

CIU98_B V2 Host SDK[V1.2]– Raspberri4B_Linux 用户使用手册

V1.1



北京中电华大电子设计有限责任公司
CEC Huada Electronic Design Co.,Ltd

2021 年 12 月

声 明

本文档的版权属北京中电华大电子设计有限责任公司所有。任何未经授权对本文档进行复印、印刷和出版发行的行为，都将被视为是对北京中电华大电子设计有限责任公司版权的侵害。北京中电华大电子设计有限责任公司保留对此行为诉诸法律的权力。

北京中电华大电子设计有限责任公司保留未经通知用户对本手册内容进行修改的权利。

本文档并未以暗示、反言或其他形式转让本公司以及任何第三方的专利、商标、版权、所有权等任何权利或许可。本公司不承担因使用、复制、修改、散布等行为导致的任何法律责任。

变更记录

版本	修改描述	日期
V1.0	初稿	2021-7-3
V1.1	1) 修改参考资料中的文档名，添加“V2”、“Raspberrypi4B_Linux”字样； 2) 修改第 4 章，将接口抽象层描述改为协议抽象层描述； 3) 修改第 8 章，修改笔误：“openssl 且其版本为 1.1.d”改为“openssl 且其版本为 1.1.1d”；	2021-12-13

目 录

1	引言	1
2	参考资料	1
3	概述	1
4	整体架构	1
5	目录结构	2
6	基于树莓派的 DEMO 使用说明	3
6.1	基于树莓派的硬件环境搭建	4
6.2	主端安全 SDK 代码移植说明	5
7	API 应用示例说明	5
7.1	源码编译说明	6
7.2	示例演示说明	7
7.3	注意事项	9
8	OPENSSL 引擎集成 SE 密码服务示例说明	10
8.1	MAKEFILE 修改说明	10
8.2	示例演示说明	10
8.3	注意事项	13
9	常用错误码含义	14
附录:	15
	日志打印	15
	生成 CA 证书	15
	1、生成 ECC 私钥	15
	2、生成 CA 请求文件	16
	3、生成 CA 证书	16
	签发证书过程	16
	1、生成 ECC 私钥	16
	2、生成证书请求文件	16
	3、生成证书	16
	4、验证证书链	16
	查看证书	16

1 引言

本文档主要介绍 CIU98_B V2 Host SDK[V1.2]（以下简称主端安全 SDK）的整体架构、目录结构、API 应用示例说明以及相关注意事项，帮助客户快速使用主端安全 SDK。

2 参考资料

《CIU98_B V2 Host SDK[V1.2]-Raspberrypi4B_Linux API 接口说明.chm》V1.1.

《CIU98_B V2 Host SDK[V1.2]-Raspberrypi4B_Linux 用户移植手册.doc》V1.1

3 概述

本主端安全 SDK 包括通信、控制、设备认证、密钥管理、PIN 相关操作、密码运算、文件管理、信息管理和 Loader 升级接口功能。

支持的通信接口为 SPI 和 I2C，支持的算法有 3DES/AES/ECC256/RSA/SM2/SM3/SM4 /SHA/系列。其中 ECC 算法的 ECDSA、ECIES、ECDH 符合 SEC 规范，曲线参数支持 nistp256r1。

本 SDK 代码空间统计如下：

接口	RAM	FLASH
I2C	3 K	105K
SPI	3 K	105K

说明：

1、此空间统计不包含 demo

2、RAM空间仅包含栈空间（此SDK未使用堆），用户可以配置

“CIU98_B_V2_Host_SDK[V1.2.0-release]-Raspberrypi4B_Linux/src/util/util.h”中宏定义“DEQUE_MAX_SIZE”来根据平台具体情况自定义配置所占RAM空间。

4 整体架构

本 SDK 采用分层设计，从上到下依次分为 APP（客户应用）、APIs Layer（应用编程接口）、Command Layer（命令层）、Protocol Abstraction Layer（协议抽象层）、Link Protocol Layer（链路通信协议层）和 Hardware Portable Layer（硬件适配层）6 部分组成，整体架构图如下：

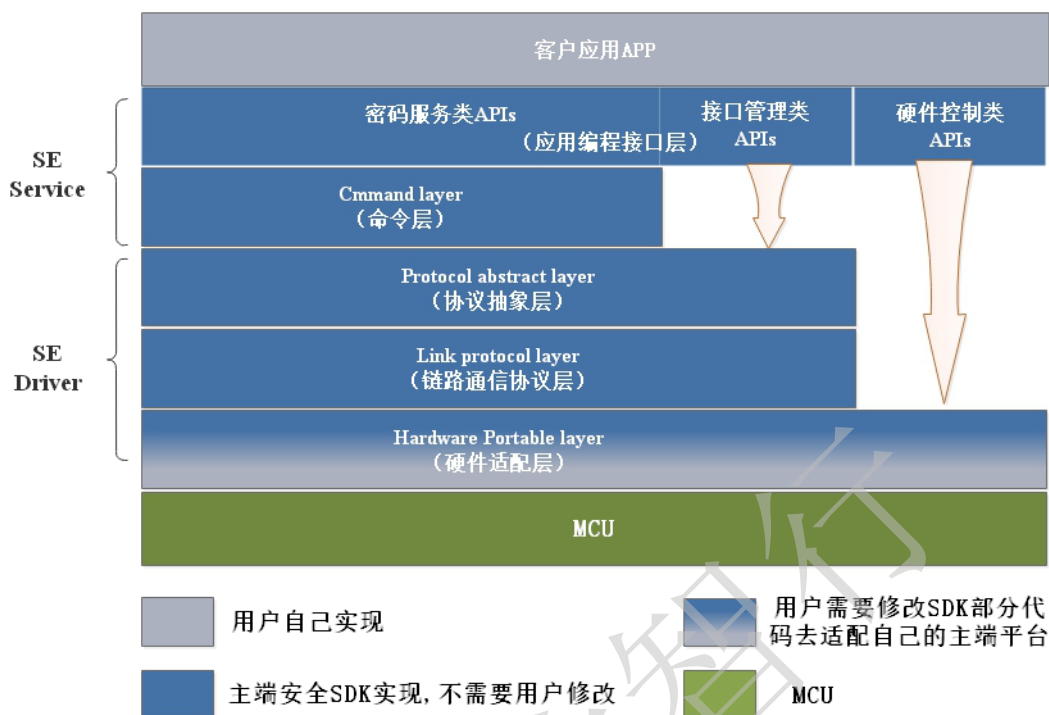


图 1 整体架构图

- ✓ **APIs Layer (应用编程接口层)**: 封装了 SE 对外提供的所有应用功能函数接口，包括通信、控制、鉴权、密钥管理、PIN 操作、文件管理、密码运算等功能函数接口，方便用户开发应用。
- ✓ **Command Layer (命令层)**: 封装与 COS 对应的应用命令，为上层应用功能提供基础命令支持，易于上层应用功能接口开发和移植。
- ✓ **Protocol Abstraction Layer (协议抽象层)**: 定义统一的抽象接口，屏蔽链路通信协议层不同接口协议的实现差异，封装了底层驱动协议实现细节。
- ✓ **Link Protocol Layer (链路通信协议层)**: 实现 HED 各链路通信接口的通信协议。
- ✓ **Hardware Portable Layer (硬件适配层)**: 屏蔽相同硬件接口的不同 MCU 实现细节及差异，提供统一硬件访问接口，并依据所定义的函数接口实现在不同 MCU 环境下的硬件功能适配和移植

5 目录结构

主端安全 SDK 为标准 C 语言编写的应用接口源码。

主端安全 SDK 的源码目录结构和各目录含义如图 2、图 3 所示：

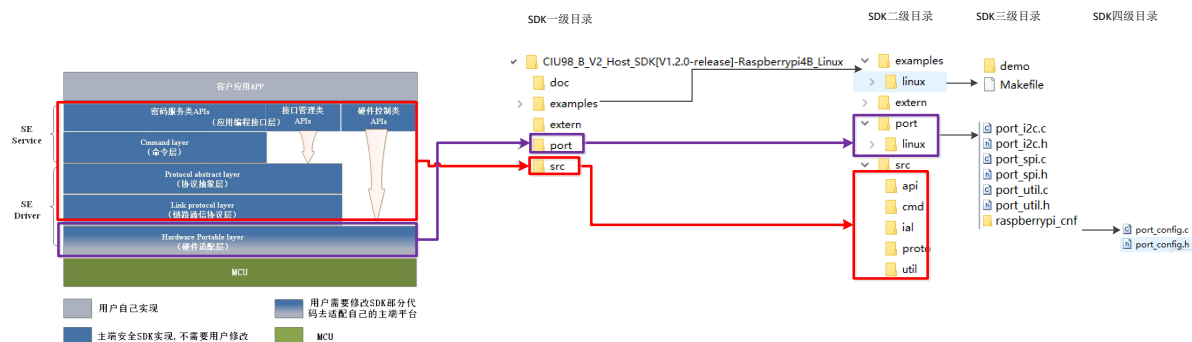


图 2 主端安全 SDK 目录结构图

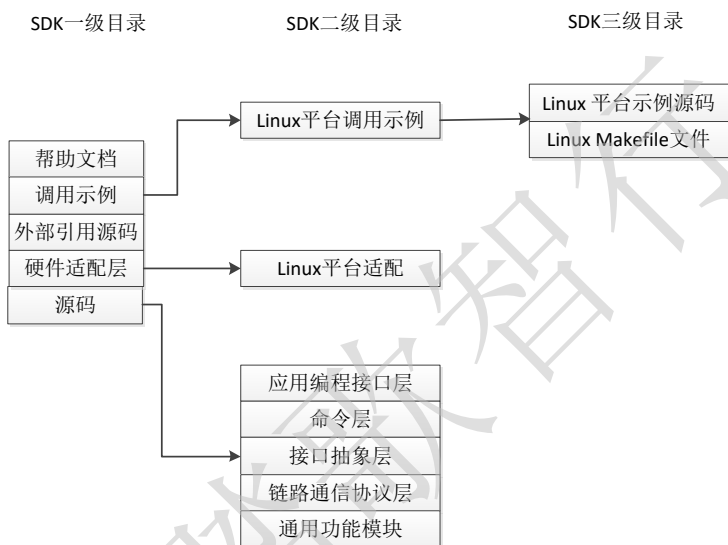


图 3 主端安全 SDK 目录结构含义

客户需要关注的文件夹为 src/api、port 和 examples，其中：
src/api 为应用编程接口，为上层应用程序调用的接口，具体参考《CIU98_B V2 Host SDK[V1.2]-Raspberrypi4B_Linux API 接口说明.chm》。

examples 为 demo 示例，为 app 中的接口调用例程，具体参考本文档“7 API 应用示例说明”。

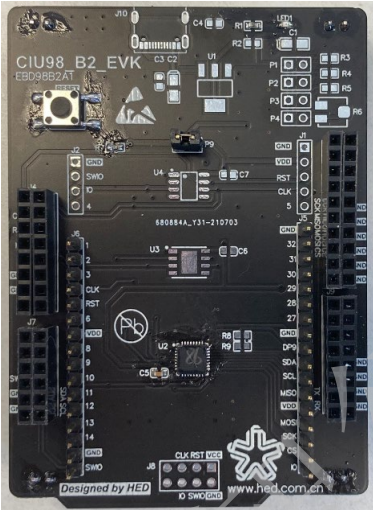
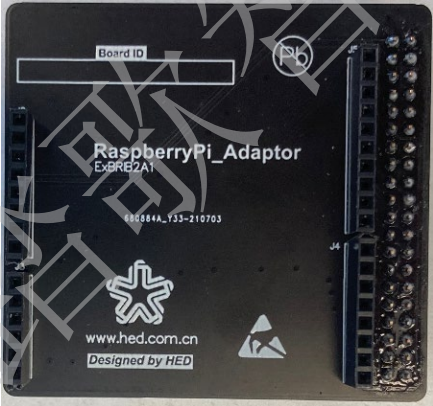
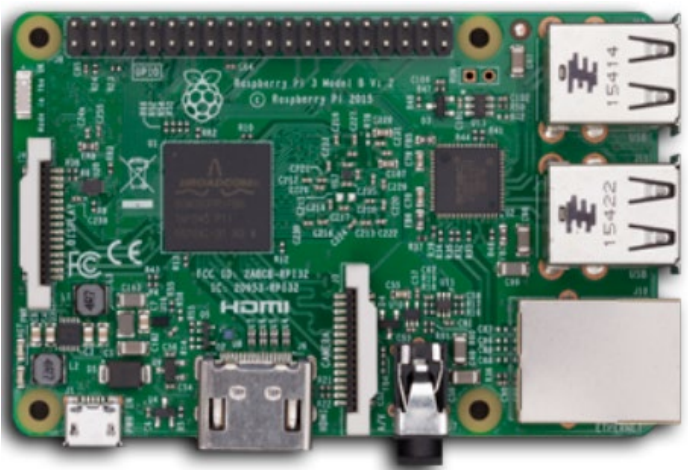
port 为硬件适配层，为 Linux 操作系统硬件平台（树莓派 4B）的硬件适配层示例，客户可根据不同硬件平台进行修改，具体接口参考《CIU98_B V2 Host SDK[V1.2]-Raspberrypi4B_Linux API 接口说明.chm》，移植过程参考文档《CIU98_B V2 Host SDK[V1.2]-Raspberrypi4B_Linux 用户移植手册 V1.1》。

6 基于树莓派的 Demo 使用说明

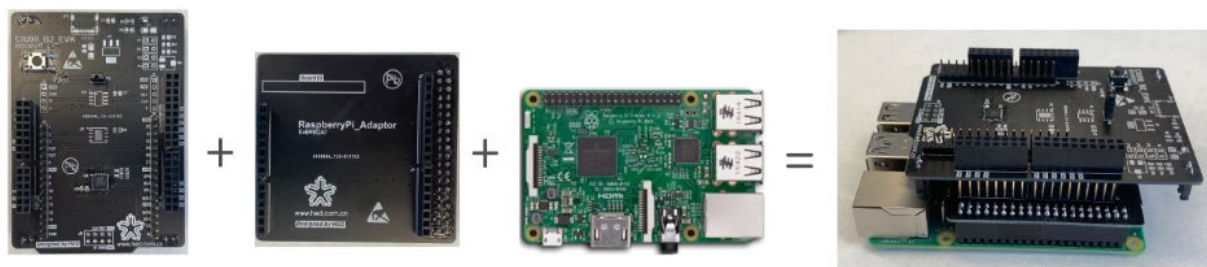
在使用基于树莓派硬件平台的 demo 时，首先完成硬件连接环境的搭建，其次完成主端安全 SDK 代码的移植，具体过程参考 6.1 和 6.2。

6.1 基于树莓派的硬件环境搭建

所需硬件如下表所示：

序号	硬件电路	图片
1	CIU98_B2 开发板	
2	树莓派适配板	
3	树莓派	

硬件电路连接如下图所示：



6.2 主端安全 SDK 代码移植说明

参考《CIU98_B V2 Host SDK[V1.2]-Raspberrypi4B_Linux 用户移植手册 V1.1》完成代码移植

7 API 应用示例说明

在获得主端安全 SDK 代码之后参考《CIU98_B V2 Host SDK[V1.2]-Raspberrypi4B_Linux 用户移植手册 V1.1》完成代码移植，并且依照“6 基于树莓派的 Demo 环境使用说明”搭建硬件环境，即可参照本部分介绍的“API 应用示例说明”调用主端安全 SDK 提供的接口，具体的接口使用说明参考《CIU98_B V2 Host SDK[V1.2]-Raspberrypi4B_Linux API 接口说明.chm》。

在树莓派 linux 平台上以指令方式提供了各个功能的演示示例，其源码路径为：examples/linux/demo/cmd_api_test。

同时也提供了以 C 源码方式调用 API 的示例，其源码路径为：examples/linux/demo/all_api_test。

主端安全 SDK 接口使用调用流程如下：

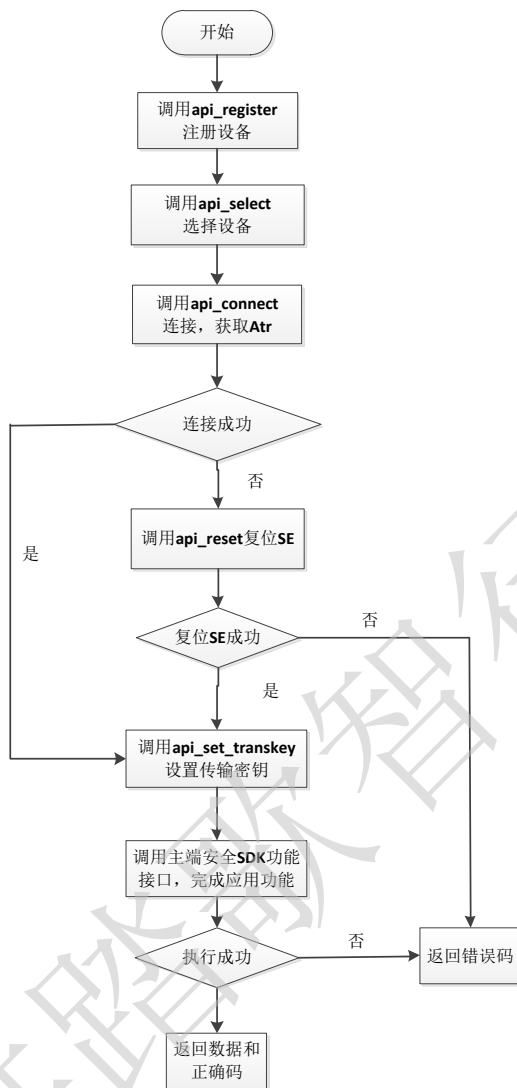


图 4 主端安全 SDK 接口使用调用流程

7.1 源码编译说明

提供的演示示例 Makefile 文件所在路径为：[examples/linux](#)。首先按照以下步骤进行编译：

1、进入主端安全 SDK 的主目录

例如将主端安全 SDK 放在家目录下并命名为 CIU98_B_Host_SDK[V1.2]。执行如下命令：

```
cd ~/CIU98_B_Host_SDK[V1.2]
```

2、进入 Makefile 文件所在的目录

执行如下命令：

```
cd examples/linux/
```

3、执行 make 指令完成编译

执行如下命令：

```
make
sudo make install
```

若重新编译和装载需要执行如下指令：

```
sudo make uninstall
sudo make clean
```

最终编译成功之后会在 [examples/linux/demo/cmd_api_test](#) 中生成可执行文件，如下图所示：

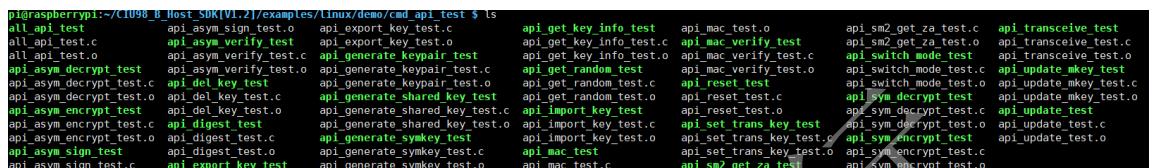


图 5 生成可执行文件

7.2 示例演示说明

- 1、首先进入如下路径：~/CIU98_B_Host_SDK[V1.2]/examples/linux/demo/cmd_api_test
执行命令：

```
cd ~/CIU98_B_Host_SDK[V1.2]/examples/linux/demo/cmd_api_test
```

- 2、如果第一次运行主端安全 SDK。可通过执行如下命令来检查是否通信正常：

```
./api_reset_test
```

若通信正常，其输出结果如下：

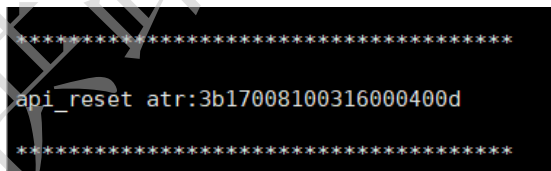


图 6 api_reset_test 运行结果

- 3、以 api_sym_encrypt_test 为例，首先通过查看相应示例指令的帮助，确定应该输入的参数。

输入如下指令查看帮助：

```
./api_sym_encrypt_test -h
```

输出结果如下：

```

pi@raspberrypi:~/CIU98_B_Host_SDK[V1.2]/examples/linux/demo/cmd_api_test $ ./api_sym_encrypt_test -h

I - demo/cmd_api_test/api_sym_encrypt_test.c(100) - main: test begin
./api_sym_encrypt_test: invalid option -- 'h'

Help menu: api_sym_encrypt_test <option> ...<option>
option:-
-a <alg> : alg type: ALG_AES128 : 0x00
           ALG_DES128 : 0x00
           ALG_SM4 : 0x40
-k <kid> : The kid for the sym encrypt
-l <input data length> : The input data length
-i <input data file> : The input data file
-m <sym encrypt mode> : sym_mode: SYM_MODE_CBC : 0x00
                       SYM_MODE_ECB : 0x01
-p <padding type> : padding_type: PADDING_NOPADDING : 0x00
                  padding_type: PADDING_PKCS7 : 0x03
-v <iv value> : please input the iv value
-h : Print this help

```

图 7 api_sym_encrypt_test 运行帮助

按照以上输入参数格式组装指令如下：

./api_sym_encrypt_test -a 0x40 -k 0x06 -l 32 -i sym_encrypt_input_32_sm4.txt -m 0x01 -p 0x00

输出结果如下图所示：

```

*****
api_sym_encrypt in_buf:
1122334455667788112233445566778811223344556677881122334455667788
the alg is:ALG_SM4
the sym_mode is : SYM_MODE_ECB
the output encrypted data length:32
the output encrypted data :
9e7c1591872956648e9664d994ca7ce99e7c1591872956648e9664d994ca7ce9
api_sym_encrypt successfully!
*****

```

图 8 api_sym_encrypt_test 运行结果

4、以 api_asym_decrypt_test 为例，首先通过查看相应示例指令的帮助，确定应该输入的参数。

输入如下指令查看帮助：

./api_asym_decrypt_test -h

输出结果如下：

```
pi@raspberrypi:~/CIU98_B_Host_SDK[V1.2]/examples/linux/demo/cmd_api_test $ ./api_asym_decrypt_test -h
I - demo/cmd_api_test/api_asym_decrypt_test.c(97) - main: test begin
./api_asym_decrypt_test: invalid option -- 'h'

Help menu: api_asym_decrypt <option> ...<option>
option:-
-a <alg>  : alg type: ALG_SM2      : 0x50
           ALG_RSA1024 : 0x31
           ALG_RSA2048 : 0x33
           ALG_ECC256_NIST : 0x70
-k <kid>   : The kid for the asym encrypt
-l <input data length> : The input data length
-i <input data file>   : The input data file
-p <pading type just for rsa> : pading_type: PADDING_NOPADDING : 0x00
                               : pading_type: PADDING_PKCS1 : 0x04
-h         : Print this help
```

图 9 api asym decrypt test 运行帮助

按照以上输入参数格式组装指令如下:

```
./api asym decrypt test -a 0x50 -k 0x0F -l 346 -i asym_decrypt input 346 sm2.txt
```

输出结果如下图所示:

```

*****
api_asm decrypt in buf:
3ae3654d9d284922af73203a4cd9c5aa8483bd6f0b4d16a29971ff516c8f13d91b826d2b7186fb2e522390a93a61cfff5e1401ae6027653a0f81160b91b1fff896cef3059e89811761
80e170e27cda654c9b07f04455679ee327b74809476527786589e987e170983d19b2c06383240b474ad10a3a889f4c30b0450241a95f4b78200cc58a52c6b5cbca09b944550e92a1367
f3a68c869a2c659c32b13c5e613833a864e8f77f08318f9e5b9b134831684a2df3a10ec4f7e303b1c79dd781189cdf8a9e162230b08fb6d3cd482c38ff7230b721ab4b4d9b50345f
3b03c8322c01b1f33833e5b61f2c6a827de941f3e816163d3df579e4fe04e26ff16795484eeaf928795737f883f8c399a331605d289837e2370b3347f94fc428fe52fe63789f819378a737
e0d088f45f44ca9fe176ef39de1f510a53baa68835af0f8dd514dd98e41d2db13066da4d6bb5f9c9b53c858d38e87c7f5f
the alg is:ALG_SM2

the kid is:0x10

the output decrypted data length:250
the output decrypted data :
1122334455667788990011223344556677889900112233445566778899001122112233445566778899001122334455667788990011223344556677889900112211223344556677889900112
122334455667788990011223344556677889900112211223344556677889900112233445566778899001122334455667788990011221122334455667788990011223344556677889900112
2112233445566778899001122334455667788990011223344556677889900112211223344556677889900112233445566778899001122334455667788990011223344556677889900112
api_asm_decrypt successfully!
*****

```

图 10 api asym decrypt test 运行结果

7.3 注意事项

- 1、 示例源码都是以 SPI 接口为例实现的，若需要改变为 I2C 接口修改说明如下：

```
//-- 1. 注册及选择设备 ----
ret = api_register(PERIPHERAL_SPI, SPI_PERIPHERAL_SE0);
if(ret!=SE_SUCCESS)
{
    LOGE("E=1");
    ret = api_register(PERIPHERAL_I2C, I2C_PERIPHERAL_SE0);
}

//-- 2. 连接，获取atr ----
ret = api_select(PERIPHERAL_SPI, SPI_PERIPHERAL_SE0);
if(ret!=SE_SUCCESS)
{
    LOGE("E=2");
    ret = api_select(PERIPHERAL_I2C, I2C_PERIPHERAL_SE0);
}
```

- 2、以 C 源码方式调用 API 的示例，其可执行文件路径为：
examples/linux/demo/cmd api test。文件名为：all api test

在上述目录下运行如下指令即可执行:

./ all_api_test

8 openssl 引擎集成 SE 密码服务示例说明

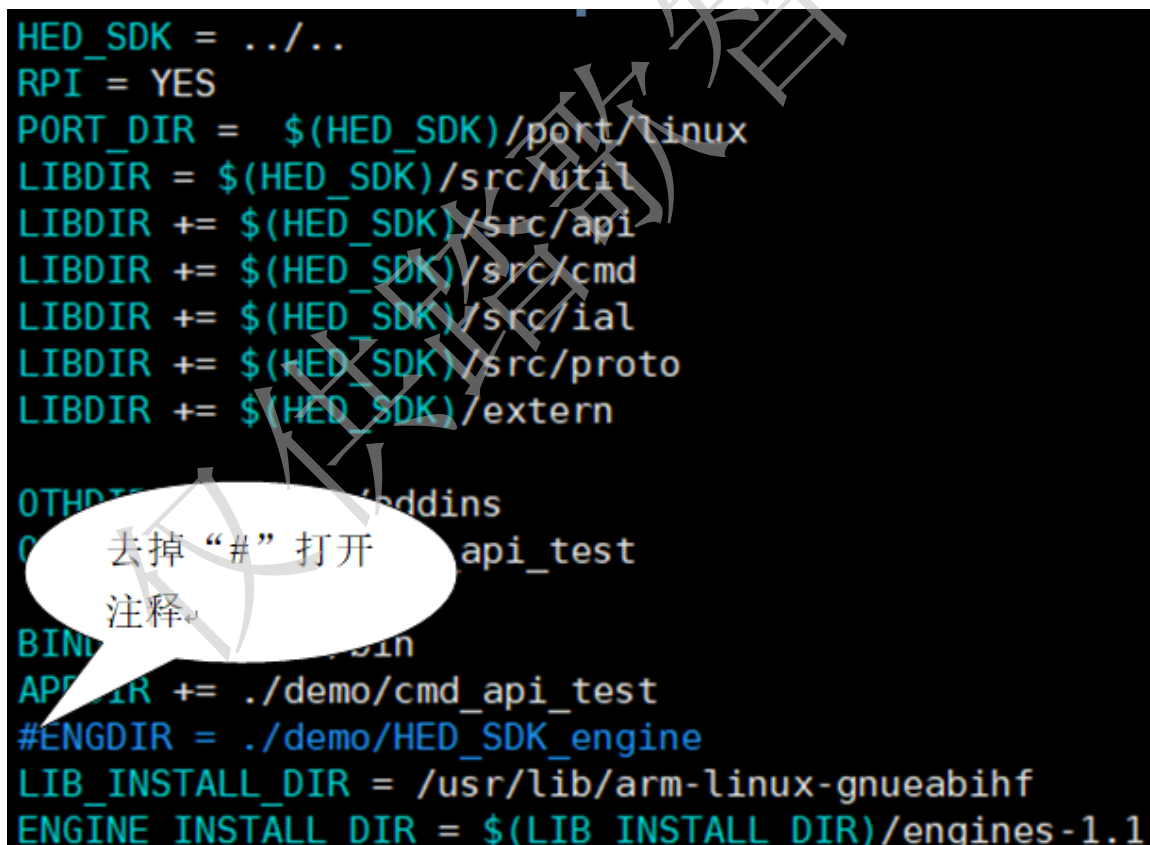
在 demo 中提供了 openssl 引擎集成 SE 密码服务的示例，其 openssl 引擎适配代码路径为：[examples/linux/demo/HED_SDK_engine](#)，如若无 openssl 集成 SE 密码服务的需求请忽略此章节。该引擎适配代码适配了 openssl 的 ECDSA(ECC256)、ECDH(ECC256)、load private key、generate ecc key(ECC256)。在 demo 中同时提供了演示 openssl 引擎机制的服务端和客户端代码（双端通过 TLS 协议进行通信），其路径分别为：

[examples/linux/demo/cmd_api_test/server_test.c](#)

[examples/linux/demo/cmd_api_test/client_test.c](#)

8.1 makefile 修改说明

如若需要演示 openssl 引擎机制的 demo，执行“7.1 源码编译说明”make 编译源码之前请预先修改 makefile。修改方式如下图所示：



```
HED_SDK = ../..
RPI = YES
PORT_DIR = $(HED_SDK)/port/linux
LIBDIR = $(HED_SDK)/src/util
LIBDIR += $(HED_SDK)/src/api
LIBDIR += $(HED_SDK)/src/cmd
LIBDIR += $(HED_SDK)/src/ial
LIBDIR += $(HED_SDK)/src/proto
LIBDIR += $(HED_SDK)/extern

OTHER_DIRS += $(HED_SDK)/src/headers
OTHER_DIRS += $(HED_SDK)/src/api_test

BIN_DIR = ./bin
APP_DIR += ./demo/cmd_api_test
#ENGINE_DIR = ./demo/HED_SDK_engine
LIB_INSTALL_DIR = /usr/lib/arm-linux-gnueabi/hf
ENGINE_INSTALL_DIR = $(LIB_INSTALL_DIR)/engines-1.1
```

8.2 示例演示说明

- 1、演示该示例之前请确保已经安装 **openssl** 且其版本为 **1.1.1d** 以上。
- 2、向 SE 导入用于演示的服务端私钥，其过程如下：

进入如下路径：[examples/linux/demo/cmd_api_test](#)，执行如下命令：

`./api_import_key_test -a 0x70 -k 0x11 -l 32 -i import_key_ECC_sever.txt -t 0x00`

其执行结果如下：

```
I - demo/cmd_api_test/api_import_key_test.c(103) - main: test begin

I - ../../src/proto/proto_spi.c(936) - proto_spi_init: Success!
I - ../../src/proto/proto_spi.c(1132) - proto_spi_open: Open Periph Success!
I - ../../src/proto/proto_spi.c(1262) - proto_spi_transceive: Communication Success!
I - ../../src/proto/proto_spi.c(1262) - proto_spi_transceive: Communication Success!
I - ../../src/proto/proto_spi.c(1262) - proto_spi_transceive: Communication Success!
I - ../../src/proto/proto_spi.c(1262) - proto_spi_transceive: Communication Success!
I - ../../src/proto/proto_spi.c(1262) - proto_spi_transceive: Communication Success!
I - ../../src/proto/proto_spi.c(1262) - proto_spi_transceive: Communication Success!

*****
the alg is:ALG_ECC256_NIST
the import key process protected by trans_key
the input key data length:32
the input key data :
725072f29501579fd8af1435241744fe1023ba1bf1ab91d28ff9882a6dc3d4cc
api_import_key successfully!

*****

I - ../../src/proto/proto_spi.c(991) - proto_spi_deinit: Success!
I - demo/cmd_api_test/api_import_key_test.c(306) - main: test end
```

3、运行服务端和客户端的测试。

首先进入如下路径：`~/CIU98_B_Host_SDK[V1.2]/examples/linux/demo/cmd_api_test`
执行命令：

`cd ~/CIU98_B_Host_SDK[V1.2]/examples/linux/demo/cmd_api_test`

在一个进程中执行服务端演示 demo，执行如下指令进入监听状态：

`./server_test`

其显示结果如下图所示：

```
pi@raspberrypi:~/N1901_Host_SDK[V1.2]/examples/linux/demo/cmd_api_test $ ./server_test
+++++
Listening to incoming connection
```

在另一个进程中执行客户端 demo，执行如下命令：

`./client_test`

其显示结果如下图所示：

```
pi@raspberrypi:~/N1901_Host_SDK_[V1.2]/examples/linux/demo/cmd_api_test $ ./client_test
*****
s_ipaddr : 127.0.0.1

Connecting to server ....

Connected to 127.0.0.1, port :0x8813
Connected with (NONE) encryption
Performing Handshaking .....
The cipher suite is: TLS_AES_256_GCM_SHA384
The SSL/TLS version is : TLSv1.3
From Server [1459] : 001
From Server [1459] : 002
From Server [1459] : 003
From Server [1459] : 004
From Server [1459] : 005
From Server [1459] : 006
From Server [1459] : 007
From Server [1459] : 008
From Server [1459] : 009
From Server [1459] : 010
From Server [1459] : 011
```

此时运行 ./server_test 的进程，其显示结果如下：

```
pi@raspberrypi:~/N1901_Host_SDK_[V1.2]/examples/linux/demo/cmd_api_test $ ./server_test
*****
Listening to incoming connection
Connection from 127.0.0.1, port :0x7283
Listening to incoming connection
Connection from 127.0.0.1, port :0x7283Engine 0x7eee70 init

I - ../../src/proto/proto_spi.c(936) - proto_spi_init: Success!
I - ../../src/proto/proto_spi.c(1132) - proto_spi_open: Open Periph Success!
I - ../../src/proto/proto_spi.c(1262) - proto_spi_transceive: Communication Success!
I - ../../src/proto/proto_spi.c(1262) - proto_spi_transceive: Communication Success!
I - ../../src/proto/proto_spi.c(1262) - proto_spi_transceive: Communication Success!
hedEngine_init_ec begin*****

hedEngine_init_ec end*****
Engine ID : hed_engine
hed engine init Ok
hed engine setting Ok

hed_loadKey begin*****
key_id=<0x11:serverpub.pem>
cb_data=0x<0>

value 0x11

KID : 0x11
filename : serverpub.pem
len: 91Load Certificate ok
Private Key Match the Server Certificate.
Load CA cert ok
waiting for the client connection!
hed keygen begin *****
the tmp pubkey kid is: 0xf0
the tmp prikey kid is: 0xf1

I - ../../src/proto/proto_spi.c(1262) - proto_spi_transceive: Communication Success!hed keygen end *****

hed engine ECDH begin *****

I - ../../src/proto/proto_spi.c(1262) - proto_spi_transceive: Communication Success!hed engine ECDH end *****

hed engine ECC sign begin *****
the sign kid is: 0x11
the dgst len: 32
I - ../../src/proto/proto_spi.c(1262) - proto_spi_transceive: Communication Success!
api asym_sign sig:
3046022100b3d36f4a86486056a2fef4da7eaabe4b7c9ffd3c2a6b4dcb918cab556620f32022100805c28b873f5055d570d603aed3a2d16e92aad2abbef7026c6c4250f50896989
hed engine ECC sign end *****
The cipher suite is : TLS_AES_256_GCM_SHA384
The SSL/TLS version is : TLSv1.3
[1459] the data from client is : 1
[1459] the data from client is : 2
[1459] the data from client is : 3
[1459] the data from client is : 4
[1459] the data from client is : 5
[1459] the data from client is : 6
```


8.3 注意事项

- 1、 为方便演示服务端和客户端通过 TLS 进行安全通信，我们已经提前生成了：CA 证书 (ca.crt)、CA 私钥 (capri.key)、服务端证书 (server.crt)、服务端私钥 (serverpri.key)、服务端公钥 (serverpub.pem)，用户可对上述文件进行修改，具体过程可参考附录。如若修改了上述文件请注意以下几点：

- 若服务端私钥(serverpri.key)发生了变化，请修改 import_key_ECC_sever.txt 中的私钥数据。
- 若服务端私钥 (serverpri.key) 发生了变化，使用 openssl 指令生成新的证书请求，具体如下所示：

```
openssl req -keyform engine -engine hed_engine -key 0x11:serverpub.pem
-new -out sever11.req -config /usr/local/openssl/openssl.cnf
```

- 2、 安全存储私钥的 kid 暂时选取了：0x11，如需修改该 kid，请修改文件：[examples/linux/demo/cmd_api_test/server_test.c](#)

其具体修改过程如下图所示：

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <time.h>
#include <openssl/crypto.h>
#include <openssl/ssl.h>
#include <openssl/err.h>
#include <openssl/engine.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

// 引擎名字
#define ENGINE_NAME "hed_
//写入ECC私钥KID+公钥文件
#define ECCPRI_PUB "0x11:serverpub.pem"

//服务器端证书文件
#define SEVER_CERT_FILE "server.crt"
//CA证书
#define CA_CERT_FILE "ca.crt"
// 设置端口IP
#define DEFAULT_IP "127.0.0.1"
#define DEFAULT_PORT 5000
// 设置安全传输模式
#define SECURE_COMM TLS_server_method()
//#define SECURE_COMM DTLS_server_method()
```

同时导入私钥时的 kid 也要随之修改，如下指令的标红部分：

```
./api_import_key_test -a 0x70 -k 0x11 -l 32 -i import_key_ECC_sever.txt -t 0x00
```

- 3、在 ECDH 过程中生成的临时公私钥暂时选取的 kid 为：0xF0 和 0xF1，如需修改该 kid，请修改文件：[examples/linux/demo/HED_SDK_engine/hed_engine.h](#) 其具体修改过程如下图所示：

```

//Macro define
/// Definition for false
#ifndef FALSE
#define FALSE (0U)
#endif

/// Definition for true
#ifndef TRUE
#define TRUE (1U)
#endif

#define HED_ENGINE_SUCCESS 1
#define HED_ENGINE_FAIL 0

#define EVP_SUCCESS
#define EVP_FAIL

#define PUBKEYFILE_SIZE
#define PUBKEY_SIZE

//设置用于ECDHE的临时公私钥KID
#define TMPKID_PUB 0xF0
#define TMPKID_PRI 0xF1
    
```

修改如下两个临时密钥 kid (0xF0~0xFF)。

9 常用错误码含义

错误码	含义
0x30000002	连接超时，请检查硬件是否正确连接，适配层是否正确适配。
0x30000007	通信错误，请尝试对 SE 重新上电。
0x100063Cx	验证 pin 错，请检查 pin 数据是否正确。
0x10006985	使用条件不满足，请检查是否在执行该操作前进行了安全认证，例如：pin

	认证或者设备认证。
0x10006A84	空间不足，请检查数据长度是否超过了文件支持的最大长度。
0x10006A88	kid 不存在，请检查 kid 是否存在。
0x10006582	SE 程序跑飞，请检查输入数据的正确性。
0x20000001	输入参数错，检查输入参数是否合法。

附录：

日志打印

如下图所示为出现错误所打印的日志：

```
E - demo/cmd_api_test/api_digest_test.c(171) - main: failed to api_verify_pin
```

上图中的日志报错，指明对应文件为 api_digest_test.c，错误行数为 171 行，错误函数为 api_verify_pin。若需要看返回错误码，只需要按 printf 的语法规则，将出错函数的返回值 ret 打印出来即可。

例：printf(“%04x”,ret);

生成 CA 证书

首先在生成 ca 证书的目录下新建文件 ca.srl（写入 01）和 index.txt。

1、生成 ECC 私钥

openssl ecparam -out capri.key -name prime256v1 -genkey

openssl pkey -in serverpri.key -text 查看 ECC 公私钥的二进制形式

```
pi@raspberrypi:~/CIU98_B_Host_SDK[1.2.0-release]-Raspberrypi4B_Linux/
-----BEGIN PRIVATE KEY-----
MIGHAgEAMBMGBYqGSM49AgEGCCqGSM49AwEHBG0wawIBAQQgc1By8pUBV5/YrxQ1
JBdE/hAjuhvXq5HSj/mIKm3D1MyhRANCAAQYn9p4rxj3KbBhRW/+jwvuzzwjZebF
Q95GdBMnaSqo2BMaE+01cJl6CA9YYYQG8KWRveoJQ96Uv7mJ26zAwfey
-----END PRIVATE KEY-----
Private-Key: (256 bit)
priv:
  72:50:72:f2:95:01:57:9f:d8:af:14:35:24:17:44:
  fe:10:23:ba:1b:f1:ab:91:d2:8f:f9:88:2a:6d:c3:
  d4:cc
pub:
  04:18:9f:da:78:af:18:f7:29:b0:61:45:6f:fe:8f:
  0b:ee:cf:3c:23:65:e6:c5:43:de:46:74:19:a7:69:
  2a:a8:d8:13:1a:13:e3:b5:70:99:7a:08:0f:58:61:
  84:06:f0:a5:91:bd:ea:09:43:de:94:bf:b9:89:db:
  ac:c0:c1:f7:b2
ASN1 OID: prime256v1
NIST CURVE: P-256
pi@raspberrypi:~/CIU98_B_Host_SDK[1.2.0-release]-Raspberrypi4B_Linux/
```

2、生成 CA 请求文件

```
openssl req -key capri.key -new -out cat.req -config /usr/local/openssl/openssl.cnf
```

输入 ca 证书信息：国家 公司 部门 人名 邮箱 等等

3、生成 CA 证书

```
openssl x509 -req -in cat.req -signkey capri.key -out ca.crt -days 365
```

签发证书过程

1、生成 ECC 私钥

```
openssl ecparam -out serverpri.key -name prime256v1 -genkey
```

提出对应公钥：

```
openssl pkey -outform PEM -in serverpri.key -pubkey -out serverpub.pem
```

2、生成证书请求文件

```
openssl req -keyform engine -engine hed_engine -key 0x11:serverpub.pem -new -out sever11.req  
-config /usr/local/openssl/openssl.cnf
```

输入证书信息：国家 公司 部门 人名 邮箱 等等（不能和 ca 请求的一样）

3、生成证书

```
openssl x509 -req -in server.req -CA ca.crt -CAkey capri.key -out server.crt -days 10000
```

从证书中提出公钥方法如下：

```
openssl x509 -outform PEM -in 证书文件 -pubkey -out 公钥文件
```

4、验证证书链

```
openssl verify -CAfile ca.crt -show_chain server.crt
```

查看证书

查看证书二进制方法如下：

```
openssl x509 -in 证书 -text
```