

TH1520_Linux_安全子系统_用户手册

[版本历史](#)

[缩写和术语](#)

[概述](#)

[系统架构](#)

[功能特性](#)

[安全资源](#)

[功能列表](#)

[平台功能](#)

[RoT密钥管理](#)

[安全启动](#)

[安全升级](#)

[安全隔离](#)

[安全计算](#)

[安全存储](#)

[REE FS存储](#)

[eFuse数据存储](#)

[csi_efuse_read](#)

[csi_efuse_write](#)

[efuse配置自检](#)

[安全工具](#)

[Fastboot](#)

[Product](#)

[Sectool](#)

[UBOOT镜像签名](#)

[TF镜像签名](#)

[TEE镜像签名](#)

[内核镜像签名](#)

[安全应用开发](#)

开发接口

TEE Internal APIs

Trusted Core Framework API

Trusted Storage API

Cryptographic operation API

Timer API

TEE Client APIs

C Lib APIs

TA开发SDK

SDK介绍

SDK开发

CA开发

代码开发

MK脚本修改

代码编译

TA 开发

编码开发

TA API

定义TA属性

MK脚本修改

代码编译

编译选项

默认编译选项

增加编译选项

TA/CA部署

运行验证

参考文档

版本历史

版本	说明	作者	日期
----	----	----	----

v1.0	• 初始版本	T-Head	20230303
------	--------	--------	----------

缩写和术语

名称	定义
TEE	Trusted Execution Environment
REE	Regular Execution Environment
TF	Trust Firmware
TA	Trusted Application
CA	Client Application
XT ² EE	Xuan Tie TEE
GP	Global Platform
AES	Advanced Encryption Standard
API	Application Programming Interface
ID	Identifier
JTAG	Joint Test Action Group
ROM	Read Only Memory
RAM	Random Access Memory
RoT	Root of Trust
TLS	Transport Layer Security
SSL	Secure Sockets Layer
TOE	Target of Evaluation
AES	Advanced Encryption Standard
RSA	Rivest / Shamir / Adleman asymmetric algorithm
SoC	System-on-Chip

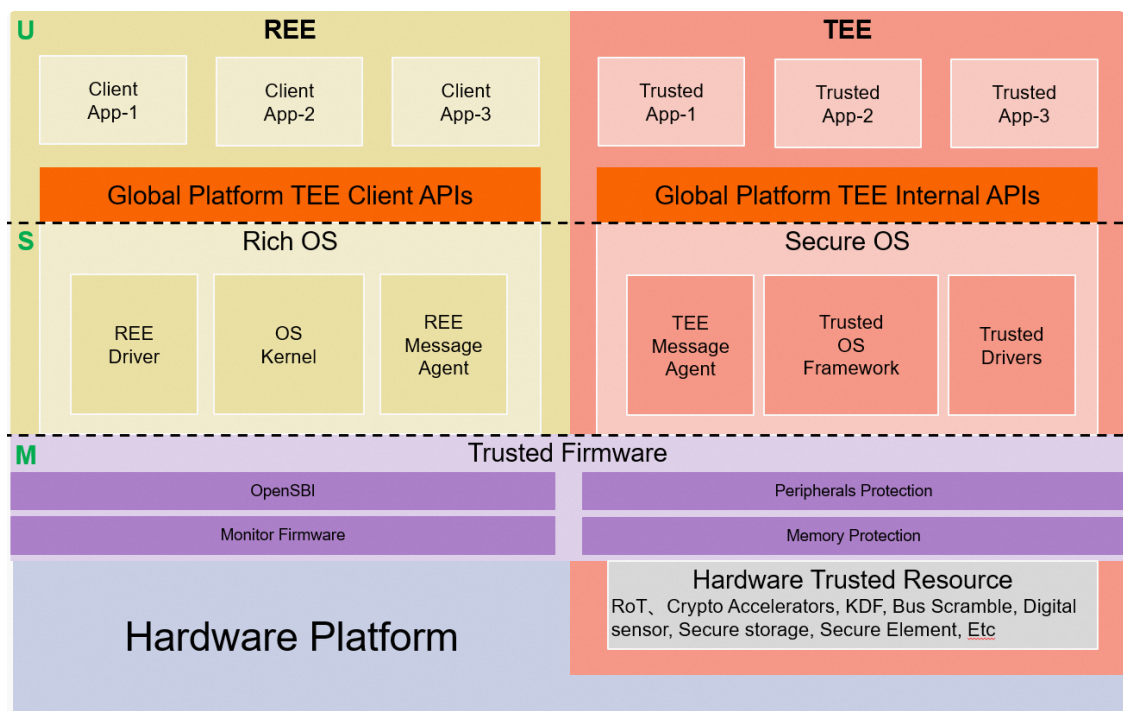
概述

平头哥玄铁RISV-C架构的Virtual Zone技术可以将系统分为两个域：安全域和非安全域。基于内存的划分，我们可以把内存DRAM分为安全内存和非安全内存，运行在安全内存里的代码称为安全代码，运行在非安全内存里的代码称为非安全代码。除了内存DRAM外，进一步可以将SRAM、外设、中断等相关硬件资源划分为安全资源和非安全资源。安全资源可以被安全代码访问执行，但不能被非安全代码访问。

系统架构

XT²EE是基于Virtual Zone技术实现的安全TEE子系统。通过硬件提供的强隔离机制，整个系统运行在安全域环境里，为非安全域的物联网应用提供可信的安全服务。根据具体的物联网安全需求，基于硬件提供的安全基础，TH1520将XT²EE与安全硬件相结合，整个系统中的软件，CPU，内存和外围设备都具备安全性，有了可信赖的外围设备，保证了整个设备生命周期的高安全性。

图1-1 TH1520安全子系统架构



功能特性

XT²EE的具备的功能特性：

- 支持平头哥RISC-V架构的Virtual Zone技术，支持RISC-V三种安全权限工作模式
- 支持国际Global Platform 标准安全接口，符合GP TEE Protection Profile规范
- 支持原生OPTEE系统，兼容Linaro TEE 生态，提供安全应用开发SDK

- 支持Linux Kernel TEE 软件和TEE驱动
- 支持Linux supplicant daemon，提供远程服务调用
- 支持test suite(xtest)测试程序，可评估TEE基础功能

安全资源

XT²EE安全子系统会将使用到的内存、外设IO，设备中断等资源配置为安全属性，利用Physical Memory Protection (PMP)和IOPMP隔离机制实现对资源的安全访问，确保非安全代码不能访问安全资源。TH1520生态软件里XT²EE安全子系统将以下资源划分为安全资源。

表1-1 安全资源描述

安全资源	地址范围
Trusted Firmware	0x0000_0000 – 0x001F~FFFF
TEE-OS	0x1C00_0000 – 0x1DFF_FFFF
AUDIO_IOPMP0	0xFF_CB02_E000 – 0xFF_CB02_EFFF
AUDIO_IOPMP1	0xFF_CB02_F000 – 0xFF_CB02_FFFF
IOPMP_SDIO0	0xFF_FC02_9000 – 0xFF_FC02_9FFF
IOPMP_SDIO1	0xFF_FC02_A000 – 0xFF_FC02_AFFF
IOPMP_USB0	0xFF_FC02_E000 – 0xFF_FC02_EFFF
IOPMP_Crypto engine	0xFF_FF22_0000 – 0xFF_FF25_FFFF
KeyRAM	0xFF_FF26_0000 – 0xFF_FF26_FFFF
Digital_Sensor	0xFF_FF27_0000 – 0xFF_FF27_4FFF
Crypto Engine	0xFF_FF30_0000 – 0xFF_FF33_FFFF

注意：

- 如果有新的安全设备资源，需要联系平头哥协助。

功能列表

XT²EE安全子系统为用户提供了丰富的平台安全能力，比如安全启动、安全升级等。用户可以根据自己的应用场景对使用到的资源进行安全隔离，基于安全平台开发出自己的安全应用，方便部署在XT²EE安全子系统里，从而实现具有安全特性的设备产品。

XT²EE安全子系统支持的平台安全功能如下所示：

表1-2 支持的安全功能列表

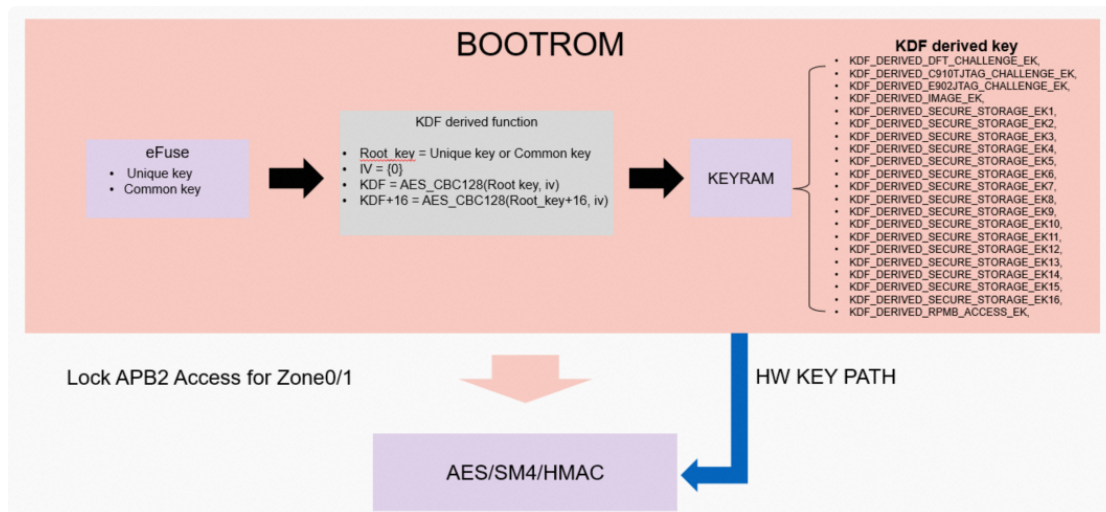
安全功能	说明
RoT密钥管理	支持密钥派生，密钥使用，密钥销毁等key store功能
安全启动	验证平台完整性和真实性，建立启动可信链
安全升级	支持本地升级，包括uboot、TEE、TF镜像
安全隔离	支持三层系统安全隔离，满足国际GP安全标准
安全计算	支持国际、国密算法、随机数， Curve25519等
安全存储	存储数据文件，一文一密，支持完整性验证，文件访问原子操作

平台功能

RoT密钥管理

TH1520提供硬安全可靠的RoT密钥管理机制，从系统上电开始，在芯片硬件BOOTROM里就根据eFuse里的Unique key关键字和Common Key关键字，利用硬件KDF引擎进行密钥派生，然后将派生出来的密钥存在KEYRAM里，最后关闭KEYRAM的CPU访问权限。整个过程没有密钥泄露和被篡改的风险。

图2-4 密钥派生描述



KDF派生出来的密钥有多用途，其中包括：

- 安全调试
- 启动镜像加密
- 安全存储
- 安全RPMB权限访问

注意：

- TH1520生态开发板不支持efuse烧写，请联系平头哥进行协助
- 在efuse没有烧写过的情况下，密钥管理由TEE OS统一管理

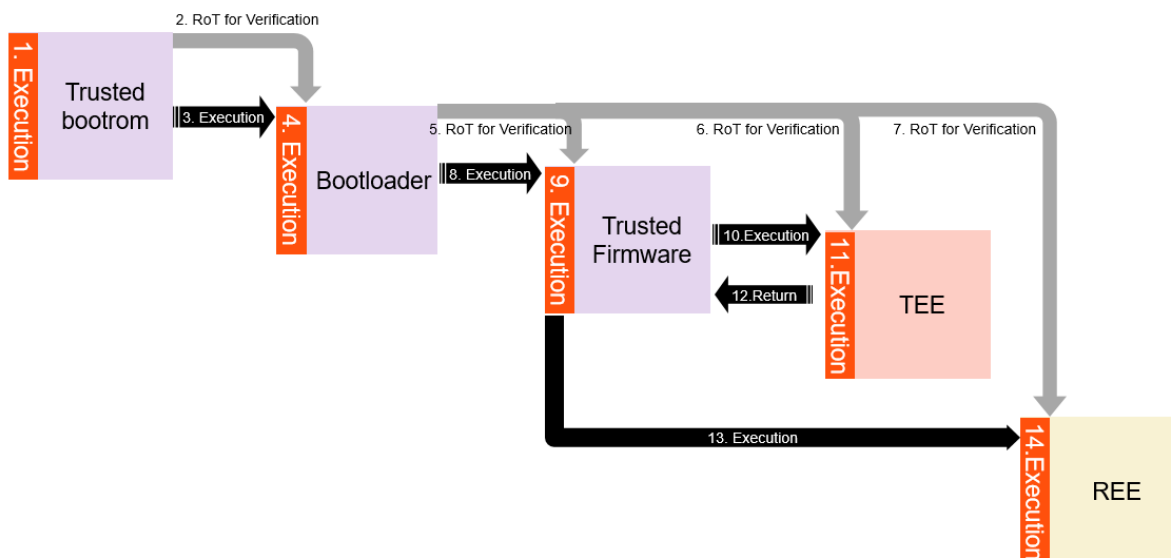
安全启动

安全启动是指启动链路上子镜像要求被父镜像通过证书里的公钥进行验签，只有验签通过后，父镜像才能把执行权交给子镜像，否则，系统会重启。

使能安全启动功能需要依赖以下资源：

- eFuse里的安全功能配置字，包括安全启动使能字段、镜像是否加密字段、镜像签名算法字段等
- 镜像签名工具，对启动镜像进行签名生成签名镜像，这包括uboot、TEE、TF、内核镜像做签名

图2-1 安全启动流程描述



注意：

- 由于TH1520生态开发板没有对eFuse进行安全配置烧写，所以BL0(BootROM)对BL1(Bootloader)无法进行镜像验签，但Bootloader会对Trusted Firmware和TEE镜像进行验签，从而启动TEE系统。
- 如何对镜像签名，具体操作参考3.3章节。

安全升级

安全升级支持OTA升级和离线升级两种。

• OTA升级

OTA 英文全称是Over-the-Air Technology，即空间下载技术，是通过移动通信(GSM或CDMA)的空中接口对SIM卡数据及应用进行远程管理的技术。空中接口可以采用WAP、GPRS、CDMA1X及短消息技术。OTA技术的应用，使得移动通信不仅可以提供语音和数据服务，而且还能提供新业务下载。这种技术一般使用在产品使用阶段。

• 离线升级

离线升级一般是指通过物理通道（比如：串口，USB等）将升级镜像从主机传输到设备中，主机上一般会运行上位机，用于和设备进行数据交互，比如Fastboot等。

注意：

Linux SDK默认支持离线烧写，具体请参考3.1章节。

图2-2 安全升级流程描述



注意：

- 升级包的完整性和真实性由OTA服务商提供
- 升级包里的升级文件的完整性和真实性由OTA方案商提供

安全镜像升级

1. OTA Client 从云端服务器下载安全镜像（TEE.EXT4）
2. OTA Client 负责将安全镜像写入Stash分区。

```

1  1. /* list all partitions defined in uboot under linux */
2  root@light-a-val:~# ls /dev/mmc* -l
3  brw-rw----    1 root    disk      179,    0 Aug  1 10:24 /dev/mmcblk0
4  brw-rw----    1 root    disk      179,    8 Aug  1 10:24 /dev/mmcblk0boot0
5  brw-rw----    1 root    disk      179,   16 Aug  1 10:24 /dev/mmcblk0boot1
6  brw-rw----    1 root    disk      179,    1 Aug  1 10:24 /dev/mmcblk0p1
7  brw-rw----    1 root    disk      179,    2 Aug  1 10:24 /dev/mmcblk0p2
8  brw-rw----    1 root    disk      179,    3 Aug  1 10:24 /dev/mmcblk0p3
9  brw-rw----    1 root    disk      179,    4 Aug  1 10:24 /dev/mmcblk0p4
10 brw-rw----    1 root    disk      179,    5 Aug  1 10:24 /dev/mmcblk0p5
11 brw-rw----    1 root    disk      179,    6 Aug  1 10:24 /dev/mmcblk0p6
12 brw-rw----    1 root    disk      179,    7 Aug  1 10:24 /dev/mmcblk0p7
13 crw-----    1 root    root      246,    0 Aug  1 10:24 /dev/mmcblk0rpm
14 root@light-a-val:~#
15 2. /* mount /dev/mmcblk0p5 to your temp foldr */
16 root@light-a-val:~# mkdir ~/temp
17 root@light-a-val:~# mount -o rw /dev/mmcblk0p5 ~/temp
18 [ 137.974309] EXT4-fs (mmcblk0p5): warning: mounting unchecked fs, runnin
   g e2fsck is recommended
19 [ 137.983914] EXT4-fs (mmcblk0p5): mounted filesystem without journal. Op
   ts: (null)
20 [ 137.991515] ext4 filesystem being mounted at /home/root/temp supports t
   imestamps until 2038 (0x7fffffff)
21 root@light-a-val:~# cp ota.bin ~/temp

```

注意:

- 如果mount失败, 就先格式化分区 `mkfs.ext4 /dev/mmcblk0p5`

3. OTA Client 通过Shell命令 `fw_printenv`和`fw_setenv` 设置 uboot的env环境变量 `sec_upgrade_mode`来升级不同的安全镜像:

- 当为0x5a5aa5a5, 升级TEE镜像
- 当为0x5555aaaa, 升级TF镜像

```

1  root@light-a-val:~# fw_setenv sec_upgrade_mode 0x5a5aa5a5
2  root@light-a-val:~# fw_printenv sec_upgrade_mode
3  sec_upgrade_mode=0x5a5aa5a5

```

4. OTA Client 执行重启, 进入uboot进行安全镜像升级。

5. 重启进入系统根据章节3.4查看结果。

注意:

- 安全镜像升级仅支持全量升级。只允许TEE或TF单个镜像升级。

系统镜像升级

1. OTA Client 从云端服务器下载系统镜像（BOOT.EXT4 或ROOTFS.EXT4）
2. OTA Client 负责将系统镜像写入系统分区B。

▼ Bash 复制代码

```
1 cat ROOT.EXT4 > /dev/disk/by-partlabel/rootB
2 cat BOOT.EXT4 > /dev/disk/by-partlabel/bootB
```

3. OTA Client 负责将系统分区A里的镜像复制到系统分区B，再进行差分恢复最新的升级镜像。

▼ Bash 复制代码

```
1 ota-burndiff diff.bin
```

4. OTA Client 通过Shell命令 fw_printenv和fw_setenv 设置 uboot的env环境变量 root_partition 为 rootfsB或rootfsA

▼ Bash 复制代码

```
1 root@light-a-val:~# fw_setenv root_partition rootfsB
2 root@light-a-val:~# fw_printenv root_partition
3 root_partition=rootfsB
```

5. OTA Client 执行重启，进入uboot进行系统镜像启动。

镜像版本号获取

可以通过以下的伪代码进行TEE和TF的代码获取和解析。

- 版本获取

通过Shell命令 fw_printenv

▼ Bash 复制代码

```
1 root@light-a-val:~# fw_printenv tee_version
2 tee_version=0x000000100
3 root@light-a-val:~# fw_printenv tf_version
4 tf_version=0x000000100
```

- 版本解析

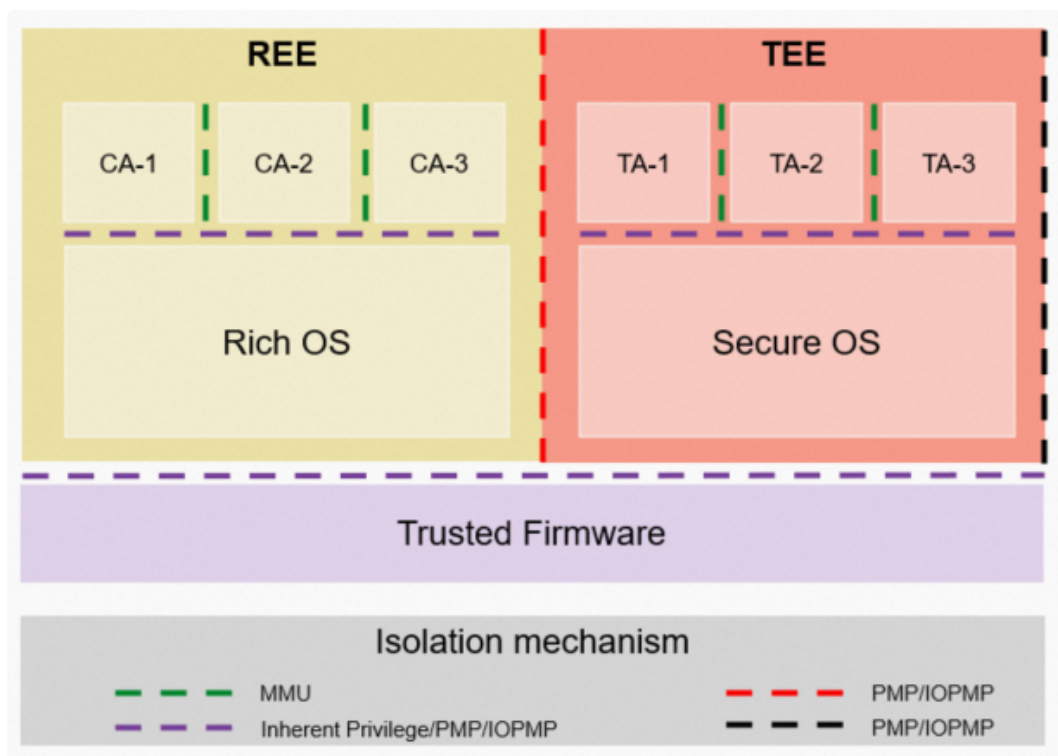
```
1 printf("\n\n");
2 printf("Secure Firmware image version info: \n");
3 printf("uboot Firmware v%d.0\n", (uboot_ver & 0xff00) >> 8);
4 printf("Trust Firmware v%d.%d\n", (tf_ver & 0xff00) >> 8, tf_ver & 0xff);
5 printf("TEE OS v%d.%d\n", (tee_ver & 0xff00) >> 8, tee_ver & 0xff);
6 printf("\n\n");
```

安全隔离

安全系统的安全隔离机制整体上满足国际Global Platform的三层隔离要求，利用RISC-V原生的运行权限模式管理、物理内存保护机制以及可扩展的IOPMP机制等对整个系统的安全资源进行了重新隔离管理，从而实现了以下三层隔离：

- 安全固件和TEE OS/安全应用之间的安全隔离
- TEE OS和REE OS之间的安全隔离
- 安全应用核TEE OS之间的安全隔离

图2-3 安全隔离描述



注意：

- 默认的情况下，TH1520生态开发板的安全资源描述请参考1.3章节。
- 如果有需要新增安全资源，比如内存、外设等，需要联系平头哥进行协助。

安全计算

TH1520提供了丰富的密码学算法，应用开发者可以参考Global Platform Cryptographic APIs进行密码学算法开发。

表2-1 支持的硬算法列表

算法列表	是否支持
HASH-MD5	<input checked="" type="checkbox"/>
HASH-SHA1	<input checked="" type="checkbox"/>
HASH-SHA224	<input checked="" type="checkbox"/>
HASH-SHA256	<input checked="" type="checkbox"/>
HASH-SHA384	<input checked="" type="checkbox"/>
HASH-SHA512	<input checked="" type="checkbox"/>
HASH-MD5SHA1	<input type="checkbox"/>
HASH-SM3	<input checked="" type="checkbox"/>
HMAC-MD5	<input checked="" type="checkbox"/>
HMAC-SHA1	<input checked="" type="checkbox"/>
HMAC-SH224	<input checked="" type="checkbox"/>
HMAC-SH256	<input checked="" type="checkbox"/>
HMAC-SHA384	<input checked="" type="checkbox"/>
HMAC-SHA512	<input checked="" type="checkbox"/>
HMAC-SM3	<input checked="" type="checkbox"/>
AES-ECB-NOPAD	<input checked="" type="checkbox"/>
AES-CBC-NOPAD	<input checked="" type="checkbox"/>
AES-CTR	<input checked="" type="checkbox"/>
AES-CTS	<input type="checkbox"/>
AES-XTS	<input type="checkbox"/>
AES-CFB	<input checked="" type="checkbox"/>

AES-OFB	<input checked="" type="checkbox"/>
AES-CBC-MAC-NOPAD	<input type="checkbox"/>
AES-CBC-MAC-PKCS5	<input type="checkbox"/>
AES-CMAC	<input type="checkbox"/>
AES-CCM	<input type="checkbox"/>
AES-GCM	<input type="checkbox"/>
DES-ECB-NOPAD	<input checked="" type="checkbox"/>
DES-CBC-NOPAD	<input checked="" type="checkbox"/>
DES-MAC-NOPAD	<input type="checkbox"/>
DES-MAC-PKCS5	<input type="checkbox"/>
DES3-ECB-NOPAD	<input checked="" type="checkbox"/>
DES3-CBC-NOPAD	<input checked="" type="checkbox"/>
DES3-CBC-MAC-NOPAD	<input type="checkbox"/>
DES3-CBC-MAC-PKCS5	<input type="checkbox"/>
SM4-EBC-NOPAD	<input checked="" type="checkbox"/>
SM4-CBC-NOPAD	<input checked="" type="checkbox"/>
SM4-CTR	<input type="checkbox"/>
RSASSA-PCKS1-V1-5-MD5	<input checked="" type="checkbox"/>
RSASSA-PCKS1-V1-5-SHA1	<input checked="" type="checkbox"/>
RSASSA-PCKS1-V1-5-SHA224	<input checked="" type="checkbox"/>
RSASSA-PCKS1-V1-5-SHA256	<input checked="" type="checkbox"/>
RSASSA-PCKS1-V1-5-SHA384	<input checked="" type="checkbox"/>
RSASSA-PCKS1-V1-5-SHA512	<input checked="" type="checkbox"/>
RSASSA-PCKS1-V1-5-MD5SHA1	<input type="checkbox"/>
RSASSA-PSS-MGF1-SHA1	<input checked="" type="checkbox"/>

RSASSA-PSS-MGF1-SHA224	<input checked="" type="checkbox"/>
RSASSA-PSS-MGF1-SHA256	<input checked="" type="checkbox"/>
RSASSA-PSS-MGF1-SHA384	<input checked="" type="checkbox"/>
RSASSA-PSS-MGF1-SHA512	<input type="checkbox"/>
RSAES-PKCS1-OAEP-MGF1-SHA1	<input checked="" type="checkbox"/>
RSAES-PKCS1-OAEP-MGF1-SHA224	<input checked="" type="checkbox"/>
RSAES-PKCS1-OAEP-MGF1-SHA256	<input checked="" type="checkbox"/>
RSAES-PKCS1-OAEP-MGF1-SHA384	<input checked="" type="checkbox"/>
RSAES-PKCS1-OAEP-MGF1-SHA512	<input checked="" type="checkbox"/>
RSAES-PKCS1-V1-5	<input checked="" type="checkbox"/>
RSA-NOPAD	<input checked="" type="checkbox"/>
DSA-SHA1	<input type="checkbox"/>
DSA-SHA224	<input type="checkbox"/>
DSA-SHA256	<input type="checkbox"/>
SM2-DSA-SM3	<input checked="" type="checkbox"/>
SM2-KEP	<input checked="" type="checkbox"/>
DH-DERIVE-SHARED-SECRET	<input type="checkbox"/>
ECDH-CURVE25519	<input checked="" type="checkbox"/>
TRNG	<input checked="" type="checkbox"/>

安全存储

REE FS存储

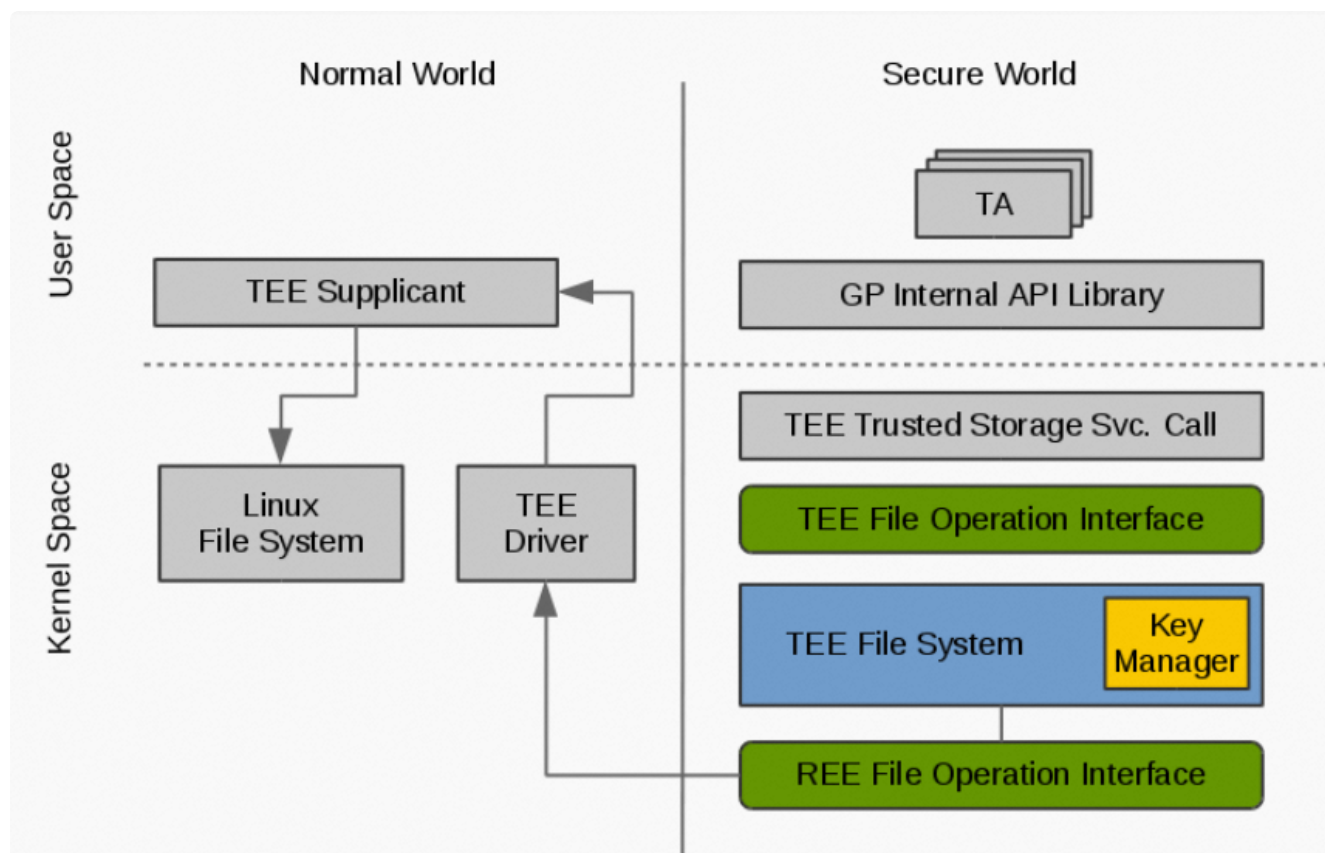
安全系统中的安全存储是根据GlobalPlatform的TEE内部核心API(这里称为可信存储)中定义的内容实现的。该规范要求应该能够存储通用数据和关键材料，以保证所存储数据的机密性和完整性，以及修改存储的操作的原子性(原子性在这里意味着整个操作成功完成或没有写入)。

目前OP-TEE中有两种安全存储实现:

第一个依赖于普通世界(REE)文件系统。它在本文档中有描述, 是默认的实现。它在编译时通过CFG_REE_FS=y启用。

第二个使用eMMC设备的重放保护内存块(RPMB)分区, 通过设置CFG_RPMB_FS=y启用。在RPMB安全存储中有描述。

图2-5 安全存储描述



当TA调用GP可信存储API提供的写函数将数据写入持久对象时, 将调用TEE可信存储服务中实现的相应系统调用, 该系统调用将调用一系列TEE文件操作来存储数据。然后TEE文件系统将对数据进行加密, 并通过一系列RPC消息将REE文件操作命令和加密后的数据发送给TEE请求者。TEE请求者将接收消息并相应地将加密数据存储到Linux文件系统中。读取文件的处理方式与此类似。

注意:

- TH1520 Linux SDK默认情况下采用REE FS文件系统


eFuse数据存储

eFuse具有OTP (One Time Program) 的特性, 写入到efuse里的数据不能被改写和更新, 在硬件机制也保证写入数据的完整性, 所以其具有一定的安全性。

针对有些产品， 开发者需要根据产品独有的功能进行个性化功能测试开发，这就需要使用到测试套件里的efuse访问接口，根据efuse里的数据或功能配置进行个性化测试。具体efuse map的字段含义请联系平头哥。

注意：个性化测试必须在芯片处于生命周期OEM的时候实施, TH1520生态开发板由于芯片处于生命周期INIT，所以不具备访问权限。

为了方便用户开发，TH1520 Linux SDK提供了操作efuse的用户开发库，参考以下接口可以完成对efuse用户区域的访问。

交付项	交付件
libefuse 操作库	 efuse_test_demo_v1.0.zip
demo 代码	
用户手册	

csi_efuse_read

▼ Plain Text |  复制代码

```
1  ● 功能描述：
2    ○ 从efuse 中读取efuse 的值
3  ● 参数：
4    ○ offset:  efuse 中的偏移地址，单位位byte
5    ○ data: 指向存放读取efuse值的指针
6    ○ cnt: 读取的字节数
7  ● 返回值：
8    ○ 读取的字节数，如果不等于cnt，说明数据读取出错
9  int  csi_efuse_read(unsigned int offset, void *data, unsigned int cnt)
10
11 ● 使用示例：
12 int ret;
13 unsigned int addr = 0;
14 unsigned char data[4];
15
16 ret = csi_efuse_read(addr, data, sizeof(data));
17 if (ret !=  sizeof(data))
18     return -1;
```

csi_efuse_write

```

1  ● 功能描述：
2    ○ 向efuse区域写对应的值
3  ● 参数：
4    ○ offset: efuse 中的便宜地址，单位位byte
5    ○ data: 指向存放要写入efuse值的指针
6    ○ cnt: 需要写入的字节数
7  ● 返回值：
8    ○ 写入的字节数，如果不等于cnt，说明数据读取出错
9  int csi_efuse_write(unsigned int offset, void *data, unsigned int cnt)
10
11 ● 使用示例：
12 int ret;
13 unsigned int addr = 0;
14 unsigned int data = 0x12345678;
15
16 ret = csi_efuse_write(addr, &data, sizeof(data));
17 if (ret != sizeof(data))
18     return -1;

```

efuse配置自检

以下开发测试伪代码，用户可以很方便的集成到自己的测试程序里去。如果要增加个性化测试代码，请在Step2进行添加。

```

1  int ret;
2
3  //step 1: efuse check
4  ret = csi_efuse_self_check();
5  if (ret)
6      return -1;
7
8  //step 2: //TBD by user: 用户自己决定是否有烧写efuse的需求
9
10 //step 3: update life cycle to product
11 ret = csi_efuse_update_lc_pro();
12 if (ret)
13     return -1;
14
15 return 0;

```

编译成功运行后，可以看到以下输出日志:

```









1  start efuse self check...
2  check SECURE_BOOT passed.
3  check BROM_PRINT_DIS passed.
4  check CV_BROM_CCT_DIS passed.
5  check BROM_ICACHE_EN passed.
6  check BROM_DCACHE_EN passed.
7  check BROM_TEECLK_SWITCH passed.
8  check TRNG_BYPASS passed.
9  check CV_USB_FASTBOOT_DIS passed.
10 check DIGEST_SCHEME passed.
11 check SIGN_SCHEME passed.
12 check IMAGE_BL1_ENC passed.
13 check TEECLK_SWITCH_MODE passed.
14 check HASH_ROTPK passed.
15 life cycle check passed.
16 efuse sefl check finished,error occurs.
17 life cycle update to product success.

```

安全工具

Fastboot

fastboot是一种比recovery更底层的刷机模式（俗称引导模式）。就是使用USB数据线连接终端的一种刷机模式。我们利用fastboot进行系统镜像的更新。

-  light_fm_single_rank_system.bat
-  light_fm_single_rank_tee.bat
-  light_fm_single_rank_tee_upd.bat
-  light_fm_single_rank_tf.bat
-  light_fm_single_rank_tf_upd.bat
-  light_fm_single_rank_uboot.bat
-  light_fm_single_rank_uboot_upd.bat
-  light_fm_single_rank_uboot-raw.bat

- light_fm_single_rank_system.bat – 用于直接烧写所有系统镜像
- light_fm_single_rank_uboot.bat – 用于直接烧写uboot镜像，不进行升级流程
- light_fm_single_rank_uboot_upd.bat – 用于更新uboot镜像，进行升级路程
- light_fm_single_rank_uboot_raw.bat – 用于更新烧写uboot镜像(uboot损坏)，不进行升级流程
- light_fm_single_rank_tee.bat – 用于直接烧写tee镜像，不进行升级流程

- light_fm_single_rank_tee_upd.bat – 用于直接更新tee镜像，进行升级流程
- light_fm_single_rank_tf.bat – 用于直接烧写tf镜像，不进行升级流程
- light_fm_single_rank_tf_upd.bat – 用于直接更新tf镜像，进行升级流程

注意：

Fastboot工具位于 `../tools/fastboot` 路径下 ([链接](#))，使用前需要安装ADB驱动，请参考 *TH1520_Linux_SDK_QuickStart*。

Product

product用于系统镜像的签名，支持指定的密钥证书等功能。该工具位于sectool目录下，无需再重新安装。

功能命令：

sigx

说明：

对已生成好的单个镜像(或者公钥PEM文件)进行签名。

输入product sigx可查看帮助信息。

示例：

2级公钥的签名

- 国际算法例子：

```
1 product sigx keystore/pubkeyB.pem -pvk keystore/privatekeyA.pem -pubk keystore/pubkeyA.pem -ss RSA2048 -ds SHA256 -npubk keystore/pubkeyB.pem -nss RSA2048 -nds SHA256 -o sign_2nd_pubkey.bin
```

- 国密算法例子：

```
1 product sigx keystore_sm/pubkeyB.pem -pvk keystore_sm/privatekeyA.pem -pubk keystore_sm/pubkeyA.pem -ss SM2 -ds SM3 -npubk keystore_sm/pubkeyB.pem -nss SM2 -nds SM3 -o sign_2nd_pubkey.bin
```

镜像的签名(带有下级公钥)

- 国际算法例子:

▼ [C](#) | [复制代码](#)

```
1 product sigx iw.bin -pvk keystore/privatekeyB.pem -ss RSA2048 -ds SHA256 -n
  pubk keystore/pubkeyC.pem -nss RSA2048 -nds SHA256 -iv 0 -ra 0xFFE0000000 -
  o sign_iw.bin
```

- 国密算法例子:

▼ [C](#) | [复制代码](#)

```
1 product sigx iw.bin -pvk keystore_sm/privatekeyB.pem -ss SM2 -ds SM3 -npubk
  keystore_sm/pubkeyC.pem -nss SM2 -nds SM3 -iv 0 -ra 0xFFE0000000 -o sign_i
  w.bin
```

镜像的加密签名(带下级公钥)

- 国际算法例子:

▼ [C](#) | [复制代码](#)

```
1 product sigx iw.bin -pvk keystore/privatekeyB.pem -ss RSA2048 -ds SHA256 -n
  pubk keystore/pubkeyC.pem -nss RSA2048 -nds SHA256 -ent AES_256_CBC -enk ke
  ystore/aes_256_cbc.key -iv 0 -ra 0xFFE0000000 -o sign_iw.bin
```

- 国密算法例子:

▼ [C](#) | [复制代码](#)

```
1 product sigx iw.bin -pvk keystore_sm/privatekeyB.pem -ss SM2 -ds SM3 -npubk
  keystore_sm/pubkeyC.pem -nss SM2 -nds SM3 -ent SM4_CBC -enk keystore_sm/sm
  4.key -iv 0 -ra 0xFFE0000000 -o sign_iw.bin
```

镜像的签名(不带下级公钥)

- 国际算法例子:

▼ [C](#) | [复制代码](#)

```
1 product sigx iw.bin -pvk keystore/privatekeyB.pem -ss RSA2048 -ds SHA256 -i
  v 0 -ra 0xFFE0000000 -o sign_iw.bin
```

- 国密算法例子:

C | 复制代码

```
1 product sigx iw.bin -pvk keystore_sm/privatekeyB.pem -ss SM2 -ds SM3 -iv 0  
-ra 0xFFE0000000 -o sign_iw.bin
```

镜像的加密签名(不带有下级公钥)

- 国际算法例子:

C | 复制代码

```
1 product sigx iw.bin -pvk keystore/privatekeyB.pem -ss RSA2048 -ds SHA256 -e  
nt AES_256_CBC -enk keystore/aes_256_cbc.key -iv 0 -ra 0xFFE0000000 -o sign  
_iw.bin
```

- 国密算法例子:

C | 复制代码

```
1 product sigx iw.bin -pvk keystore_sm/privatekeyB.pem -ss SM2 -ds SM3 -ent S  
M4_CBC -enk keystore_sm/sm4.key -iv 0 -ra 0xFFE0000000 -o sign_iw.bin
```

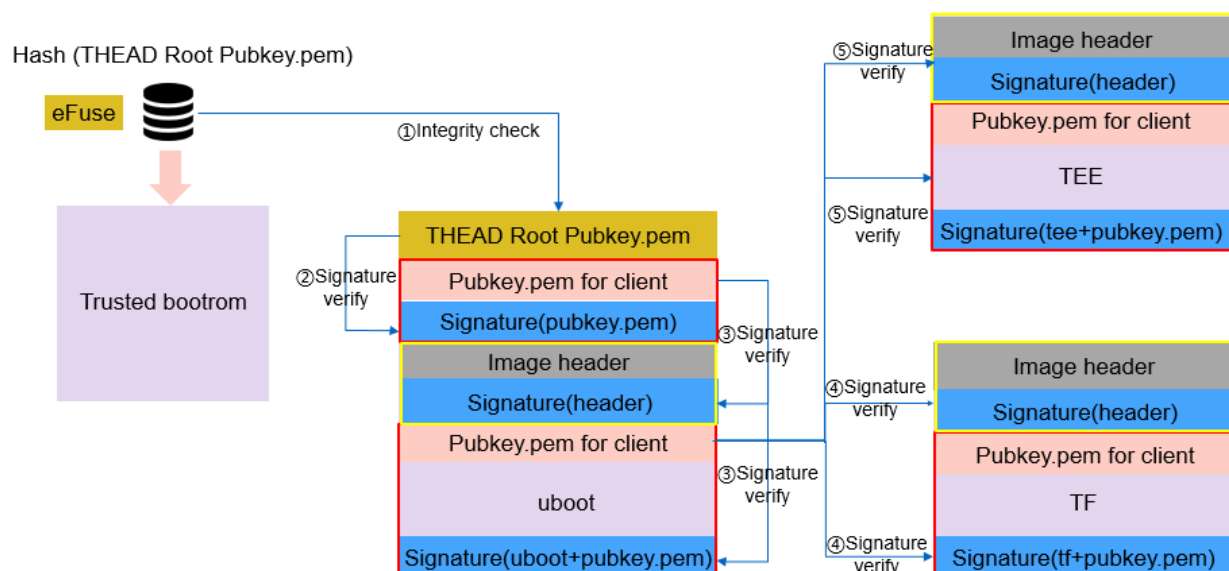
Sectool

sectool用于镜像签名打包的工具，包括支持二进制文件打包成EXT4文件。其主要包括imagesign.sh和bin2ext4.sh脚本。其位于Linux SDK开发包根目录/tools/sectool路径下。

Bash | 复制代码

```
1 cxx194832@docker-ubuntu18:~/linux_sdk/tools/sectool$ ls  
2 bin2ext4 imagesign.sh keystore keystore_sm README.md tee tf tool ub  
oot
```

Pubkey.pem for client表示二级签名私钥对应的公钥



在进行镜像签名前，可以在imagesign.sh脚本里通过修改以下两个变量来指定密钥证书。

- `client_public_cert`
- `client_private_cert`

注意：国际算法证书必须放在keystore文件夹下，国密算法证书必须放在keystore_sm文件夹下。以下证书用于uboot镜像签名，一般情况下由平头哥进行提供管理：

- `thead_root_public_cert`
- `thead_root_private_cert`
- `thead_b1_public_cert`
- `thead_b1_private_cert`

bin2ext4

bin2ext4工具用于将一个文件打包生成EXT4文件。

注意：在默认情况下，打包的EXT4文件大小是8M，如果文件真实大小大于8M，需要调整成合适的值。

bin2ext4工具用于将一个文件打包生成EXT4文件。

注意：在默认情况下，打包的EXT4文件大小是8M，如果文件真实大小大于8M，需要调整成合适的值。

```

1  dd if=/dev/zero of=$1 bs=1M count=8

```

imagesign

imagesign工具是一个运行在shell下的一个脚本文件，其用于将一个二进制文件用指定的算法进行签名，同时还能指定镜像是否加密和版本号等信息，最后进行镜像内容和签名数据等信息打包，生成一个签名文件。

Bash | 复制代码

```
1  imagesign.sh utility version v1.0
2  The utility is designed for aim to help user generate new image file
3  with signature with desired sign scheme.
4
5  Usage:
6  imagesign.sh [chkuboot] [ia/sm] [enc/nor] [tf/tee/uboot] [ver]
7  chkuboot: check uboot binary file is signed or not, if yes, it generates the
8  original bboot binary file
9  signed algorithms
10         ia - international algorithm
11         sm - china algorithms
12 secure attribution
13         enc - signed image with encryption
14         nor - signed image without encryption
15 image file type
16         tf - trust_firmware binary image
17         tee - tee binary image
18         boot - uboot binary image
19 version
20         ver - image version (x.y), eg 1.1, 2.1
```

UBOOT镜像签名

1. 将需要签名的u-boot-with-spl.bin复制到uboot文件夹里

注意：文件名必须是 **u-boot-with-spl.bin**，uboot文件夹名字不可以更改

Bash | 复制代码

```
1  cxx194832@docker-ubuntu18:~/sectool$ ls uboot/
2  u-boot-with-spl.bin
```

2. 执行下面命令进行镜像签名


```

1  cxx194832@docker-ubuntu18:~/sectool$ ./imagesign.sh ia enc uboot 1.2
2  Dump all parameters from user input.
3  -----
4  Signed algorithm: ia
5  Secure attribution: enc
6  Image file: uboot
7  Image version: 1.2
8  -----
9  Enter into image sign process ...
10 Start uboot Image (1.2) signed with international algorithms with secure a
    ttr (enc)
11 sign tool path: ./tool/product
12 Original file: uboot/u-boot-with-spl.bin
13 Signed file: uboot/signed_ia_enc_u-boot-with-spl.bin
14 Image Version: 258
15 Relocate Addr: 0xFFE0000800
16 ▾ [2022-03-23 14:09:44] Sign a public key file.
17 ▾ [2022-03-23 14:09:44] rsa verify ok....
18 ▾ [2022-03-23 14:09:44] rsa verify ok....
19 ▾ [2022-03-23 14:09:44] Sign ok.
20 ▾ [2022-03-23 14:09:44] Sign a image file.
21 ▾ [2022-03-23 14:09:44] Sign ok.
22 Exit from image sign process ...
23

```

3. 查看生成的uboot签名镜像

```

1  cxx194832@docker-ubuntu18:~/sectool$ ls uboot/
2  signed_ia_enc_u-boot-with-spl.bin  signed_image_u-boot-with-spl.bin
3  signed_pubkey_u-boot-with-spl.bin  u-boot-with-spl.bin

```

生成的 `signed_ia_enc_u-boot-with-spl.bin` 位于uboot文件夹下。

注意：

- `uboot`签名镜像由 `signed_pubkey_u-boot-with-spl.bin` 和 `signed_image_u-boot-with-spl.bin` 两部分签名镜像组成。
- `signed_pubkey_u-boot-with-spl.bin` 由平头哥管理
- `signed_image_u-boot-with-spl.bin` 用于镜像升级，但签名镜像的私钥必须和 `signed_pubkey_u-boot-with-spl.bin` 了用的公钥配对。
- 使用的时候必须将 `signed_ia_enc_u-boot-with-spl.bin` 改名为 `u-boot-with-spl.bi`

n

- 当不确定当前的u-boot-with-spl.bin是否已经签名, 可以通过 `imagesign.sh chkuboot u-boot-with-spl.bin` 命令进行检查, 如果已经签名, 则会生成原始未签名的uboot.original文件(注: 目前chkuboot不支持加密镜像的还原,即国密加密或者国际加密镜像无法使用chkuboot进行还原), 可以使用该文件进行其他算法的签名。

TF镜像签名

1. 将需要签名的trust_firmware.bin复制到tf文件夹里

注意: 文件名必须是 `trust_firmware.bin`, tf文件夹名字不可以更改

```
▼ Bash 复制代码
1 cxx194832@docker-ubuntu18:~/sectool$ ls tf
2 trust_firmware.bin
```

2. 执行下面命令进行镜像签名

```
▼ Bash 复制代码
1 cxx194832@docker-ubuntu18:~/sectool$ ./imagesign.sh ia nor tf 1.2
2 Dump all parameters from user input.
3 -----
4 Signed algorithm: ia
5 Secure attribution: nor
6 Image file: tf
7 Image version: 1.2
8 -----
9 Enter into image sign process ...
10 Start tf Image (1.2) signed with international algorithms with secure att
   r (nor)
11 sign tool path: ./tool/product
12 Original file: tf/trust_firmware.bin
13 Signed file: tf/signed_ia_nor_trust_firmware.bin
14 Image Version: 258
15 Relocate Addr: 0x0
16 [2022-03-23 14:12:36] Sign a image file.
17 [2022-03-23 14:12:36] Sign ok.
18 Exit from image sign process ...
```

3. 查看生成的TF签名镜像

```
1 cxx194832@docker-ubuntu18:~/sectool$ ls tf
2 signed_ia_nor_v1.2_trust_firmware.bin trust_firmware.bin
```

生成的 `signed_ia_nor_trust_firmware.bin` 位于tf文件夹下。

注意：

- 使用的时候必须将 `signed_ia_nor_v1.2_trust_firmware.bin` 改名为 `trust_firmware.bin`

TEE镜像签名

1. 将需要签名的tee.bin复制到tee文件夹里

注意：文件名必须是 `tee.bin`，tee文件夹名字不可以更改

```
1 cxx194832@docker-ubuntu18:~/sectool$ ls tee
2 tee.bin
```

2. 执行下面命令进行镜像国密算法签名

```

1  cxx194832@docker-ubuntu18:~/sectool$ ./imagesign.sh ia nor tee 1.2
2  Dump all parameters from user input.
3  -----
4  Signed algorithm: ia
5  Secure attribution: nor
6  Image file: tee
7  Image version: 1.2
8  -----
9  Enter into image sign process ...
10 Start tee Image (1.2) signed with international algorithms with secure att
    r (nor)
11 sign tool path: ./tool/product
12 Original file: tee/tee.bin
13 Signed file: tee/signed_ia_nor_tee.bin
14 Image Version: 258
15 Relocate Addr: 0xff000000
16 [2022-03-23 14:13:31] Sign a image file.
17 [2022-03-23 14:13:31] Sign ok.
18 Exit from image sign process ...
19

```

3. 查看生成的TEE签名镜像

```

1  cxx194832@docker-ubuntu18:~/sectool$ ls tee
2  signed_sm_nor_v1.2_tee.bin  tee.bin

```

生成的 `signed_sm_nor_v1.2_tee.bin` 位于tee文件夹下。

注意：

- 使用的时候必须将 `signed_sm_nor_v1.2_tee.bin` 改名为 `tee.bin`

内核镜像签名

1. sectool根目录下新建ree文件夹
2. 将需要签名boot.ext4复制到ree文件夹

```

1  kjs@kjs-VirtualBox:~/sectool$ ls ./ree
2  boot.ext4

```

3. 执行下面命令进行镜像国际算法签名

Bash | 复制代码

```
1 kjs@kjs-VirtualBox:~/sectools/sectool$ ./imagesign.sh ia nor ree 1.0
2 Dump all parameters from user input.
3 -----
4 Signed algorithm: ia
5 Secure attribution: nor
6 Image file: ree
7 Image version: 1.0
8 -----
9 Enter into image sign process ...
10 Start ree Image (1.0) signed with international algorithms with secure att
   r (nor)
11 sign tool path: ./tool/product
12 Original file: bootimg/Image
13 Signed file: ree/signed_ia_nor_v1.0_Image
14 Image Version: 256
15 Relocate Addr: 0x200000
16 [2023-02-16 18:30:09] Sign a image file.
17 [2023-02-16 18:30:09] Sign ok.
18 Exit from image sign process ...
```

4. 查看生成的REE签名镜像

Bash | 复制代码

```
1 kjs@kjs-VirtualBox:~/sectools/sectool/ree$ ls
2
3 boot.ext4  signed_ia_nor_v1.0_Image
```

生成的 签名的boot.ext4 位于ree文件夹下。

安全应用开发

开发接口

TEE Internal APIs

Trusted Core Framework API

函数接口	是否支持
TEE_GetPropertyAsString	✓
TEE_GetPropertyAsBool	✓
TEE_GetPropertyAsU32	✓
TEE_GetPropertyAsU6	☐
TEE_GetPropertyAsBinaryBlock	✓
TEE_GetPropertyAsUUID	✓
TEE_GetPropertyAsIdentity	✓
TEE_AllocatePropertyEnumerator	✓
TEE_FreePropertyEnumerator	✓
TEE_StartPropertyEnumerator	✓
TEE_ResetPropertyEnumerator	✓
TEE_GetPropertyName	✓
TEE_GetNextProperty	✓
TEE_Panic	✓
TEE_OpenTASession	✓
TEE_CloseTASession	✓
TEE_InvokeTACommand	✓
TEE_GetCancellationFlag	✓
TEE_MaskCancellation	✓
TEE_CheckMemoryAccessRights	✓
TEE_SetInstanceData	✓
TEE_GetInstanceData	✓
TEE_Malloc	✓
TEE_Realloc	✓

TEE_Free	✓
TEE_MemMove	✓
TEE_MemCompare	✓
TEE_MemFill	✓

Trusted Storage API

函数接口	是否支持
TEE_GetObjectInfo1	✓
TEE_RestrictObjectUsage1	✓
TEE_GetObjectBufferAttribute	✓
TEE_GetObjectValueAttribute	✓
TEE_CloseObject	✓
TEE_AllocateTransientObject	✓
TEE_FreeTransientObject	✓
TEE_ResetTransientObject	✓
TEE_PopulateTransientObject	✓
TEE_InitRefAttribute	✓
TEE_InitValueAttribute	✓
TEE_CopyObjectAttributes1	✓
TEE_GenerateKey	✓
TEE_OpenPersistentObject	✓
TEE_CreatePersistentObject	✓
TEE_CloseAndDeletePersistentObject1	✓
TEE_RenamePersistentObject	✓
TEE_AllocatePersistentObjectEnumerator	✓

TEE_FreePersistentObjectEnumerator	✓
TEE_ResetPersistentObjectEnumerator	✓
TEE_StartPersistentObjectEnumerator	✓
TEE_GetNextPersistentObject	✓
TEE_ReadObjectData	✓
TEE_WriteObjectData	✓
TEE_TruncateObjectData	✓

Cryptographic operation API

函数接口	是否支持
TEE_AllocateOperation	✓
TEE_FreeOperation	✓
TEE_GetOperationInfo	✓
TEE_GetOperationInfoMultiple	✓
TEE_ResetOperation	✓
TEE_SetOperationKey	✓
TEE_SetOperationKey2	✓
TEE_CopyOperation	✓
TEE_IsAlgorithmSupported	✓
TEE_DigestUpdate	✓
TEE_DigestDoFinal	✓
TEE_CipherInit	✓
TEE_CipherUpdate	✓
TEE_CipherDoFinal	✓
TEE_MACInit	✓

TEE_MACUpdate	✓
TEE_MACComputeFinal	✓
TEE_MACCompareFinal	✓
TEE_AEInit	✓
TEE_AEUpdateAAD	✓
TEE_AEUpdate	✓
TEE_AEEncryptFinal	✓
TEE_AEDecryptFinal	✓
TEE_AsymmetricEncrypt	✓
TEE_AsymmetricDecrypt	✓
TEE_AsymmetricSignDigest	✓
TEE_AsymmetricVerifyDigest	✓
TEE_DeriveKey	✓
TEE_GenerateRandom	✓

Timer API

函数接口	是否支持
TEE_GetSystemTime	✓
TEE_Wait	✓
TEE_GetTAPersistentTime	✓
TEE_SetTAPersistentTime	✓
TEE_GetREETime	✓

TEE Client APIs

函数接口	是否支持
------	------

TEEC_InitializeContext	✓
TEEC_FinalizeContext	✓
TEEC_OpenSession	✓
TEEC_CloseSession	✓
TEEC_InvokeCommand	✓
TEEC_RegisterSharedMemory	✓
TEEC_AllocateSharedMemory	✓
TEEC_ReleaseSharedMemory	✓
TEEC_RequestCancellation	✓

C Lib APIs

序号	接口
1	void *malloc(size_t size);
2	void *calloc(size_t nmemb, size_t size);
3	void *realloc(void *ptr, size_t size);
4	void free(void *ptr)
5	int printf(const char *fmt, ...)
6	int sprintf(char *str, const char *fmt, ...)
7	int snprintf(char *str, size_t size, const char *fmt, ...)
8	int puts(const char *str);
9	int putchar(int c);
10	void qsort(void *aa, size_t n, size_t es, int (*cmp)(const void *, const void *));
11	void abort(void);
12	int abs(int i);
13	int rand(void);

14	unsigned long strtoul (const char *s, char **ptr, int base);
15	void *memcpy(void *__restrict s1, const void *__restrict s2, size_t n);
16	void *memmove(void *s1, const void *s2, size_t n);
17	int memcmp(const void *s1, const void *s2, size_t n);
18	int strcmp(const char *s1, const char *s2);
19	int strncmp(const char *s1, const char *s2, size_t n);
20	size_t strlen(const char *s);
21	size_t strnlen(const char *s, size_t n);
22	char *strdup(const char *s);
23	char *strndup(const char *s, size_t n);
24	char *strchr(const char *s, int c);
25	char *strstr(const char *big, const char *little);
26	char *strcpy(char *dest, const char *src);
27	char *strncpy(char *dest, const char *src, size_t n);
28	char *strrchr(const char *s, int i);
29	void *memchr(const void *buf, int c, size_t length);
30	int bcmp(const void *s1, const void *s2, size_t n);
31	size_t strlcpy(char *dst, const char *src, size_t size);
32	size_t strlcat(char *dst, const char *src, size_t size);
33	void *memset(void *s, int c, size_t n);

TA开发SDK

SDK介绍

TA开发SDK位于根目录~/software/Tsec_dev_kit/Tsec_sdk/路径下，目录文件结构如下：

```

1  .
2  |— demo
3  |   |— host
4  |   |— ta
5  |— export
6  |   |— ta-rv64
7  |   |— tee-client

```

其中：

- demo

存放的是CA/TA的参考例子。host 文件夹下存放的是参考CA， ta目录存放的是参考TA。

- export

存放的是开发CA/TA所需要的一些依赖，如，头文件，库，以及编译脚本。其中：

- ta-rv64 为编译TA所需要的依赖
- tee-client 为编译CA所需要的依赖

SDK开发

安全应用的开发包括两部分工作，一部分是CA的开发，CA运行在Linux侧；另一部分是TA的开发，TA运行在TEE侧。

CA开发

代码开发

CA 是运行在REE侧的应用程序。用户可以通过调用GP 定义client API接口调用指定TA的安全功能。CA的代码与普通的linux应用程序的开发没什么特别大的差别，唯一的差别是在API接口上集成了GP的Client API。开发者可以按照自身的开发需求在需要的时候按照Client API的调用标准（详细说明请参考GP发布的TEE Client API Specification）去调用TA。如demo中的例子，TEEC_InitializeContext -> TEEC_OpenSession -> TEEC_InvokeCommand -> TEEC_CloseSession。

```

/* Initialize a context connecting us to the TEE */
res = TEEC_InitializeContext(NULL, &ctx);
if (res != TEEC_SUCCESS)
    errx(1, "TEEC_InitializeContext failed with code 0x%x", res);

/*
 * Open a session to the "hello world" TA, the TA will print "hello
 * world!" in the log when the session is created.
 */
res = TEEC_OpenSession(&ctx, &sess, &uuid,
                      TEEC_LOGIN_PUBLIC, NULL, NULL, &err_origin);
if (res != TEEC_SUCCESS)
    errx(1, "TEEC_OpenSession failed with code 0x%x origin 0x%x",
        res, err_origin);

/*
 * Execute a function in the TA by invoking it, in this case
 * we're incrementing a number.
 *
 * The value of command ID part and how the parameters are
 * interpreted is part of the interface provided by the TA.
 */

/* Clear the TEEC_Operation struct */
memset(&op, 0, sizeof(op));

/*
 * Prepare the argument. Pass a value in the first parameter,
 * the remaining three parameters are unused.
 */
op.paramTypes = TEEC_PARAM_TYPES(TEEC_VALUE_INOUT, TEEC_NONE,
                                TEEC_NONE, TEEC_NONE);
op.params[0].value.a = 42;

/*
 * TA_HELLO_WORLD_CMD_INC_VALUE is the actual function in the TA to be
 * called.
 */
printf("Invoking TA to increment %d\n", op.params[0].value.a);
res = TEEC_InvokeCommand(&sess, TA_HELLO_WORLD_CMD_INC_VALUE, &op,
                        &err_origin);
if (res != TEEC_SUCCESS)
    errx(1, "TEEC_InvokeCommand failed with code 0x%x origin 0x%x",
        res, err_origin);
printf("TA incremented value to %d\n", op.params[0].value.a);

/*
 * We're done with the TA, close the session and
 * destroy the context.
 *
 * The TA will print "Goodbye!" in the log when the
 * session is closed.
 */
TEEC_CloseSession(&sess);

```

MK脚本修改

```
#####
## parameter define
#####

## Toolchain
CROSS_COMPILE ?= /home/qbq/work/src/xuantie-tee/toolchains/riscv64/bin/riscv64-linux-gnu-

## TEE export path
TEEC_EXPORT ?= ../../export/tee-client

CC      = $(CROSS_COMPILE)gcc
LD       = $(CROSS_COMPILE)ld
AR       = $(CROSS_COMPILE)ar
NM       = $(CROSS_COMPILE)nm
OBJCOPY  = $(CROSS_COMPILE)objcopy
OBJDUMP  = $(CROSS_COMPILE)objdump
READELF  = $(CROSS_COMPILE)readelf

OBJS = main.o

CFLAGS += -Wall -I../ta/include -I$(TEEC_EXPORT)/include -I./include
#Add/link other required libraries here
LDADD += -lteec -L$(TEEC_EXPORT)/lib

BINARY = optee_example_hello_world

.PHONY: all
all: $(BINARY)
    echo $(CROSS_COMPILE)

$(BINARY): $(OBJS)
    echo $(CROSS_COMPILE)
    $(CC) $(CFLAGS) -o $@ $< $(LDADD)

.PHONY: clean
clean:
    rm -f $(OBJS) $(BINARY)

%.o: %.c
    $(CC) $(CFLAGS) -c $< -o $@

~
~
```

参考例子中Makefile如上图所示，一些关键变量说明如下：

- CROSS_COMPILE
定义了工具链的路径，开发者需要替换成自己开发环境中的工具链
- TEEC_EXPORT
为上面SDK开发环境介绍中的export目录，如果目录结构有改动，需要调整这个路径
- OBJS
为需要编译对象的目标文件
- CFLAGS
为编译参数
- LDADD
为链接相关的编译参数
- BINARY
为要生成的目标文件

代码编译

编译命令：

▼

C | 复制代码

```
1 make
```

编译成功后，会生成目标二进制文件：

```
qbq@docker-ubuntu18:host$ ll
total 36
drwxr-xr-x 2 qbq users 4096 Mar  3 22:17 ./
drwxr-xr-x 4 qbq users 4096 Mar  1 14:47 ../
-rw-r--r-- 1 qbq users 3465 Mar  1 14:48 main.c
-rw-r--r-- 1 qbq users 3688 Mar  3 17:08 main.o
-rw-r--r-- 1 qbq users  957 Mar  1 15:06 Makefile
-rwxr-xr-x 1 qbq users 12576 Mar  3 17:08 optee_example_hello_world*
qbq@docker-ubuntu18:host$
```

TA 开发

TA是运行TEE-OS的安全应用。在安全应用中，用户可以通过GP internal API接口调用TEE-OS的相关驱动或者其它TA。

要编译TA以下文件必须要实现：

- Makefile -- 编译用的Makefile文件
- sub.mk -- 定义要编译的文件
- user_ta_header_defines.h -- 当前要编译的用户 ta头文件，里面定义了当前TA的属性
- 实现TA功能的.c 文件，该文件中必须要实现以下GP 定义的函数（具体函数定义可以参考GP internal API手册）
 - TA_CreateEntryPoint()
 - TA_DestroyEntryPoint()
 - TA_OpenSessionEntryPoint()
 - TA_CloseSessionEntryPoint()
 - TA_InvokeCommandEntryPoint()

可以参考demo中TA如下：

```
1  .
2  |— Android.mk.DD
3  |— hello_world_ta.c
4  |— include
5  |   |— hello_world_ta.h
6  |— Makefile
7  |— sub.mk
8  |— user_ta_header_defines.h
```

编码开发

TA API

用户在开发TA的时候，以下函数必须要实现：


```

1  TEE_Result TA_CreateEntryPoint(void)
2  {
3      /* Allocate some resources, init something, ... */
4      ...
5      /* Return with a status */
6      return TEE_SUCCESS;
7  }
8  void TA_DestroyEntryPoint(void)
9  {
10     /* Release resources if required before TA destruction */
11     ...
12 }
13 TEE_Result TA_OpenSessionEntryPoint(uint32_t ptype,
14 TEE_Param param[4],
15 void **session_id_ptr)
16 {
17     /* Check client identity, and alloc/init some session resources if any */
18     ...
19     /* Return with a status */
20     return TEE_SUCCESS;
21 }
22 void TA_CloseSessionEntryPoint(void *sess_ptr)
23 {
24     /* check client and handle session resource release, if any */
25     ...
26 }
27 TEE_Result TA_InvokeCommandEntryPoint(void *session_id,
28 uint32_t command_id,
29 uint32_t parameters_type,
30 TEE_Param parameters[4])
31 {
32     /* Decode the command and process execution of the target service */
33     ...
34     /* Return with a status */
35     return TEE_SUCCESS;
36 }

```

定义TA属性

TA的属性定义在头文件：user_ta_header_defines.h中里面必须包含：

- TA_UUID

可以通过以下脚本生成UUID：

```
1 python -c "import uuid; u=uuid.uuid4(); print(u); \
2 n = [' ', 0x'] * 11; \
3 n[::2] = ['{:12x}'.format(u.node)[i:i + 2] for i in range(0, 12, 2)]; \
4 print('\n' + '#define TA_UUID\n\t{ ' + \
5 '0x{:08x}'.format(u.time_low) + ', ' + \
6 '0x{:04x}'.format(u.time_mid) + ', ' + \
7 '0x{:04x}'.format(u.time_hi_version) + ', \\ \n\n\t\t{ ' + \
8 '0x{:02x}'.format(u.clock_seq_hi_variant) + ', ' + \
9 '0x{:02x}'.format(u.clock_seq_low) + ', ' + \
10 '0x' + ''.join(n) + ' } }'"
```

- 运行该命令得到如下结果（因为uuid是随机的，所以每次运行的结果是不一样的，以下参数仅供参考）：

```
1 7e5fd739-7a76-4391-9b99-c43a10a817fb
2
3 #define TA_UUID
4 { 0x7e5fd739, 0x7a76, 0x4391, \
5
6 { 0x9b, 0x99, 0xc4, 0x3a, 0x10, 0xa8, 0x17, 0xfb} }
```

- TA_FLAGS
定义TA的属性，如TA_FLAG_USER_MODE, TA_FLAG_EXEC_DDR, TA_FLAG_SINGLE_INSTANCE, 具体的含义可参考GP internal API 手册
- TA_STACK_SIZE
定义当前TA栈的大小，建议值2*1024
- TA_DATA_SIZE
定义当前TA堆的大小，建议用32*1024

MK脚本修改

```

CFG_TEE_TA_LOG_LEVEL ?= 4

CROSS_COMPILE=~/.work/src/xuantie-tee/toolchains/riscv64/bin/riscv64-linux-gnu-
TA_DEV_KIT_DIR=../../export/ta-rv64
# The UUID for the Trusted Application
BINARY=8aaaf200-2450-11e4-abe2-0002a5d5c51b

-include $(TA_DEV_KIT_DIR)/mk/ta_dev_kit.mk

ifeq ($(wildcard $(TA_DEV_KIT_DIR)/mk/ta_dev_kit.mk), )
clean:
    @echo 'Note: $$ (TA_DEV_KIT_DIR)/mk/ta_dev_kit.mk not found, cannot clean TA'
    @echo 'Note: TA_DEV_KIT_DIR=$(TA_DEV_KIT_DIR)'
endif
~
~
~
~

```

Makefile文件格式如上，其中：

- CFG_TEE_TA_LOG_LEVEL
当前TA的log级别，参数越大，打印级别越高，如果配置成0，会关闭当前TA的log
- CROSS_COMPILE
编译TA用到的工具链，开发者需要替换自己工作环境下工具链
- BINARY
当前TA生成的目标文件。注意：这个参数必须和上面定义在TA_UUID保持一致，那么对应的 BINARY就应该为：

```

1 #define TA_UUID { 0x7e5fd739, 0x7a76, 0x4391, \
2 { 0x9b, 0x99, 0xc4, 0x3a, 0x10, 0xa8, 0x17, 0xfb} }

```

代码编译

TA的编译方法为输入make命令，以demo中TA为例：

```

1 docker-ubuntu18:ta$ make
2 CC      hello_world_ta.o
3 CC      user_ta_header.o
4 CPP     ta.lds
5 GEN     dyn_list
6 LD      8aaaf200-2450-11e4-abe2-0002a5d5c51b.elf
7 OBJDUMP 8aaaf200-2450-11e4-abe2-0002a5d5c51b.dmp
8 OBJCOPY 8aaaf200-2450-11e4-abe2-0002a5d5c51b.stripped.elf
9 SIGN    8aaaf200-2450-11e4-abe2-0002a5d5c51b.ta

```

其中：

- 8aaaf200-2450-11e4-abe2-0002a5d5c51b.elf -- 当前TA的elf文件
- 8aaaf200-2450-11e4-abe2-0002a5d5c51b.dmp -- 当前TA的dump文件
- 8aaaf200-2450-11e4-abe2-0002a5d5c51b.stripped.elf -- strip之后的当前TA的elf文件
- 8aaaf200-2450-11e4-abe2-0002a5d5c51b.ta -- 目标TA文件

编译选项

默认编译选项

用户可以通过编译命令（make V=1）查看，以demo中的hello_world_ta 为例：

```
1 -march=rv64imacxtheadc -mabi=lp64 -std=gnu99 -fdiagnostics-show-option -Wall -Wcast-align -Werror-implicit-function-declaration -Wextra -Wfloat-equal -Wformat-nonliteral -Wformat-security -Wformat=2 -Winit-self -Wmissing-declarations -Wmissing-format-attribute -Wmissing-include-dirs -Wmissing-noreturn -Wmissing-prototypes -Wnested-externs -Wpointer-arith -Wshadow -Wstrict-prototypes -Wswitch-default -Wwrite-strings -Wno-missing-field-initializers -Wno-format-zero-length -Wredundant-decls -Wold-style-definition -Wstrict-aliasing=2 -Wundef -march=rv64imacxtheadc -mabi=lp64 -mmodel=medany -O0 -g3 -fpic -mstrict-align -march=rv64imacxtheadc -mabi=lp64 -MD -MF ./hello_world_ta.o.d -MT hello_world_ta.o -nostdinc -isystem /home/qbq/work/src/xuantie-tee/toolchains/riscv64/bin/./lib/gcc/riscv64-unknown-linux-gnu/10.2.0/include -I./include -I./ -DRISCV64=1 -D__LP64__=1 -DMBEDTLS_SELF_TEST -DTRACE_LEVEL=4 -I. -I/home/qbq/work/src/riscv_yocto/thead-build/light-fm/tmp-glibc/work/riscv64-oe-linux/op-tee/0.1-r0/git/export/ta-rv64/include -DCFG_TEE_TA_LOG_LEVEL=4 -DCFG_RISCV64_ta_riscv64=1 -DCFG_SYSTEM_PTA=1 -DCFG_UNWIND=1 -DCFG_TA_BGET_TEST=1 -DCFG_TA_MBEDTLS=1 -DCFG_TA_MBEDTLS_SELF_TEST=1 -DCFG_TA_MBEDTLS_MPI
```

增加编译选项

如果用户想要增加编译选项，可以在sub.mk 文件中通过以下格式增加：

```
1 cflags-"file_name"-y += "option"
```

其中：

●file_name：是指要增加编译选项的文件名

●option：是指要增加的编译选项

如，我们对hello_world_ta.c增加编译选项“-Wno-strict-prototypes”，可以在sub.mk中增加：

▼ Plain Text | 复制代码

```
1 cflags-hello_world_ta.c-y += -Wno-strict-prototypes
```

TA/CA部署

编译好的TA/CA需要拷贝到linux的文件系统中，可以通过静态编译的或者动态拷贝的方式实现。

- 静态编译
 - 将生成的TA 文件拷贝到rootfs目录/lib/optee_armtz/
 - 将生成的CA文件拷贝到rootfs目录/usr/bin
- 动态拷贝

在系统起来以后可以通过SCP和adb的方式将

 - TA文件拷贝到rootfs目录/lib/optee_armtz/
 - CA文件拷贝到rootfs目录/usr/bin

运行验证

在开发板上运行安全子系统，当系统进入kernel之后：

- 将上述生成的ta（8aaaf200-2450-11e4-abe2-0002a5d5c51b.ta） 拷贝到/lib/optee_armtz/
- 将ca 拷贝到用户目录~/

运行后得到结果如下，说明ta运行正常（注意：如果当前没有运行过tee-suppllicant 需要先执行tee-suppllicant &）

```
1 root@light-fm-linux-v0:~# ./optee_example_hello_world
2 D/TC:? 0 tee_ta_init_pseudo_ta_session:299 Lookup pseudo TA 8aaaf200-2450-11e4-abe2-0002a5d5c51b
3 D/TC:? 0 ldelf_load_ldelf:95 ldelf load address 0x40002000
4 D/LD: ldelf:134 Loading TS 8aaaf200-2450-11e4-abe2-0002a5d5c51b
5 D/TC:? 0 ldelf_syscall_open_bin:147 Lookup user TA ELF 8aaaf200-2450-11e4-abe2-0002a5d5c51b (Secure Storage TA)
6 D/TC:? 0 ldelf_syscall_open_bin:151 res=0xffff0008
7 D/TC:? 0 ldelf_syscall_open_bin:147 Lookup user TA ELF 8aaaf200-2450-11e4-abe2-0002a5d5c51b (REE)
8 D/TC:? 0 ldelf_syscall_open_bin:151 res=0
9 D/LD: ldelf:168 ELF (8aaaf200-2450-11e4-abe2-0002a5d5c51b) at 0x4007d000
10 D/TA: TA_CreateEntryPoint:39 has been called
11 D/TA: TA_OpenSessionEntryPoint:68 has been called
12 I/TA: Hello World!
13 D/TA: inc_value:105 has been called
14 I/TA: Got value: 42 from NW
15 I/TA: Increase value to: 43
16 D/TC:? 0 tee_ta_close_session:516 csess 0xff0967a0 id 1
17 D/TC:? 0 tee_ta_close_session:535 Destroy session
18 I/TA: Goodbye!
19 D/TA: TA_DestroyEntryPoint:50 has been called
20 D/TC:? 0 destroy_context:312 Destroy TA ctx (0xff096740)
21 Invoking TA to increment 42
22 TA incremented value to 43
```

参考文档

1. GlobalPlatform Device Technology TEE Client API Specification v1.0
2. GlobalPlatform Device Technology TEE Client API Specification v1.0 Errata and Precisions v2.0
3. GlobalPlatform Device Technology TEE Internal Core API Specification v1.0
4. GlobalPlatform Device Technology TEE Internal Core API Specification v1.0 Errata and Precisions v1.0
5. OPTEE官网 <https://optee.readthedocs.io/en/latest/architecture/index.html>