_compute_msg_hash(const EVP_MD *digest,
                const uint8_t *msg, size_t msg_len,
                uint8_t *dgst, size_t *dgst_len)
{
    EVP_MD_CTX *hash = EVP_MD_CTX_new();
    const int md_size = EVP_MD_size(digest);
    int ret = 0;
    int tmp = 0;

    if (dgst == NULL)
    {
        goto done;
    }
```

```
    if (md_size < 0)
    {
        goto done;
    }

    if (hash == NULL)
    {
        goto done;
    }

    if (!EVP_DigestInit_ex(hash, digest, e) ||
        !EVP_DigestUpdate(hash, msg, msg_len) ||
        !EVP_DigestFinal_ex(hash, dgst, &tmp))
    {
        goto done;
    }

    *dgst_len = tmp;
    ret = 1;

done:
    EVP_MD_CTX_free(hash);
    return ret;
}

int main(int argc, char *argv[])
{

    EC_KEY *sm2Key = NULL;
    EC_GROUP *group = NULL;
    EVP_PKEY *sm2_pkey = NULL;
```

```
    EVP_PKEY_CTX *sm2_pctx = NULL;
    EVP_PKEY_CTX *sm2_vfy_pctx = NULL;
    EVP_PKEY *ver_pkey = NULL;
    const char *message = "the encryption data";
    const size_t msg_len = strlen(message);
    unsigned char hash[32] = {0};
    size_t hashLen = 0;
    size_t signValueLen = 0;
    unsigned char *signValue = NULL;
    int ret = -1;

    ENGINE_load_builtin_engines();
    if ((e = ENGINE_by_id(engine_name)) == NULL)
    {
        printf("ENGINE_by_id error\n");
        return -1;
    }

    group = EC_GROUP_new_by_curve_name(NID_sm2);
    if (group == NULL)
    {
        printf("EC_GROUP_new_by_curve_name failed\n");
        return -1;
    }

    sm2Key = EC_KEY_new();
    if (!sm2Key)
    {
        printf("EC_KEY_new sm2Key failed!\n");
        return -1;
    }
```

```
ret = EC_KEY_set_group(sm2Key, group);
if (ret != 1)
{
    printf("EC_KEY_set_group failed!\n");
    return -1;
}

/*generate the key*/
ret = EC_KEY_generate_key(sm2Key);
if (ret != 1)
{
    printf("EC_KEY_generate_key failed!\n");
    return -1;
}

sm2_pkey = EVP_PKEY_new();
if (sm2_pkey == NULL)
{
    printf("EVP_PKEY_new sm2_pkey failed!\n");
    return -1;
}

if (!EVP_PKEY_set1_EC_KEY(sm2_pkey, sm2Key))
{
    printf("Error set private key\n");
    return -1;
}

if ((EVP_PKEY_set_alias_type(sm2_pkey, EVP_PKEY_SM2)) != 1)
{
```

```
        printf("EVP_PKEY_set_alias_type failed.");
        return -1;
    }

    sm2_pctx = EVP_PKEY_CTX_new(sm2_pkey, e);
    if (sm2_pctx == NULL)
    {
        printf("EVP_PKEY_CTX_new sm2_pctx failed!\n");
        return -1;
    }

    if (!EVP_compute_msg_hash(EVP_sm3(), message,
            msg_len, hash, &hashLen))
    {
        printf("EVP_compute_msg_hash failed\n");
        return -1;
    }

    if (EVP_PKEY_sign_init(sm2_pctx) <= 0)
    {
        printf("EVP_PKEY_sign_init failed\n");
        return -1;
    }

    if (EVP_PKEY_sign(sm2_pctx, NULL, &signValueLen,
                    hash, hashLen) <= 0)
    {
        printf("EVP_PKEY_sign first failed\n");
        return -1;
    }
```

```
    signValue = malloc(signValueLen + 1);
    if (!signValue)
    {
        printf("malloc signValue fialed\n");
        return -1;
    }
    memset(signValue, 0, signValueLen + 1);

    if (!(ret = EVP_PKEY_sign(sm2_pctx, signValue,
                              &signValueLen, hash, hashLen)))
    {
        printf("EVP_PKEY_sign sec failed\n");
        return -1;
    }

    ver_pkey = EVP_PKEY_new();
    if (ver_pkey == NULL)
    {
        printf("EVP_PKEY_new ver_pkey fialed\n");
        return -1;
    }

    if (!EVP_PKEY_set1_EC_KEY(ver_pkey, sm2Key))
    {
        printf("Error set public key\n");
        return -1;
    }

    if ((EVP_PKEY_set_alias_type(ver_pkey, EVP_PKEY_SM2)) != 1)
    {
        printf("EVP_PKEY_set_alias_type failed.");
```

```
        return -1;
    }

    sm2_vfy_pctx = EVP_PKEY_CTX_new(ver_pkey, e);
    if (sm2_vfy_pctx == NULL)
    {
        printf("EVP_PKEY_CTX_new fialed\n");
        return -1;
    }

    if (EVP_PKEY_verify_init(sm2_vfy_pctx) <= 0)
    {
        printf("EVP_PKEY_verify_init failed\n");
        return -1;
    }

    if (EVP_PKEY_verify(sm2_vfy_pctx, signValue,
                        signValueLen, hash, hashLen) != 1)
    {
        printf("EVP_PKEY_verify failed\n");
    }
    printf("verify ok\n");

    EVP_PKEY_free(sm2_pkey);
    EVP_PKEY_CTX_free(sm2_pctx);
    EVP_PKEY_CTX_free(sm2_vfy_pctx);
    free(signValue);

    return 0;
}
```

A.17 海光sm3算法接口示例

```
//编译：gcc sm3.c -lcrypto -o _sm3
//运行： ./_sm3
//功能：使用加速引擎进行sm3运算
#include <stdio.h>
#include <openssl/evp.h>
#include <openssl/engine.h>

int main()
{
    ENGINE *e = NULL;
    int len, dataLen = 3;
    char source[3] = {'a', 'b', 'c'};
    const EVP_MD *sm3_md;
    EVP_MD_CTX *sm3_ctx;
    unsigned char hash_value[64];

    ENGINE_load_builtin_engines();
    e = ENGINE_by_id("hct");
    if (e == NULL)
    {
        printf("load hct engine failed\n");
        return -1;
    }

    sm3_md = EVP_sm3();
    if (!sm3_md)
    {
```

```
        printf("new sm3_md failed\n");
        return -1;
    }

    sm3_ctx = EVP_MD_CTX_new();
    if (!sm3_ctx)
    {
        printf("EVP_MD_CTX_new sm3_ctx failed\n");
        return -1;
    }

    EVP_DigestInit_ex(sm3_ctx, sm3_md, e);
    EVP_DigestUpdate(sm3_ctx, source, dataLen);
    EVP_DigestFinal_ex(sm3_ctx, hash_value, &len);

    for (int i = 0; i < len; i++)
        printf("%02x ", hash_value[i]);
    printf("\n");

    EVP_MD_CTX_free(sm3_ctx);
    return 0;
}
```

A.18 海光sm4算法接口示例

```
//编译：gcc -g sm4.c -lcrypto -o _sm4
//运行： ./_sm4
//功能： sm4加解密测试
#include <stdio.h>
#include <assert.h>
```

```
#include <openssl/evp.h>
#include <openssl/engine.h>

void printHex(char *name, unsigned char *c, int n)
{
   int i;
   printf("%s: ", name);
   for (i = 0; i < n; i++)
   {
      printf("%02X", c[i]);
   }
   printf("\n");
}

int main()
{
   ENGINE *e = NULL;
   int outdata_len = 0;
   int dataLen = 16;
   int plain_dataLen = 17;
   int enc_dataLen = 16;
   EVP_CIPHER_CTX *ctx;
   const EVP_CIPHER *mode;
   int sm4enStrLen, sm4deStrLen;
   unsigned char *sm4_en = malloc(2 * dataLen);
   unsigned char *sm4_de = malloc(2 * dataLen);

   unsigned char key[16] = {0x15, 0x67, 0x28, 0xe1,
                            0x5f, 0x9a, 0xfc, 0x01,
                            0xd4, 0xb6, 0x1b, 0x4e,
                            0x44, 0x5d, 0xbb, 0x26};
```

```
unsigned char plain[17] = {0x15, 0x67, 0x28, 0xe1,
                    0x5f, 0x9a, 0xfc, 0x01,
                    0xd4, 0xb6, 0x1b, 0x4e,
                    0x44, 0x5d, 0xbb, 0x26, 0x1};

unsigned char iv[16] = {0x15, 0x67, 0x28, 0xe1,
                  0x5f, 0x9a, 0xfc, 0x01,
                  0xd4, 0xb6, 0x1b, 0x4e,
                  0x44, 0x5d, 0xbb, 0x26};

ENGINE_load_builtin_engines();
e = ENGINE_by_id("hct");
if (e == NULL)
{
   printf("load hct engine failed\n");
   return -1;
}

ctx = EVP_CIPHER_CTX_new();
if (!ctx)
{
   printf("EVP_CIPHER_CTX_new failed\n");
   return -1;
}

mode = EVP_sm4_ecb();
if (!mode)
{
   printf("EVP_sm4_ecb failed\n");
   return -1;
```

```
}

/* Encrypt */
EVP_EncryptInit_ex(ctx, mode, NULL, key, NULL);
if (0 == plain_dataLen % 16)
{
   EVP_CIPHER_CTX_set_padding(ctx, 0);
}
else
{
   EVP_CIPHER_CTX_set_padding(ctx, 1);
}

if (!EVP_EncryptUpdate(ctx, sm4_en, &sm4enStrLen,
                plain, plain_dataLen))
   return -1;

outdata_len = sm4enStrLen;
sm4enStrLen = 0;
if (!EVP_EncryptFinal_ex(ctx, sm4_en + outdata_len,
                 &sm4enStrLen))
   return -1;

sm4enStrLen += outdata_len;
printHex("Encrypt", sm4_en, sm4enStrLen);

/* Decrypt */
EVP_DecryptInit_ex(ctx, mode, NULL, key, NULL);
if (0 == sm4enStrLen % 16)
{
   EVP_CIPHER_CTX_set_padding(ctx, 0);
```

```
    }
    else
    {
        EVP_CIPHER_CTX_set_padding(ctx, 1);
    }

    if (!EVP_DecryptUpdate(ctx, sm4_de, &sm4deStrLen,
                    sm4_en, sm4enStrLen))
    {
        return -1;
    }

    outdata_len = sm4deStrLen;
    sm4deStrLen = 0;
    if (!EVP_DecryptFinal_ex(ctx, sm4_de + outdata_len,
                     &sm4deStrLen))
    {
        return -1;
    }

    sm4deStrLen += outdata_len;
    printHex("Decrypt", sm4_de, sm4deStrLen);

    EVP_CIPHER_CTX_free(ctx);
    return 0;
}
```

A.19 海光sdf_sm2算法接口示例

```
// 创建SM2签名密钥：sudo hag tkm intl_key_gen -idx 1 -type sm2sign -password 12345678
// 编译：gcc sdf_sm2_sign.c -lhysdf -lhct -o sdf_sm2_sign
```

```
// 运行：sudo ./sdf_sm2_sign
// 销毁SM2签名密钥：sudo hag tkm key_destroy -idx 1 -type sm2sign -password 12345678
// 功能：使用海光SDF接口计算SM2签名



#include <hysdf.h>
#include <stdio.h>
#include <string.h>

#define ERR_COND(cond, err, label, fmt, ...) \
    if (cond) {                          \
        printf("ERR: %s:%u: " fmt, __func__, __LINE__, ##__VA_ARGS__);   \
        ret = err;                       \
        goto label;                      \
    }

#define hexdump(data, len) {         \
    fprintf(stdout, "%s in %p, size %d:", #data, data, (len));     \
    _hexdump_(data, len);   \
}

__attribute__ ((unused)) static void _hexdump_(uint8_t *data, uint32_t len)
{
    uint32_t i = 0;
    for (i = 0; i < len; ++i) {
        if (i % 32 == 0) fprintf(stdout, "\n");
        fprintf(stdout, "%02X", data[i]);
    }
    fprintf(stdout, "\n\n");
}

int main ()
{
    int ret = 0;
    void *hDeviceHandle = NULL;
    void *hSessionHandle = NULL;
    ECCSignature sign;
    ECCrefPublicKey puckey;
    unsigned int zlen, idx = 1;
    unsigned char hash_z[32], data[128];
    char *password = "12345678";

    ret = SDF_OpenDevice(&hDeviceHandle);
```

```
ERR_COND(ret || hDeviceHandle == NULL, ret, out,
        "SDF_OpenDevice failed 0x%x\n", ret);
ret = SDF_OpenSession(hDeviceHandle, &hSessionHandle);
ERR_COND(ret || hSessionHandle == NULL, ret, out,
        "SDF_OpenSession failed 0x%x\n", ret);
ret = SDF_ExportSignPublicKey_ECC(hSessionHandle, idx, &puckey);
ERR_COND(ret, ret, out,
        "SDF_ExportSignPublicKey_ECC failed 0x%x\n", ret);
ret = SDF_GetPrivateKeyAccessRight(hSessionHandle, idx,
            (unsigned char*)password, strlen(password));
ERR_COND(ret, ret, out,
            "SDF_GetPrivateKeyAccessRight failed 0x%x\n", ret);

ret = SDF_GenerateRandom(hSessionHandle, 128, data);
ERR_COND(ret, -1, out, "SDF_GenerateRandom failed 0x%x\n", ret);
ret = SDF_HashInit(hSessionHandle, SGD_SM3, &puckey,
            (unsigned char *)"1234567812345678", 16);
ERR_COND(ret, -1, out, "SDF_HashInit failed 0x%x\n", ret);
ret = SDF_HashFinal(hSessionHandle, hash_z, &zlen);
ERR_COND(ret || zlen != 32, -1, out,
            "SDF_HashFinal failed 0x%x\n", ret);

printf("input digest:\n");
hexdump(hash_z, zlen);

ret = SDF_InternalSign_ECC(hSessionHandle,
            idx, hash_z, zlen, &sign);
ERR_COND(ret, ret, out, "SDF_InternalSign_ECC failed 0x%x\n", ret);

printf("output sign:\n");
hexdump(sign.r + 32, 32);
hexdump(sign.s + 32, 32);

ret = SDF_ExternalVerify_ECC(hSessionHandle, SGD_SM2_1,
                &puckey, hash_z, zlen, &sign);
ERR_COND(ret, ret, out,
            "SDF_ExternalVerify_ECC failed 0x%x\n", ret);
printf("verify sign success by SDF_ExternalVerify_ECC\n");

ret = SDF_InternalVerify_ECC(hSessionHandle,
            idx, hash_z, zlen, &sign);
ERR_COND(ret, ret, out,
        "SDF_InternalVerify_ECC failed 0x%x\n", ret);
printf("verify sign success by SDF_InternalVerify_ECC\n");
```

```
    ret = SDR_OK;
out:
    if (hSessionHandle)
        SDF_CloseSession(hSessionHandle);
    if (hDeviceHandle)
        SDF_CloseDevice(hDeviceHandle);
    return ret;
}
```

```
// 创建SM2加密密钥：sudo hag tkm intl_key_gen -idx 1 -type sm2enc -password 12345678
// 编译：gcc sdf_sm2_enc.c -lhysdf -lhct -o sdf_sm2_enc
// 运行：sudo ./sdf_sm2_enc
// 销毁SM2加密密钥：sudo hag tkm key_destroy -idx 1 -type sm2enc -password 12345678
// 功能：使用海光SDF接口计算SM2加密



#include <hysdf.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define ERR_COND(cond, err, label, fmt, ...) \
    if (cond) {                          \
        printf("ERR: %s:%u: " fmt, __func__, __LINE__, ##__VA_ARGS__);   \
        ret = err;                       \
        goto label;                       \
    }

#define hexdump(data, len) {         \
    fprintf(stdout, "%s in %p, size %d:", #data, data, (len));     \
    _hexdump_(data, len);   \
}

__attribute__ ((unused)) static void _hexdump_(uint8_t *data, uint32_t len)
{
    uint32_t i = 0;
    for (i = 0; i < len; ++i) {
        if (i % 32 == 0) fprintf(stdout, "\n");
        fprintf(stdout, "%02X", data[i]);
    }
    fprintf(stdout, "\n");
}
```

```
int main ()
{
    int ret = 0;
    void *hDeviceHandle = NULL;
    void *hSessionHandle = NULL;
    ECCCipher *encdata = NULL;
    ECCrefPublicKey puckey;
    unsigned char data[32], plaintext[32];
    unsigned int plainlen = 32, idx = 1;
    char *password = "12345678";

    ret = SDF_OpenDevice(&hDeviceHandle);
    ERR_COND(ret || hDeviceHandle == NULL, ret, out, "SDF_OpenDevice failed
0x%x\n", ret);
    ret = SDF_OpenSession(hDeviceHandle, &hSessionHandle);
    ERR_COND(ret || hSessionHandle == NULL, ret, out, "SDF_OpenSession failed
0x%x\n", ret);
    ret = SDF_GetPrivateKeyAccessRight(hSessionHandle, idx, (unsigned
char*)password, strlen(password));
    ERR_COND(ret, ret, out, "SDF_GetPrivateKeyAccessRight failed 0x%x\n", ret);

    ret = SDF_GenerateRandom(hSessionHandle, plainlen, data);
    ERR_COND(ret, -1, out, "SDF_GenerateRandom failed 0x%x\n", ret);
    printf("\ninput data:\n");
    hexdump(data, plainlen);

    encdata = malloc(sizeof(ECCCipher) + plainlen);
    ret = SDF_InternalEncrypt_ECC(hSessionHandle, idx, data, 32, encdata);
    ERR_COND(ret, ret, out, "SDF_InternalEncrypt_ECC failed 0x%x\n", ret);
    printf("\nouput sm2 ciphertext:\n");
    hexdump(encdata->x + 32, 32);
    hexdump(encdata->y + 32, 32);
    hexdump(encdata->M, 32);
    hexdump(encdata->C, encdata->L);

    ret = SDF_InternalDecrypt_ECC(hSessionHandle, idx, encdata, plaintext,
&plainlen);
    ERR_COND(ret, ret, out, "SDF_InternalDecrypt_ECC failed 0x%x\n", ret);
    printf("\noutput decrypted data:\n");
    hexdump(plaintext, plainlen);
    ERR_COND(memcmp(data, plaintext, 32), -1, out, "memcmp data and
plaintext failed\n");

    printf("\ndecrypt success:\n");
    ret = SDR_OK;
```

```
out:
    if (encdata)
        free(encdata);
    if (hSessionHandle)
        SDF_CloseSession(hSessionHandle);
    if (hDeviceHandle)
        SDF_CloseDevice(hDeviceHandle);
    return ret;
}
```

A.20 海光sdf_sm3算法接口示例

```
// 编译：gcc sdf_sm3.c -lhysdf -lhct -o sdf_sm3
// 运行：sudo ./sdf_sm3
// 功能：使用海光SDF接口计算SM3摘要


#include <hysdf.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define ERR_COND(cond, err, label, fmt, ...) \
    if (cond) {                          \
        printf("ERR: %s:%u: " fmt, __func__, __LINE__, ##__VA_ARGS__);   \
        ret = err;                       \
        goto label;                       \
    }

#define hexdump(data, len) {         \
    fprintf(stdout, "%s in %p, size %d:", #data, data, (len));     \
    _hexdump_(data, len);   \
}

__attribute__ ((unused)) static void _hexdump_(uint8_t *data, uint32_t len)
{
    uint32_t i = 0;
    for (i = 0; i < len; ++i) {
        if (i % 32 == 0) fprintf(stdout, "\n");
        fprintf(stdout, "%02X", data[i]);
    }
    fprintf(stdout, "\n");
}
```

```
int main ()
{
    int ret = 0, i, n;
    void *hDeviceHandle = NULL;
    void *hSessionHandle = NULL;
    unsigned int len;
    unsigned char *data = NULL, hash[32];

    ret = SDF_OpenDevice(&hDeviceHandle);
    ERR_COND(ret || hDeviceHandle == NULL, ret, out, "SDF_OpenDevice failed
0x%x\n", ret);
    ret = SDF_OpenSession(hDeviceHandle, &hSessionHandle);
    ERR_COND(ret || hSessionHandle == NULL, ret, out, "SDF_OpenSession failed
0x%x\n", ret);

    n = 3;
    data = malloc(n * 32);
    SDF_GenerateRandom(hSessionHandle, n * 32, data);
    memset(hash, 0, 32);
    SDF_HashInit(hSessionHandle, SGD_SM3, NULL, NULL, 0);
    for (i = 0; i < n; ++i) {
        unsigned offset = i * 32;
        printf("block %d:\n", i + 1);
        hexdump(data + offset, 32);
        ret = SDF_HashUpdate(hSessionHandle, data + offset, 32);
        ERR_COND(ret, ret, out, "SDF_HashUpdate failed 0x%x\n", ret);
    }
    ret = SDF_HashFinal(hSessionHandle, hash, &len);
    ERR_COND(ret || len != 32, -1, out, "SDF_HashFinal failed 0x%x\n", ret);

    printf("\ndigest:\n");
    hexdump(hash, len);
    ret = SDR_OK;
out:
    if (hSessionHandle)
        SDF_CloseSession(hSessionHandle);
    if (hDeviceHandle)
        SDF_CloseDevice(hDeviceHandle);
    if (data)
        free(data);
    return ret;
}
```

A.21 海光sdf_sm4算法接口示例

```
// 创建KEK密钥：sudo hag tkm intl_key_gen -idx 1 -type sm4 -password 12345678
// 编译：gcc sdf_sm4.c -lhysdf -lhct -o sdf_sm4
// 运行：sudo ./sdf_sm4
// 销毁KEK密钥：sudo hag tkm key_destroy -idx 1 -type sm4 -password 12345678
// 功能：使用海光SDF接口计算SM4加密



#include <hysdf.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define ERR_COND(cond, err, label, fmt, ...) \
    if (cond) {                          \
        printf("ERR: %s:%u: " fmt, __func__, __LINE__, ##__VA_ARGS__);   \
        ret = err;                       \
        goto label;                       \
    }

#define hexdump(data, len) {         \
    fprintf(stdout, "%s in %p, size %d:", #data, data, (len));     \
    _hexdump_(data, len);   \
}

__attribute__ ((unused)) static void _hexdump_(uint8_t *data, uint32_t len)
{
    uint32_t i = 0;
    for (i = 0; i < len; ++i) {
        if (i % 32 == 0) fprintf(stdout, "\n");
        fprintf(stdout, "%02X", data[i]);
    }
    fprintf(stdout, "\n");
}

int main ()
{
    int ret = 0, i, j;
    void *hDeviceHandle = NULL;
    void *hSessionHandle = NULL;
    unsigned char *data = NULL, *enc_data = NULL, *dec_data = NULL,
```

```
orig_iv[16], iv[16];
    void *KeyHandle = NULL;
    unsigned char pucKey[32];
    unsigned int puiKeyLength, enc_len, dec_len, idx = 1;
    unsigned int algid[] = {SGD_SM4_CBC, SGD_SM4_ECB, SGD_SM4_CFB, SGD_SM4_OFB, SGD_SM4_CTR,
                            SGD_AES_CBC, SGD_AES_ECB, SGD_AES_CFB, SGD_AES_OFB, SGD_AES_CTR
                        };
    char *password = "12345678";

    ret = SDF_OpenDevice(&hDeviceHandle);
    ERR_COND(ret || hDeviceHandle == NULL, ret, out, "SDF_OpenDevice failed 0x%x\n", ret);
    ret = SDF_OpenSession(hDeviceHandle, &hSessionHandle);
    ERR_COND(ret || hSessionHandle == NULL, ret, out, "SDF_OpenSession failed 0x%x\n", ret);
    ret = SDF_GetPrivateKeyAccessRight(hSessionHandle, idx, (unsigned char*)password, strlen(password));
    ERR_COND(ret, ret, out, "SDF_GetPrivateKeyAccessRight failed 0x%x\n", ret);

    ret = SDF_GenerateKeyWithKEK(hSessionHandle, 128, SGD_SM4, idx, pucKey, &puiKeyLength, &KeyHandle);
    ERR_COND(ret || puiKeyLength != 16, -1, out, "SDF_GenerateKeyWithKEK failed %d\n", puiKeyLength);
    ret = SDF_DestroyKey(hSessionHandle, KeyHandle);
    ERR_COND(ret, ret, out, "SDF_DestroyKey failed 0x%x\n", ret);
    ret = SDF_ImportKeyWithKEK(hSessionHandle, SGD_SM4, idx, pucKey, puiKeyLength, &KeyHandle);
    ERR_COND(ret, ret, out, "SDF_ImportKeyWithKEK failed 0x%x\n", ret);

    for (i = 0; i < 10; ++i) {
        int n = 2;
        data = malloc(n * 16);
        enc_data = malloc(n * 16);
        dec_data = malloc(n * 16);
        SDF_GenerateRandom(hSessionHandle, 16, iv);
        SDF_GenerateRandom(hSessionHandle, n * 16, data);
        memcpy(orig_iv, iv, 16);
        memset(enc_data, 0, n * 16);
        memset(dec_data, 0, n * 16);
        printf("input data\n");
        hexdump(data, n * 16);
        for (j = 0; j < n; ++j) {
            unsigned offset = j * 16;
            ret = SDF_Encrypt(hSessionHandle, KeyHandle, algid[i], iv, data + offset, 16, enc_data + offset, &enc_len);
```

```
        ERR_COND(ret || enc_len != 16, -1, out, "SDF_Encrypt failed 0x%x\n",
ret);
    }
    printf("output encrypted data\n");
    hexdump(enc_data, n * 16);
    for (j = 0; j < n; ++j) {
        unsigned offset = j * 16;
        ret = SDF_Decrypt(hSessionHandle, KeyHandle, algid[i], orig_iv, enc_data
+ offset, 16, dec_data + offset, &dec_len);
        ERR_COND(ret || dec_len != 16, -1, out, "SDF_Encrypt failed 0x%x\n",
ret);
    }
    printf("output decrypted data\n");
    hexdump(dec_data, n * 16);

    ERR_COND(memcmp(data, dec_data, n*16), -1, out, "SDF_Encrypt failed
0x%x\n", ret);
    free(data);
    free(enc_data);
    free(dec_data);
    data = NULL;
    enc_data = NULL;
    dec_data = NULL;
    printf("SDF_Encrypt algid 0x%x test success\n\n", algid[i]);
  }

  ret = SDF_DestroyKey(hSessionHandle, KeyHandle);
  ERR_COND(ret, ret, out, "SDF_DestroyKey failed 0x%x\n", ret);
  ret = SDR_OK;
out:
  if (hSessionHandle)
    SDF_CloseSession(hSessionHandle);
  if (hDeviceHandle)
    SDF_CloseDevice(hDeviceHandle);
  if (data)
    free(data);
  if (enc_data)
    free(enc_data);
  if (dec_data)
    free(dec_data);
  return ret;
}
```

A.22 鲲鹏KAE SM3算法接口示例

```
//编译：gcc sm3.c -lcrypto -o kae_sm3

//运行：./kae_sm3
//功能：使用kae加速引擎进行sm3运算
#include <stdio.h>
#include <openssl/evp.h>
#include <openssl/engine.h>

int main()
{
    ENGINE *e = NULL;

    OpenSSL_add_all_algorithms();
    e = ENGINE_by_id("kae");
    if(e != NULL)
        ENGINE_set_default(e, ENGINE_METHOD_ALL);

    int len, dataLen = 3;
    char source[3]={'a','b','c'};
    const EVP_MD *sm3_md;
    EVP_MD_CTX *sm3_ctx;
    unsigned char hash_value[64];

    sm3_md = EVP_sm3();
    sm3_ctx = EVP_MD_CTX_new();

    EVP_DigestInit_ex(sm3_ctx, sm3_md, e);
    EVP_DigestUpdate(sm3_ctx, source, dataLen);
    EVP_DigestFinal_ex(sm3_ctx, hash_value, &len);

    printf("hash_value:");
    for(int i = 0; i < len; i++)
        printf("%02x ",hash_value[i]);
    printf("\n");

    EVP_MD_CTX_free(sm3_ctx);
    return 0;
}
```

A.23 鲲鹏KAE SM4算法接口示例

```
//编译：gcc -g sm4.c -lcrypto -o kae_sm4
```

```
//运行：./kae_sm4
//功能：sm4加解密测试
#include <stdio.h>
#include <assert.h>
#include <openssl/evp.h>
#include <openssl/engine.h>

void printHex(char *name, unsigned char *c, int n)
{
    int i;
    printf("%s: ",name);
    for (i = 0; i < n; i++)
    {
        printf("%02X", c[i]);
    }
    printf("\n");
}

int main()
{
    ENGINE *e = NULL;
    int outdata_len = 0;
    int dataLen = 16;
    int plain_dataLen = 17;
    int enc_dataLen = 16;
    EVP_CIPHER_CTX *ctx;
    const EVP_CIPHER *mode;
    int sm4enStrLen,sm4deStrLen;
    unsigned char *sm4_en = malloc(2*dataLen);
    unsigned char *sm4_de = malloc(2*dataLen);
    unsigned char
key[16]={0x15,0x67,0x28,0xe1,0x5f,0x9a,0xfc,0x01,0xd4,0xb6,0x1
b,0x4e,0x44,0x5d,0xbb,0x26};
    unsigned char
plain[17]={0x15,0x67,0x28,0xe1,0x5f,0x9a,0xfc,0x01,0xd4,0xb6,0x
1b,0x4e,0x44,0x5d,0xbb,0x26,0x1};
unsigned char
iv[16]={0x15,0x67,0x28,0xe1,0x5f,0x9a,0xfc,0x01,0xd4,0xb6,0x1b,
0x4e,0x44,0x5d,0xbb,0x26};

    OpenSSL_add_all_algorithms();
    e = ENGINE_by_id("kae");
    if(e != NULL)
        ENGINE_set_default(e, ENGINE_METHOD_ALL);

    ctx = EVP_CIPHER_CTX_new();
    assert(ctx != NULL && "CIPHER_CTX_new");
```

```
    mode = EVP_sm4_ecb();
    assert(mode != NULL);

    /* Encrypt */
    EVP_EncryptInit_ex(ctx, mode, NULL, key, NULL);
    if( 0 == plain_dataLen % 16 )
    {
       EVP_CIPHER_CTX_set_padding( ctx, 0 );
    }else
        EVP_CIPHER_CTX_set_padding( ctx, 1 );
    if (!EVP_EncryptUpdate(ctx, sm4_en, &sm4enStrLen, plain,
plain_dataLen))
        return -1;
    outdata_len = sm4enStrLen;
    sm4enStrLen = 0;
    if (!EVP_EncryptFinal_ex(ctx, sm4_en+outdata_len,
&sm4enStrLen))
        return -1;
    sm4enStrLen += outdata_len;
    printHex("Encrypt", sm4_en, sm4enStrLen);


    /* Decrypt */
    EVP_DecryptInit_ex(ctx, mode, NULL, key, NULL);
    if( 0 == sm4enStrLen % 16 )
    {
       EVP_CIPHER_CTX_set_padding( ctx, 0 );
    }else
        EVP_CIPHER_CTX_set_padding( ctx, 1 );
    if (!EVP_DecryptUpdate(ctx, sm4_de, &sm4deStrLen, sm4_en,
sm4enStrLen)) {
        return -1;
    }
    outdata_len = sm4deStrLen;
    sm4deStrLen = 0;
    if (!EVP_DecryptFinal_ex(ctx, sm4_de + outdata_len,
&sm4deStrLen)) {
        return -1;
    }
    sm4deStrLen += outdata_len;
    printHex("Decrypt", sm4_de, sm4deStrLen);

    EVP_CIPHER_CTX_free(ctx);
    return 0;
}
```

A.24 兆芯sm2算法接口示例

```
//编译：gcc sm2_enc.c -lcrypto -o _sm2_enc

//运行：./_sm2_enc
//功能：使用加速引擎进行sm2加密运算
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#include <openssl/ec.h>
#include <openssl/evp.h>
#include <openssl/engine.h>

static char* engine_name = "";
ENGINE *e = NULL;

void printECKeyData(const EC_KEY* key)
{
    BIGNUM* gx = BN_new();
    BIGNUM* gy = BN_new();
    printf("private key: %s\n", BN_bn2hex(EC_KEY_get0_private_key(key)));
    const EC_POINT * point  = EC_KEY_get0_public_key(key);
    if(EC_POINT_get_affine_coordinates_GFp(
                EC_KEY_get0_group(key), point, gx, gy , NULL)){
        printf("public key x: %s\n", BN_bn2hex(gx));
        printf("public key y: %s\n", BN_bn2hex(gy));
    }
    BN_free(gx);
    BN_free(gy);
}

int Encrypt(const unsigned char* in_buf, size_t in_buflen, unsigned char*
out_encrypted, size_t* len_encrypted, EC_KEY* pubKey)
{
    printf("----------------------------------------encrypt ----------------------------------------
---\n");
    int ret = -1, i;
    EVP_PKEY_CTX * ectx = NULL;
    EVP_PKEY* pkey = NULL;
    EC_KEY* key_pair = NULL;
    unsigned char* ciphertext = NULL;
    size_t ciphertext_len, plaintext_len;
    pkey = EVP_PKEY_new();

#if 1
```

```
    EVP_PKEY_set1_SM2(pkey, pubKey);
#else
    if(!EVP_PKEY_set1_EC_KEY(pkey, pubKey)) {
        printf("Error set public key");
        goto clean_up;
    }
#endif
    /* compute SM2 encryption */
    if ((EVP_PKEY_set_alias_type(pkey, EVP_PKEY_SM2)) != 1)
    {
        printf("EVP_PKEY_set_alias_type failed.");
        goto clean_up;
    }

    if (!(ectx = EVP_PKEY_CTX_new(pkey, e)))
    {
        printf("EVP_PKEY_CTX_new failed.");
        goto clean_up;
    }

    if ((EVP_PKEY_encrypt_init(ectx)) != 1)
    {
        printf("EVP_PKEY_encrypt failed.");
        goto clean_up;
    }

    if ((EVP_PKEY_encrypt(ectx, NULL, &ciphertext_len, in_buf, in_buflen)) != 1)
    {
        printf("EVP_PKEY_set_alias_type failed.");
        goto clean_up;
    }

    if (!(ciphertext = OPENSSL_malloc(ciphertext_len)))
    {
        goto clean_up;
    }

    if ((ret = EVP_PKEY_encrypt(ectx, ciphertext, &ciphertext_len, in_buf,
in_buflen)) != 1)
    {
        printf("EVP_PKEY_encrypt failed.");
        goto clean_up;
    }
    memcpy(out_encrypted, ciphertext, ciphertext_len);
    *len_encrypted = ciphertext_len;
    ret = 0;
clean_up:
```

```
    if (pkey)
    {
        EVP_PKEY_free(pkey);
    }

    if (ectx)
    {
        EVP_PKEY_CTX_free(ectx);
    }

    if (ciphertext)
    {
        free(ciphertext);
    }

    return ret;
}

int Decrypt(const unsigned char* in_buf, size_t in_buflen, unsigned char*
out_plaint, size_t* len_plaint,  EC_KEY* prikey)
{
    printf("----------------------------------------decrypt ----------------------------------------
---\n");
    int ret = -1, i;
    EVP_PKEY_CTX* ectx = NULL;
    EVP_PKEY* pkey = NULL;
    EC_KEY* key_pair = NULL;
    unsigned char * plaintext = NULL;
    size_t ciphertext_len, plaintext_len;

    pkey = EVP_PKEY_new();
#if 1
    EVP_PKEY_set1_SM2(pkey, prikey);
#else
    if(!EVP_PKEY_set1_EC_KEY(pkey, prikey)) {
        printf("Error set public key");
        goto clean_up;
    }
#endif
    if ((EVP_PKEY_set_alias_type(pkey, EVP_PKEY_SM2)) != 1)
    {
        printf("EVP_PKEY_set_alias_type failed.");
        goto clean_up;
    }

    if (!(ectx = EVP_PKEY_CTX_new(pkey, e)))
    {
```

```
        printf("EVP_PKEY_CTX_new failed.");
        goto clean_up;
    }

    /* compute SM2 decryption */
    if ((EVP_PKEY_decrypt_init(ectx)) != 1)
    {
        printf("EVP_PKEY_decrypt_init failed.");
        goto clean_up;
    }

    if ((EVP_PKEY_decrypt(ectx, NULL, &plaintext_len, in_buf, in_buflen)) != 1)
    {
        printf("EVP_PKEY_decrypt failed.");
        goto clean_up;
    }

    if (!(plaintext = OPENSSL_malloc(plaintext_len)))
    {
        printf("plaintext is NULL\n");
        goto clean_up;
    }

    if ((ret = EVP_PKEY_decrypt(ectx, plaintext, &plaintext_len, in_buf, in_buflen))
!= 1)
    {
        printf("EVP_PKEY_decrypt failed.");
        goto clean_up;
    }
    memcpy(out_plaint, plaintext, plaintext_len);
    *len_plaint = plaintext_len;
    ret = 0;
clean_up:
    if (pkey)
    {
        EVP_PKEY_free(pkey);
    }
    if (plaintext)
    {
        free(plaintext);
    }
    return ret;
}

int main(int argc ,char* argv[])
{
    EC_KEY* sm2Key = NULL;
```

```
    EC_GROUP* group = NULL;
    int count = 1;
    group = EC_GROUP_new_by_curve_name(NID_sm2);
    if (group == NULL) {
        printf("EC_GROUP_new_by_curve_name failed\n");
        return -1;
    }
    sm2Key = EC_KEY_new();
    if (!sm2Key) {
        printf("EC_KEY_new failed!\n");
        return -1;
    }
    int ret = EC_KEY_set_group(sm2Key, group);
    if (ret != 1) {
        printf("EC_KEY_set_group failed!\n");
        return -1;
    }
    /*generate the key*/
    ret = EC_KEY_generate_key(sm2Key);
    if (ret != 1) {
        printf("EC_KEY_generate_key failed!\n");
        return -1;
    }
    printECKeyData(sm2Key);
    ENGINE_load_builtin_engines();
    if ((e = ENGINE_by_id(engine_name)) == NULL) {
        printf("ENGINE_by_id error\n");
        return -1;
    }
    unsigned char* message = "123456";
    const size_t msg_len = 6;
    unsigned char ciphertext[1024] = {0};
    size_t ciphertextLen = 0;
    unsigned char origintext[1024] = {0};
    size_t origintextLen = 0;

    while (count > 0) {
        memset(ciphertext, 0, 1024);
        ciphertextLen = 0;
        Encrypt((unsigned char*)message, msg_len, ciphertext, &ciphertextLen,
sm2Key);
        printf("cipher  text len: %d\n", ciphertextLen);
        for (int i = 0; i < ciphertextLen; ++i) {
            printf("%02x ", ciphertext[i]);
        }
        printf("\n");
        memset(origintext, 0, 1024);
```

```
        origintextLen = 0;
        Decrypt(ciphertext, ciphertextLen, origintext, &origintextLen, sm2Key);

        for (int i = 0; i < origintextLen; ++i) {
            printf("%02x ", origintext[i]);
        }
        printf("\n");

        count--;
    }
    printf("decrypt data: %s\n", origintext);
    if (strncasecmp(origintext, message, origintextLen) == 0) {
        printf("success\n");
    }

    EC_KEY_free(sm2Key);
    EC_GROUP_free(group);
    return 0;
}

//编译：gcc sm2_sig.c -lcrypto -o _sm2_sig

//运行：./_sm2_sig
//功能：使用加速引擎进行sm2签名运算
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>

#include <openssl/ec.h>
#include <openssl/evp.h>
#include <openssl/engine.h>

static char* engine_name = "";
static ENGINE *e = NULL;

int EVP_compute_msg_hash(const EVP_MD *digest, const uint8_t *msg, size_t
msg_len, uint8_t *dgst, size_t *dgst_len)
{
  EVP_MD_CTX *hash = EVP_MD_CTX_new();
  const int md_size = EVP_MD_size(digest);
  int ret = 0;

  if (dgst == NULL) {
    goto done;
  }
```

```
  if (md_size < 0) {
    goto done;
  }

  if (hash == NULL) {
    goto done;
  }
  int tmp = 0;

  if (!EVP_DigestInit_ex(hash, digest, e) ||
      !EVP_DigestUpdate(hash, msg, msg_len) ||
      !EVP_DigestFinal_ex(hash, dgst, &tmp)) {
    goto done;
  }
  *dgst_len = tmp;
  ret = 1;

done:
  EVP_MD_CTX_free(hash);
  return ret;
}

int main(int argc ,char* argv[])
{

    EC_KEY* sm2Key = NULL;
    EC_GROUP* group = NULL;
    EVP_PKEY* sm2_pkey = NULL;
    EVP_PKEY_CTX *sm2_pctx = NULL;
    EVP_PKEY_CTX *sm2_vfy_pctx = NULL;

#if 1
    ENGINE_load_builtin_engines();
    if ((e = ENGINE_by_id(engine_name)) == NULL) {
            printf("ENGINE_by_id error\n");
            return -1;
    }
#endif

     group = EC_GROUP_new_by_curve_name(NID_sm2);
     if (group == NULL) {
          printf("EC_GROUP_new_by_curve_name failed\n");
          return -1;
     }

     sm2Key = EC_KEY_new();
     if (!sm2Key) {
```

```
            printf("EC_KEY_new failed!\n");
            return -1;
        }
        int ret = EC_KEY_set_group(sm2Key, group);
        if (ret != 1) {
            printf("EC_KEY_set_group failed!\n");
            return -1;
        }
        /*generate the key*/
        ret = EC_KEY_generate_key(sm2Key);
        if (ret != 1) {
            printf("EC_KEY_generate_key failed!\n");
            return -1;
        }

        const char* message = "the encryption data";
        const size_t msg_len = strlen(message);

       sm2_pkey = EVP_PKEY_new();
       if (sm2_pkey == NULL) {
        return -1;
       }

       if (!EVP_PKEY_set1_SM2(sm2_pkey, sm2Key)) {
        return -1;
       }

       sm2_pctx = EVP_PKEY_CTX_new(sm2_pkey, e);
       if (sm2_pctx == NULL) {
          return -1;
         }

       unsigned char hash[32] = {0};
       size_t hashLen = 0;
       if (!EVP_compute_msg_hash(EVP_sm3(), message, msg_len, hash, &hashLen))
    {
            printf("EVP_compute_msg_hash failed\n");
            return -1;
       }

        if (EVP_PKEY_sign_init(sm2_pctx) <= 0) {
            printf("EVP_PKEY_sign_init failed\n");
            return -1;
        }

        size_t signValueLen = 0;
```

```
    if (EVP_PKEY_sign(sm2_pctx, NULL , &signValueLen, hash, hashLen) <= 0) {
        printf("EVP_PKEY_sign first failed\n");
        return -1;
    }
    unsigned char* signValue = malloc(signValueLen+1);
    memset(signValue, 0, signValueLen+1);
    if (!(ret = EVP_PKEY_sign(sm2_pctx, signValue, &signValueLen, hash,
hashLen)) ) {
        printf("EVP_PKEY_sign sec failed\n");
        return -1;
    }

   EVP_PKEY* ver_pkey = EVP_PKEY_new();
   if (ver_pkey == NULL) {
      return -1;
   }

   if (!EVP_PKEY_set1_SM2(ver_pkey, sm2Key)) {
      return -1;
   }

   sm2_vfy_pctx = EVP_PKEY_CTX_new(ver_pkey, e);
   if (sm2_vfy_pctx == NULL) {
      return -1;
     }

    if (EVP_PKEY_verify_init(sm2_vfy_pctx) <= 0) {
        printf("EVP_PKEY_verify_init failed\n");
        return -1;
    }

    if (EVP_PKEY_verify(sm2_vfy_pctx, signValue, signValueLen, hash, hashLen)
!= 1) {
        printf("EVP_PKEY_verify failed\n");
    }
    printf("verify ok\n");

    EVP_PKEY_free(sm2_pkey);
    EVP_PKEY_CTX_free(sm2_pctx);
    EVP_PKEY_CTX_free(sm2_vfy_pctx);
    free(signValue);

    return 0;
}
```

A.25 兆芯sm3算法接口示例

```
//编译：gcc sm3.c -lcrypto -o _sm3

//运行：./_sm3
//功能：使用加速引擎进行sm3运算
#include <stdio.h>
#include <openssl/evp.h>
#include <openssl/engine.h>

int main()
{
    ENGINE *e = NULL;
    ENGINE_load_builtin_engines();
    e = ENGINE_by_id("");
    if(e == NULL)
        return -1;

    int len, dataLen = 3;
    char source[3]={'a','b','c'};
    const EVP_MD *sm3_md;
    EVP_MD_CTX *sm3_ctx;
    unsigned char hash_value[64];

    sm3_md = EVP_sm3();
    sm3_ctx = EVP_MD_CTX_new();

    EVP_DigestInit_ex(sm3_ctx, sm3_md, e);
    EVP_DigestUpdate(sm3_ctx, source, dataLen);
    EVP_DigestFinal_ex(sm3_ctx, hash_value, &len);

    printf("hash_value:");
    for(int i = 0; i < len; i++)
        printf("%02x ",hash_value[i]);
    printf("\n");

    EVP_MD_CTX_free(sm3_ctx);
    return 0;
}
```

A.26 兆芯sm4算法接口示例

```
//编译：gcc -g sm4.c -lcrypto -o _sm4

//运行：./_sm4
```

```
//功能：sm4加解密测试
#include <stdio.h>
#include <assert.h>
#include <openssl/evp.h>
#include <openssl/engine.h>

void printHex(char *name, unsigned char *c, int n)
{
    int i;
    printf("%s: ",name);
    for (i = 0; i < n; i++)
    {
        printf("%02X", c[i]);
    }
    printf("\n");
}

int main()
{
    ENGINE *e = NULL;
    int outdata_len = 0;
    int dataLen = 16;
    int plain_dataLen = 17;
    int enc_dataLen = 16;
    EVP_CIPHER_CTX *ctx;
    const EVP_CIPHER *mode;
    int sm4enStrLen,sm4deStrLen;
    unsigned char *sm4_en = malloc(2*dataLen);
    unsigned char *sm4_de = malloc(2*dataLen);
    unsigned char
key[16]={0x15,0x67,0x28,0xe1,0x5f,0x9a,0xfc,0x01,0xd4,0xb6,0x1
b,0x4e,0x44,0x5d,0xbb,0x26};
    unsigned char
plain[17]={0x15,0x67,0x28,0xe1,0x5f,0x9a,0xfc,0x01,0xd4,0xb6,0x
1b,0x4e,0x44,0x5d,0xbb,0x26,0x1};
unsigned char
iv[16]={0x15,0x67,0x28,0xe1,0x5f,0x9a,0xfc,0x01,0xd4,0xb6,0x1b,
0x4e,0x44,0x5d,0xbb,0x26};

    ENGINE_load_builtin_engines();
    e = ENGINE_by_id("");
    if(e == NULL)
        return -1;

    ctx = EVP_CIPHER_CTX_new();
    assert(ctx != NULL && "CIPHER_CTX_new");

    mode = EVP_sm4_ecb();
```

```
    assert(mode != NULL);

    /* Encrypt */
    EVP_EncryptInit_ex(ctx, mode, NULL, key, NULL);
    if( 0 == plain_dataLen % 16 )
    {
       EVP_CIPHER_CTX_set_padding( ctx, 0 );
    }else
        EVP_CIPHER_CTX_set_padding( ctx, 1 );
    if (!EVP_EncryptUpdate(ctx, sm4_en, &sm4enStrLen, plain,
plain_dataLen))
        return -1;
    outdata_len = sm4enStrLen;
    sm4enStrLen = 0;
    if (!EVP_EncryptFinal_ex(ctx, sm4_en+outdata_len,
&sm4enStrLen))
        return -1;
    sm4enStrLen += outdata_len;
    printHex("Encrypt", sm4_en, sm4enStrLen);


    /* Decrypt */
    EVP_DecryptInit_ex(ctx, mode, NULL, key, NULL);
    if( 0 == sm4enStrLen % 16 )
    {
       EVP_CIPHER_CTX_set_padding( ctx, 0 );
    }else
        EVP_CIPHER_CTX_set_padding( ctx, 1 );
    if (!EVP_DecryptUpdate(ctx, sm4_de, &sm4deStrLen, sm4_en,
sm4enStrLen)) {
        return -1;
    }
    outdata_len = sm4deStrLen;
    sm4deStrLen = 0;
    if (!EVP_DecryptFinal_ex(ctx, sm4_de + outdata_len,
&sm4deStrLen)) {
        return -1;
    }
    sm4deStrLen += outdata_len;
    printHex("Decrypt", sm4_de, sm4deStrLen);

    EVP_CIPHER_CTX_free(ctx);
    return 0;
}
```

A.27 飞腾机密计算接口示例

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#include <kysec_tee_km.h>
#include <kysec_tee_algo.h>
#include <kysec_tee_sos.h>
/**
 * @brief sm2_sign_verify_test  sm2签名验签
 * @param plain 待签名数据
 * @param len   待签名数据长度
 * @return
 */
int sm2_sign_verify_test(uint8_t *plain, uint32_t len)
{
uint32_t ret;
uint8_t pub_key[64] = {0};
uint8_t pri_key[32] = {0};
uint8_t signature[64] = {0};
   KYSEC_TEE_DATA data;
   data.dptr = plain;
   data.dsize = len;
//printf("SM2 sign & verify self-check start...\n");
   ret = kysec_tee_algo_sm2_keygen(pub_key, pri_key); //生成公私钥对
if (ret)
{
printf("sm2 get key failure\n");
return 0;
}
else
{
printf("pubkey: ");
for (int i = 0; i < 64; i++)
printf("%c ", pub_key[i]);
printf("\n");
printf("pubkey: ");
for (int i = 0; i < 64; i++)
```

```
printf("%.2x ", pub_key[i]);
printf("\n");
}
    ret = kysec_tee_algo_sm2_sign(data, pub_key, pri_key, signature); //签名
if(ret) {
printf("sign failure ret = %d\n", ret);
return 0;
}
    ret = kysec_tee_algo_sm2_verify(data, pub_key, signature); //验签
if(ret) {
printf("verify failure\n");
return 0;
}
printf("SM2 sign & verify self-check sucessful\n");
return 0;
}
/**
 * @brief sm2_encrypt_decrypt_test sm2加解密测试
 * @param plain 待加密数据
 * @param len 待加密数据长度
 * @return
 */
int sm2_encrypt_decrypt_test(uint8_t *plain, uint32_t len)
{
    KT_Result ret;
uint8_t pub_key[64] = {0};
uint8_t pri_key[32] = {0};
printf("SM2 encrypt & decrypt self-check start ...\n");
    ret = kysec_tee_algo_sm2_keygen(pub_key, pri_key); //生成公私钥对
if (ret)
{
printf("sm2 get key failure\n");
return 0;
}
//C1C3C2
    KYSEC_TEE_DATA data;
    data.dptr = plain;
    data.dsize = len;
    KYSEC_TEE_DATA encrypt_data;
    ret = kysec_tee_algo_sm2_encrypt(data, pub_key, &encrypt_data); //使用公钥加密
```

```
if (ret)
{
printf("sm2 encrypt failure\n");
return 0;
}
    KYSEC_TEE_DATA decrypt_data;
    ret = kysec_tee_algo_sm2_decrypt(encrypt_data, pub_key, pri_key, &decrypt_data); //
使用私钥解密
//检测加解密结果， 解密数据长度、内容与源数据一致则通过
if(ret || (len != decrypt_data.dsize)||
memcmp(plain, decrypt_data.dptr, decrypt_data.dsize)) {
printf("sm2 decrypt failure\n");
free(encrypt_data.dptr);
        encrypt_data.dptr = NULL;
return 0;
}
free(encrypt_data.dptr);
    encrypt_data.dptr = NULL;
free(decrypt_data.dptr);
    decrypt_data.dptr = NULL;
printf("SM2 encrypt & decrypt self-check sucessful\n");
return 0;
}
/**
 * @brief sm2_test sm2 算法测试
 */
void sm2_test()
{
printf("Start sm2 test...\n");
//待签名数据
uint8_t vplain[10] = {0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39};
//待签名数据长度
uint32_t vlen = 10;
sm2_sign_verify_test(vplain, vlen);
//待加密数据
uint8_t plain[32] = {
0x30, 0x82, 0x01, 0x94, 0xa0, 0x03, 0x02, 0x01, 0x02, 0x02, 0x09, 0x00, 0xe8, 0xd2,
0x3c, 0x1f,
0x2f, 0xcf, 0xd6, 0xa4, 0x30, 0x0a, 0x06, 0x08, 0x2a, 0x81, 0x1c, 0xcf, 0x55, 0x01, 0x83,
0x75,
```

```
};
//待加密数据长度
uint32_t len = 32;
sm2_encrypt_decrypt_test(plain, len);
return;
}
/**
 * @brief sm3_functest sm3测试
 * @param input 输入数据
 * @param inlen 输入数据长度
 * @param std_digest    标准摘要计算结果，接口计算结果需与该结果比对
 */
void sm3_functest(uint8_t *input, uint32_t inlen, uint8_t std_digest[32])
{
    KT_Result res;
    ALGO_SM3_CTX *context;
uint8_t digest[32] = {0};
    res = kysec_tee_algo_sm3_init(&context); //初始化sm3上下文
if(res != 0)
{
goto out;
}
    KYSEC_TEE_DATA data;
    data.dptr = input;
    data.dsize = inlen;
    res = kysec_tee_algo_sm3_update(context, data); //sm3计算
if(res != 0)
{
goto out;
}
    res = kysec_tee_algo_sm3_finalize(context,digest); //sm3计算完成
if(res != 0)
{
goto out;
}
//比对计算摘要与标准摘要是否一致
if (memcmp(std_digest, digest, 32) == 0)
{
printf("SM3 cpu mode self-check sucessful\n");
```

```
}
else
{
printf("SM3 cpu mode self-check failure\n");
}
out:
return;
}
/**
 * @brief sm3_test sm3算法测试
 */
void sm3_test()
{
printf("Start sm3 test...\n");
uint8_t input[10]={0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39};
int inlen = 10;
uint8_t std_digest[32] = {0x09, 0x09, 0x3b, 0x72, 0x55, 0x3f, 0x5d, 0x9d, 0x62,
0x2d, 0x6c, 0x62, 0xf5, 0xff, 0xd9, 0x16, 0xee, 0x95,
0x96, 0x79, 0xb1, 0xbd, 0x4d, 0x16, 0x9c, 0x3e, 0x12,
0xaa, 0x83, 0x28, 0xe7, 0x43};
sm3_functest(input, inlen, std_digest);
return;
}
//随机数测试
void random_test()
{
printf("Start rng test...\n");
   KYSEC_TEE_DATA data;
   KT_Result res = kysec_tee_algo_random(32, &data);
if (res != KT_SUCCESS)
printf("RANDOM failure\n");
else {
printf("RANDOM sucessful\n");
free(data.dptr);
      data.dptr = NULL;
}
return;
}
uint8_t std_in[48] = {
0x81, 0x70, 0x99, 0x44, 0xE0, 0xCB, 0x2E, 0x1D, 0xB5, 0xB0, 0xA4, 0x77, 0xD1, 0xA8,
0x53, 0x9B,
```

```
0x0A, 0x87, 0x86, 0xE3, 0x4E, 0xAA, 0xED, 0x99, 0x30, 0x3E, 0xA6, 0x97, 0x55, 0x95,
0xB2, 0x45,
0x4D, 0x5D, 0x7F, 0x91, 0xEB, 0xBD, 0x4A, 0xCD, 0x72, 0x6C, 0x0E, 0x0E, 0x5E, 0x3E,
0xB5, 0x5E,
};
uint8_t std_key[16] = {
0xE0, 0x70, 0x99, 0xF1, 0xBF, 0xAF, 0xFD, 0x7F, 0x24, 0x0C, 0xD7, 0x90, 0xCA, 0x4F,
0xE1, 0x34
};
uint8_t std_iv[16] = {
0xC7, 0x2B, 0x65, 0x91, 0xA0, 0xD7, 0xDE, 0x8F, 0x6B, 0x40, 0x72, 0x33, 0xAD, 0x35,
0x81, 0xD6
};
uint8_t std_ecb_out[48] = {
0xCC, 0x62, 0x37, 0xA6, 0xA1, 0x35, 0x39, 0x75, 0xFF, 0xF5, 0xEE, 0x6A, 0xFD, 0xD7,
0x70, 0x15,
0xE1, 0x32, 0x23, 0x1F, 0x18, 0xB8, 0xC9, 0x16, 0x07, 0x27, 0x9C, 0x6C, 0x7F, 0x8F,
0x7F, 0xF6,
0xFD, 0xF1, 0xE4, 0x01, 0xEC, 0x7E, 0xD2, 0x60, 0xFD, 0xE7, 0x5C, 0xE5, 0xCF, 0x6E,
0xE7, 0x87,
};
uint8_t std_cbc_out[48] = {
0x60, 0x7A, 0xBE, 0xC9, 0xDA, 0xD7, 0x90, 0x73, 0xC7, 0x96, 0xDB, 0x34, 0x26, 0xFD,
0x2C, 0x2F,
0x8E, 0x39, 0xC7, 0x0B, 0x60, 0xB2, 0x3D, 0xBE, 0xF3, 0xA9, 0xA5, 0x46, 0x65, 0x26,
0x41, 0xB7,
0xAE, 0xC9, 0xC3, 0xAD, 0x8C, 0x9B, 0x95, 0x8D, 0x17, 0x53, 0x15, 0x35, 0x40, 0x2A,
0x8C, 0x6B,
};
char *mode_str[5] = {"ecb", "cbc", "cfb", "ofb", "ctr"};
/**
 * @brief sm4_encrypt_decrypt sm4加解密验证
 * @param mode  加解密模式
 * @param std_plain 加密数据
 * @param byteLen   数据长度
 * @param key   密钥
 * @param iv    向量
 * @param std_cipher 标准加密结果
 * @return
 */
```

```
int sm4_encrypt_decrypt(ALGO_SM4_OP_MODE mode, uint8_t *std_plain,
uint32_t byteLen, uint8_t *key, uint8_t *iv, const uint8_t *std_cipher)
{
uint32_t block_byteLen, key_byteLen;
    block_byteLen = 16;
    key_byteLen = 16;
//ENCRYPT
    KT_Result res = KT_SUCCESS;
    KYSEC_TEE_DATA data;
    data.dptr = std_plain;
    data.dsize = byteLen;
    ALGO_SM4_CTX *encpypt_ctx;
    KYSEC_TEE_DATA encrypt_data;
    res = kysec_tee_algo_sm4_init(&encpypt_ctx, mode, SM4_CRYPTO_ENCRYPT, key, iv);
//初始化加密上下文
if (res != KT_SUCCESS) {
printf("init encrypt context failed\n");
return 0;
}
    res = kysec_tee_algo_sm4_update(encpypt_ctx, SM4_CRYPTO_ENCRYPT, data, 1,
&encrypt_data); //加密计算
if (res != KT_SUCCESS) {
printf("sm4 encrypt update failed\n");
return 0;
}
//DECRYPT
    ALGO_SM4_CTX *decpypt_ctx;
    res = kysec_tee_algo_sm4_init(&decpypt_ctx, mode, SM4_CRYPTO_DECRYPT, key, iv);
//初始化解密上下文
if (res != KT_SUCCESS) {
printf("init decrypt context failed\n");
free(encrypt_data.dptr);
        encrypt_data.dptr = NULL;
return 0;
}
    KYSEC_TEE_DATA decrypt_data;
    res = kysec_tee_algo_sm4_update(decpypt_ctx, SM4_CRYPTO_DECRYPT, encrypt_data,
1, &decrypt_data); //解密计算
if (res != KT_SUCCESS) {
printf("sm4 decrypt update failed\n");
free(encrypt_data.dptr);
```

```
    encrypt_data.dptr = NULL;
return 0;
}
//校验加密结果与标准加密结果是否一致
if((std_cipher && memcmp(encrypt_data.dptr, std_cipher, byteLen))) {
printf("SM4 cpu %s self-check failure!! \n", mode_str[mode]);
} else {
printf("SM4 cpu %s self-check success!! \n", mode_str[mode]);
}
//校验解密结果与源数据是否一致
if(memcmp(decrypt_data.dptr, std_plain, byteLen)) {
printf("SM4 cpu %s self-check failure!! \n", mode_str[mode]);
} else {
printf("SM4 cpu %s self-check success!! \n", mode_str[mode]);
}
free(decrypt_data.dptr);
   decrypt_data.dptr = NULL;
free(encrypt_data.dptr);
   encrypt_data.dptr = NULL;
return 0;
}
/**
 * @brief sm4_test sm4测试
 */
void sm4_test()
{
printf("Start sm4 test...\n");
sm4_encrypt_decrypt(SM4_MODE_ECB, std_in, 48, std_key, NULL, std_ecb_out);
sm4_encrypt_decrypt(SM4_MODE_CBC, std_in, 48, std_key, std_iv, std_cbc_out);
return;
}
/**
 * @brief test_algo 算法服务测试
 */
void test_algo()
{
sm2_test();
sm3_test();
sm4_test();
random_test();
}
```

```
/**
 * @brief test_sos 安全存储测试
 */
void test_sos()
{
printf("Start sos test...\n");
char buffer[32] = "hello world";
    KYSEC_TEE_DATA data;
    data.dptr = buffer;
    data.dsize = strlen(buffer) + 1;
    SOS_ID id;
memset(id, 0x00, 32);
snprintf(id, 32, "%s", "sos:test"); //生成一个安全存储对象id
    KT_Result res = KT_SUCCESS;
    res = kysec_tee_sos_store(id, data); //将 "hello world"存入tee
if (res != KT_SUCCESS) {
printf("sos store failed 0x%.x\n", res);
return;
}
    KYSEC_TEE_DATA fetch_data;
    res = kysec_tee_sos_fetch(id, &fetch_data); //使用id在TEE中查询数据，验证是否写入
if (res != KT_SUCCESS) {
printf("sos fetch 1 failed 0x%.x\n", res);
return;
} else {
printf("sos fetch 1: %s\n", fetch_data.dptr);
free(fetch_data.dptr);
}
char buffer2[32] = "hello kylin";
    data.dptr = buffer2;
    data.dsize = strlen(buffer2) + 1;
    res = kysec_tee_sos_update(id, data); //更新TEE中 id对应的数据为"hello kylin"
if (res != KT_SUCCESS) {
printf("sos update failed 0x%.x\n", res);
return;
}
    res = kysec_tee_sos_fetch(id, &fetch_data); //使用id在TEE中查询数据,验证是否更新成功
if (res != KT_SUCCESS) {
printf("sos fetch 2 failed 0x%.x\n", res);
return;
```

```
} else {
printf("sos fetch 2: %s\n", fetch_data.dptr);
free(fetch_data.dptr);
}
    res = kysec_tee_sos_delete(id); //使用id删除数据
if (res != KT_SUCCESS) {
printf("sos delete failed 0x%.x\n", res);
return;
}
    res = kysec_tee_sos_fetch(id, &fetch_data); //使用id在TEE中查询数据，验证是否删除成功
if (res != KT_SUCCESS) {
printf("sos fetch 3 failed 0x%.x\n", res);
return;
} else {
printf("sos fetch 3: %s\n", fetch_data.dptr);
free(fetch_data.dptr);
}
}
/**
 * @brief test_km 密钥服务SDK接口测试
 */
void test_km()
{
printf("Start km test...\n");
    KM_ID id;
memset(id, 0x00, 32);
snprintf(id, 32, "km:test_%ld", time(0));
uint8_t pub_key[64] = {0};
    KT_Result res = KT_SUCCESS;
    res = kysec_tee_km_sm2_keygen(id, pub_key); //生成公私钥对
if (res != KT_SUCCESS) {
printf("km sm2 gen failed 0x%.x\n", res);
return;
} else {
printf("sm2 pub key: ");
for (int i = 0; i < 64; i++) {
printf("0x%.2x ", pub_key[i]);
}
printf("\n");
}
char buffer[32] = "hello world";
```

```
    KYSEC_TEE_DATA data;
    data.dptr = buffer;
    data.dsize = strlen(buffer) + 1;
uint8_t sign_data[64] = {0};
    res = kysec_tee_km_sm2_sign(id, data, sign_data); //对 "hello world" 签名
if (res != KT_SUCCESS) {
printf("km sm2 sign failed 0x%.x\n", res);
return;
} else {
printf("sm2 sign data: ");
for (int i = 0; i < 64; i++) {
printf("0x%.2x ", sign_data[i]);
}
printf("\n");
}
    res = kysec_tee_km_sm2_verify(id, data, sign_data); //对 "hello world" 验签
if (res != KT_SUCCESS) {
if (res != 1)
printf("km sm2 verify failed 0x%.x\n", res);
else
printf("km sm2 verify no pass\n");
} else {
printf("km sm2 verify pass\n");
}
char buffer2[32] = "hello kylin";
    data.dptr = buffer2;
    data.dsize = strlen(buffer2) + 1;
    res = kysec_tee_km_sm2_verify(id, data, sign_data); //对 "hello kylin" 验签
if (res != KT_SUCCESS) {
if (res != 1)
printf("km sm2 verify failed 0x%.x\n", res);
else
printf("km sm2 verify no pass\n");
} else {
printf("km sm2 verify pass\n");
}
}
int main()
{
test_sos();
test_km();
```

```
test_algo();
return 0;
}
```