安全功能用户手册

版权声明

修订历史

目录

1. 系统功能信息

   1.1 版本信息

2. Global Platfrom接口兼容性

   2.1 GP APIs 版本

   2.2 TEE Internal APIs

      2.2.1 Trusted Core Framework API

      2.2.2 Trusted Storage API（目前用的是reefs）

      2.2.3 Cryptographic operation API

      2.2.4 Timer API

   2.3 TEE Client APIs

3. 安全隔离

   3.1 内存隔离

   3.2 设备隔离

4. 镜像分布

   4.1 存储空间

   4.2 DDR空间

   4.3 RPMB空间

5. 镜像烧录

   5.1 BOOTROM方式

   5.2 UBOOT方式

6. 安全启动

   6.1 非验签COT

      6.1.1 uboot文件bin生成

      6.1.2 tf.ext4生成

      6.1.3 tee.ext4文件生成

6.1.4 boot.ext4文件生成
6.1.5 rootfs.yocto.ext4文件生成
6.2 验签COT
6.2.1 uboot签名bin文件生成
6.2.2 tf签名EXT4文件生成
6.2.3 tee签名EXT4文件生成
6.2.4 boot.ext4文件生成
6.2.5 rootfs.yocto.ext4文件生成
7. 安全升级
7.1 uboot升级
7.1.1 版本规则
7.1.2 版本初始化
7.1.3 升级文件制作
7.2 TF升级
7.2.1 版本规则
7.2.2 版本初始化
7.2.2.1 测试模式
7.3.2.2 产品模式
7.2.3 升级文件制作
7.3 TEE升级
7.3.1 版本规则
7.3.2 版本初始化
7.3.2.1 测试模式
7.3.2.2 产品模式
7.3.3 升级文件制作
7.4 镜像升级
7.4.1 升级流程
7.4.2 UBOOT镜像升级
7.4.3 TF镜像升级
7.4.4 TEE镜像升级
8. 安全存储
9. 安全调试

10. 安全计算
11. 安全应用
12. 自测报告
12.1 Global Platform APIs测试
12.1.1 标准测试
12.1.2 个性测试
12.2 安全启动
12.2.1 非校验COT模式
12.2.2 校验COT模式
12.3 安全升级
12.3.1 uboot镜像升级
12.3.2 TF镜像升级
12.3.3 TEE镜像升级
12.4 多核安全访问
12.5 安全应用SDK开发
12.6 安全存储
12.6.1 创建文件
12.6.2 更新文件
12.6.3 读取文件
12.6.4 删除文件
12.6.5 重命名文件
12.6 安全算法
12.6.1 随机数
12.6.2 哈希算法
12.6.3 HMAC算法
13. yocto开发测试
14. 文档参考

版权声明

注意

您购买的产品、服务或特性等应受平头哥商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，平头哥对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

平头哥半导体有限公司 T-HEAD Semiconductor Co.,LTD

地址:杭州市余杭区向往街1122号欧美金融城(EFC)英国中心西楼T6 43层 邮编: 311121

网址:www.T-head.cn

修订历史

日期	版本	功能列表	说明	作者
20220405	v0.8	• 安全升级迭代 • 安全启动迭代 • 系统安全增强 • 支持TRNG驱动 • 新增迭代文档	• 安全升级支持防回滚（版本存在RPMB），支持国际国密验签 • 安全启动支持COT • 增强系统相关安全性，支持多核管理等相关补丁，系统隔离等 • 更新玄铁TEE移植用户手册以及安全功能用户手册	夏狼
20220305	v0.5	• 支持安全存储 • 支持安全启动 • 支持安全升级	• 安全启动不支持COT，支持所有镜像分区 • 安全升级不支持RPMB， 通过ENV变量来模拟，不支持验签 • 安全应用开发	夏狼
20220205	v0.2	• TEE基础系统 • 支持GP 标准APIs • 安全隔离	• GP标准APIs通过官方xtest工具测试 • 安全内存和安全设备通过PMP和IOPMP进行隔离	夏狼

目录

[TOC]

1. 系统功能信息

- 支持RISC-V架构的TEE安全框架，支持三层安全权限工作态(机器态、超级用户态、用户态)
- 支持原生OPTEE系统，提供安全应用开发库
- 支持Linux Kernel TEE Framework和TEE驱动
- 支持Linux supplicant deamon，提供远程服务调用
- 支持test suite(xtest)测试程序
- 支持安全隔离
- 支持默认的PTA

PTA	说明
device.pta	管理device相关信息
secstor_ta_mgmt.ta	管理TA相关操作
stats.ta	检测系统内存状态
system.pta	提供系统服务，如：派生unique key、内存映射、动态库加载

1.1 版本信息

软件	版本	备注
Linux kernel	v5.10	–
opensbi	v0.9	–
optee 和xtest	v3.15	–
Trusted firmware	v2.3	–

2. Global Platfrom接口兼容性

2.1 GP APIs 版本

Global Platform Specificaton	版本
GP_Internel_api	V1.1.2(GPD_SPE_010)
GP_Client_api	V1.0(GPD_SPE_007)

2.2 TEE Internal APIs

2.2.1 Trusted Core Framework API

函数接口	是否支持
TEE_GetPropertyAsString	✅
TEE_GetPropertyAsBool	✅
TEE_GetPropertyAsU32	✅
TEE_GetPropertyAsU6（兼容性测试不做要求）	🔲
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（xtest没有覆盖）	✅

TEE_MaskCancellation（xtest没有覆盖）	✅
TEE_CheckMemoryAccessRights	✅
TEE_SetInstanceData（xtest没有覆盖）	✅
TEE_GetInstanceData（xtest没有覆盖）	✅
TEE_Malloc	✅
TEE_Realloc	✅
TEE_Free	✅
TEE_MemMove	✅
TEE_MemCompare	✅
TEE_MemFill	✅

2.2.2 Trusted Storage API（目前用的是reefs）

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（xtest没有覆盖）	✅
TEE_InitValueAttribute（xtest没有覆盖）	✅
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	✅

2.2.3 Cryptographic operation API

函数接口	是否支持
TEE_AllocateOperation	✅
TEE_FreeOperation	✅
TEE_GetOperationInfo（xtest没有覆盖）	✅
TEE_GetOperationInfoMultiple（xtest没有覆盖）	✅
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	✅

2.2.4 Timer API

函数接口	是否支持
TEE_GetSystemTime（只支持从REE侧获取）	✅
TEE_Wait	✅
TEE_GetTAPersistentTime	✅
TEE_SetTAPersistentTime	✅
TEE_GetREETime	✅

2.3 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	✅

3. 安全隔离

3.1 内存隔离

内存隔离用于保证每个系统访问指定的地址空间，而不能越界访问其他系统的地址空间。这里的系统包括但不限于：

- REE系统
- TEE系统
- Trusted firmware系统

- Share Memory 地址空间
- Memory master设备能访问的地址空间

安全子系统通过PMP安全机制对需要保护的内存进行隔离保护，从而达到保护的内存只能被指定的系统访问。对设备内存的隔离保护操作在Trusted firmware里完成。

为了防止memory master设备能随意的访问地址空间，此行为不受PMP机制管控，所以安全子系统利用IOPMP对这些memory master设备能访问的地址空间做了限制，从而保证内存被DMA攻击。对memory master设备进行访问地址限定是在系统启动过程中初始化里完成。

安全子系统的各镜像地址空间

名称	地址空间		大小	属性
	起始地址	结束地址		
TEE-OS	0x1C00_0000	0x1DFF_FFFF	32MB	security
TEE 和REE share memory	0x1A00_0000	0x1BFF_FFFF	32MB	non-security
REE或者其它设配驱动使用	0x0400_0000	0xFEDF_FFFF	4014MB	non-security
rootfs	0x0200_0000	0x0400_0000	32MB	non-security
DTB	0x01F0_0000	0x01FF_FFFF	1MB	non-security
linux内核	0x0020_0000	0x01EF_FFFF	29MB	non-security
trustfirmware/OPENSBI	0x0000_0000	0x001F~FFFF	2MB	security

3.2 设备隔离

设备隔离用于保证某些设备只能被某个系统访问，比如密码学算法引擎只能被TEE系统访问，而不能被REE系统访问，从而防止算法执行过程中密钥被窃取。
安全子系统通过PMP安全机制对需要保护的外设进行隔离保护，从而达到保护的外设只能被指定的系统访问。对设备地址空间的隔离保护操作在Trusted firmware里完成。

安全子系统(RC0版本)保护的设备列表有：

系统Memory Usage

分享 编辑

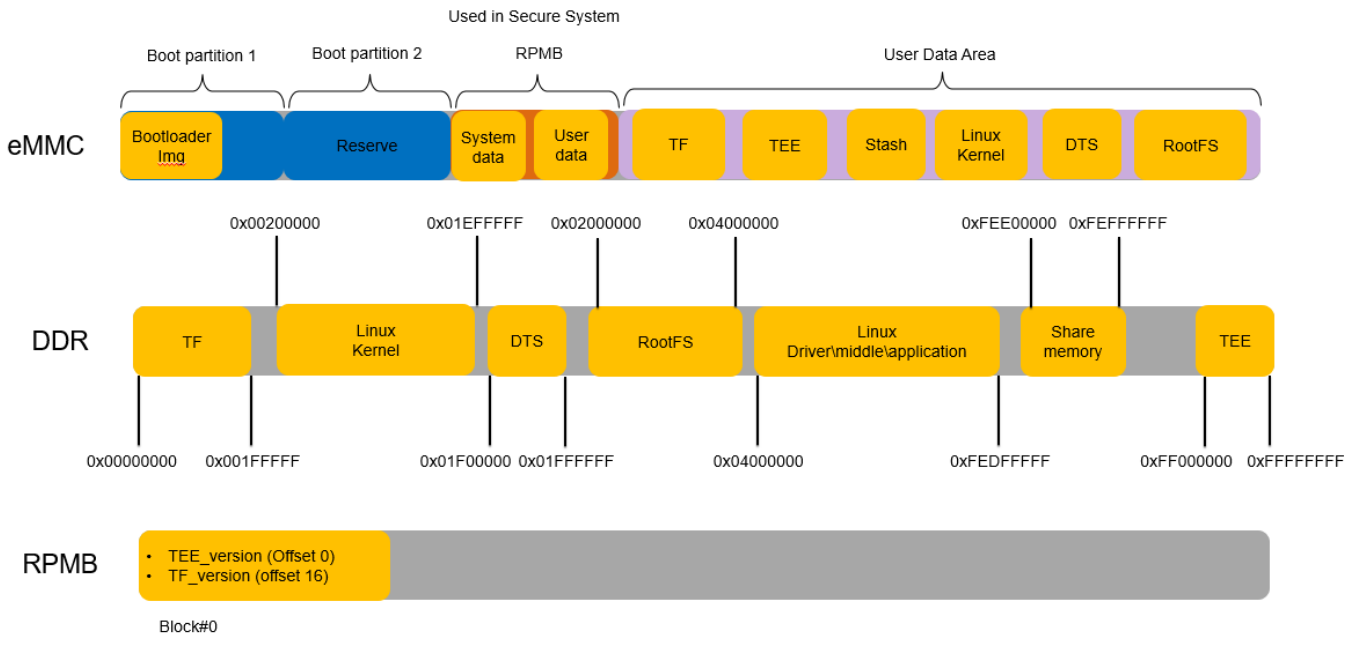
当前展示不是最新版本，可点击编辑按钮查看最新内容，并进行发布或更新

类型	名称	地址空间		大小	属性	说明
		起始地址	结束地址			
	AOSYS_T	0xFF_FFE0_0000	0xFF_FFFF_FFFF		security	
	CPUSYS_T	0xFF_FF80_0000	0xFF_FFDF_FFFF		security	
	VOSYS_T	0xFF_FF40_0000	0xFF_FF7F_FFFF		security	
	Reserved	0xFF_FF27_6000	0xFF_FF2F_FFFF		security	
	NA	0xFF_FF27_5000	0xFF_FF27_5FFF		security	
	Digital_Sensor7	0xFF_FF27_4000	0xFF_FF27_4FFF		security	
	Digital_Sensor6	0xFF_FF27_3000	0xFF_FF27_3FFF		security	
	Digital_Sensor5	0xFF_FF27_2000	0xFF_FF27_2FFF		security	
	Digital_Sensor4	0xFF_FF27_1000	0xFF_FF27_1FFF		security	
	Digital_Sensor3	0xFF_FF27_0000	0xFF_FF27_0FFF		security	
	KeyRAM	0xFF_FF26_0000	0xFF_FF26_FFFF		security	
	IOPMP_TEE_DMAC	0xFF_FF25_0000	0xFF_FF25_FFFF		security	
	IOPMP_EIP120III	0xFF_FF24_0000	0xFF_FF24_FFFF		security	
	IOPMP_EIP120II	0xFF_FF23_0000	0xFF_FF23_FFFF		security	
	IOPMP_EIP120I	0xFF_FF22_0000	0xFF_FF22_FFFF		security	
TEESYS	EFUSE_CTL	0xFF_FF21_0000	0xFF_FF21_FFFF		non-security	
	Reserved	0xFF_FF20_1000	0xFF_FF20_FFFF		security	
	TEE_SYSREG	0xFF_FF20_0000	0xFF_FF20_FFFF		security	

内存分布 同步 IO 100%

注意：efuse control在v0.2版本不支持，后续版本会支持隔离保护

4. 镜像分布



4.1 存储空间

默认开发板使用eMMC存储介质，安全子系统存储和使用用到了eMMC里的boot partiion 分区，RPMB分区和User Data Area分区。具体镜像存储位置参考上图。

我们采用EXT4的文件系统进行所有镜像管理，主要包括以下系统分区：

- uboot.bin (存在eMMC 默认的boot partition)
- tf.ext4
- tee.ext4
- stash.ext4
- boot.ext4
- rootfs.yocto.ext4

这么做的好处是uboot在加载启动镜像的时候，可以统一采用ext4load指令进行加载。

其中:

- tf分区和tee分区用于单独管理TF和TEE镜像。
- stash分区用于TF和TEE镜像的安全升级。具体参考安全升级章节。

4.2 DDR空间

系统启动后，系统镜像加载到DDR里指定的位置，通过Chain of Trust方式进行跳转执行。具体镜像运行空间如上图所示。

4.3 RPMB空间

RPMB用户存放系统敏感数据和用户数据。目前只有TEE_version和TF_version存在RPMB Block #0

- TEE_VERSION – Block#0 Offset#0
- TF_VERSION – Block#0 Offset#16

5. 镜像烧录

视语融合LIGHT芯片的烧入通过FASTBOOT工具（Window & Linux）进行烧入。平头哥提供的镜像烧录机制如下：

- 利用BOOTROM进行镜像烧写
- 利用UBOOT进行镜像烧写

5.1 BOOTROM方式

BOOTROM烧录方式也称为Imagewriter烧录方式，因为其需要下载imagewriter后，Fastboot工具通过和imagewriter进行交互下载镜像，适用于芯片第一次镜像烧写，一般是烧写完整的所有镜像文件。

操作步骤：

1. 调整板子上的BOOT SWITCH拨码开关，保证拨码开关1234均位于OFF侧，上电复位后可以在串口

看到以下信息：

```
brom_ver 7
[APP][E] protocol_connect failed, exit.
```

2. 在PC上运行light_fm_single_rank_system.bat可以直接通过fastboot进行烧写。

```
C:\WINDOWS\system32\cmd.exe
IIIIIIIIIIIIIII Run Cmd:  ".\wins_tools\fastboot flash ram u-boot-imagewriter.bin"
Sending 'ram' (676 KB)                             OKAY [  0.181s]
Writing 'ram'                                      OKAY [  0.003s]
Finished. Total time: 0.203s
IIIIIIIIIIIIIII Run Cmd:  ".\wins_tools\fastboot reboot"
Rebooting                                          OKAY [  0.001s]
Finished. Total time: 0.004s
IIIIIIIIIIIIIII Run Cmd:  ".\wins_tools\fastboot flash uboot u-boot-with-spl.bin"
Sending 'uboot' (760 KB)                           OKAY [  0.211s]
Writing 'uboot'                                    OKAY [  0.045s]
Finished. Total time: 0.297s
IIIIIIIIIIIIIII Run Cmd:  ".\wins_tools\fastboot flash tf trust_firmware.bin"
Sending 'tf' (79 KB)                               OKAY [  0.038s]
Writing 'tf'                                       OKAY [  0.006s]
Finished. Total time: 0.081s
IIIIIIIIIIIIIII Run Cmd:  ".\wins_tools\fastboot flash tee tee.bin"
Sending 'tee' (481 KB)                             OKAY [  0.139s]
Writing 'tee'                                      OKAY [  0.006s]
Finished. Total time: 0.184s
IIIIIIIIIIIIIII Run Cmd:  ".\wins_tools\fastboot flash boot boot.ext4"
^C终止批处理操作吗(Y/N)?
```

同时在串口上可以看到bootrom接收imagewriter且运行起来后，接收其他的文件的打印信息。

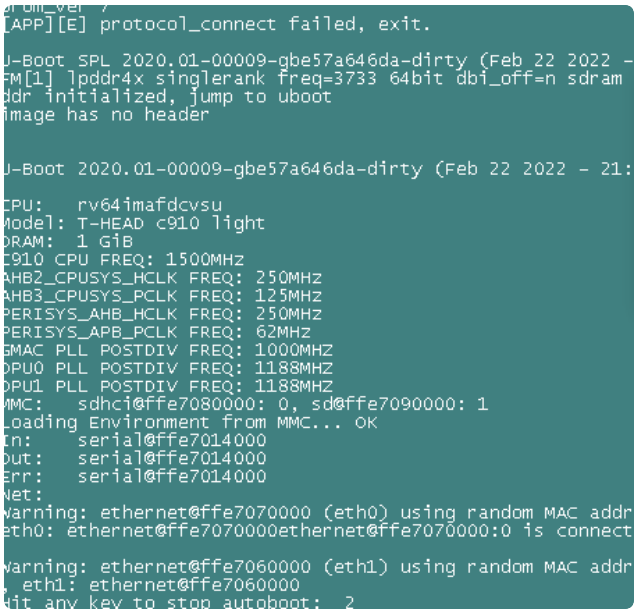
```
CPU:   rv64imafdcvsu
Model: T-HEAD c910 light
DRAM:  1 GiB
C910 CPU FREQ: 1500MHz
AHB2_CPUSYS_HCLK FREQ: 250MHz
AHB3_CPUSYS_PCLK FREQ: 125MHz
PERISYS_AHB_HCLK FREQ: 250MHz
PERISYS_APB_PCLK FREQ: 62MHz
GMAC PLL POSTDIV FREQ: 1000MHZ
DPU0 PLL POSTDIV FREQ: 1188MHZ
DPU1 PLL POSTDIV FREQ: 1188MHZ
MMC:   sdhci@ffe7080000: 0, sd@ffe7090000: 1
Loading Environment from MMC... OK
In:    serial@ffe7014000
Out:   serial@ffe7014000
Err:   serial@ffe7014000
request 00000000ffe88380 was not queued to ep1in-bulk
request 00000000ffe88380 was not queued to ep1in-bulk
request 00000000ffe88380 was not queued to ep1in-bulk
Starting download of 778312 bytes
request 00000000ffe88380 was not queued to ep1in-bulk
.....
downloading of 778312 bytes finished
request 00000000ffe88380 was not queued to ep1in-bulk

MMC write: dev # 0, block # 0, count 1521 ... 1521 blo
request 00000000ffe88380 was not queued to ep1in-bulk
request 00000000ffe88380 was not queued to ep1in-bulk
request 00000000ffe88380 was not queued to ep1in-bulk
request 00000000ffe88380 was not queued to ep1in-bulk
Starting download of 81408 bytes
request 00000000ffe88380 was not queued to ep1in-bulk

downloading of 81408 bytes finished
request 00000000ffe88380 was not queued to ep1in-bulk
request 00000000ffe88380 was not queued to ep1in-bulk
request 00000000ffe88380 was not queued to ep1in-bulk
request 00000000ffe88380 was not queued to ep1in-bulk
request 00000000ffe88380 was not queued to ep1in-bulk
Starting download of 493000 bytes
request 00000000ffe88380 was not queued to ep1in-bulk
...
downloading of 493000 bytes finished
request 00000000ffe88380 was not queued to ep1in-bulk
request 00000000ffe88380 was not queued to ep1in-bulk
request 00000000ffe88380 was not queued to ep1in-bulk
request 00000000ffe88380 was not queued to ep1in-bulk
request 00000000ffe88380 was not queued to ep1in-bulk
```

3. 调整板子上的BOOT SWITCH 拨码开关，保证拨码开关2位于ON侧，上电复位后可以看到以下串口

打印信息，表示烧写成功。



5.2 UBOOT方式

UBOOT烧录方式是指Fastboot工具和uboot进行交互完成镜像烧录的方式。一般用于升级某一个镜像文件，比如TEE， Trust firmware和uboot镜像。

操作步骤：

1. 上电复位启动后，按任意键进入uboot的命令行，串口信息如下：

2. 在串口输入"fastboot usb 0"命令，执行后uboot进入升级状态。
3. 在PC上运行light_fm_single_rank_uboot.bat, light_fm_single_rank_tf.bat,

light_fm_single_rank_tee.bat, 可以单独升级uboot, trust_firmware, tee镜像。

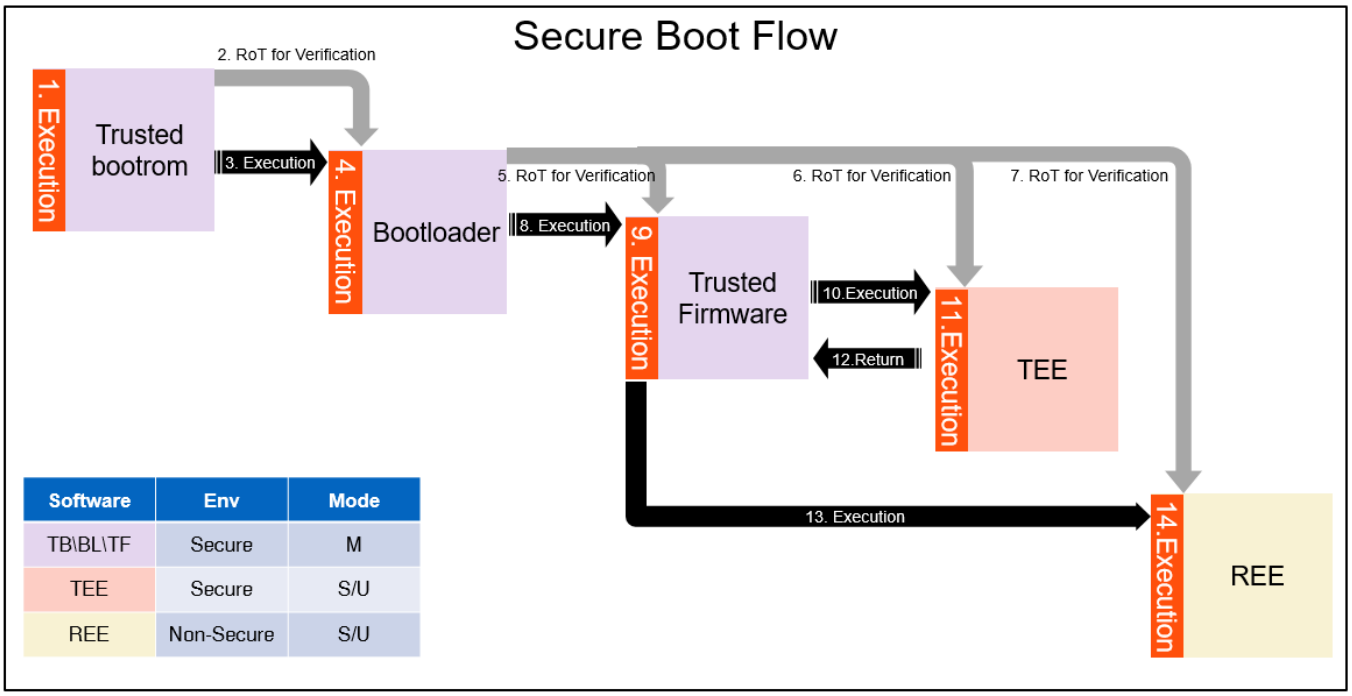
```
C:\WINDOWS\system32\cmd.exe
IIIIIIIIIIIIIIII Run Cmd:  ".\wins_tools\fastboot flash tf tf.ext4"
Sending 'tf' (120 KB)                              OKAY [  0.048s]
Writing 'tf'                                       OKAY [  0.022s]
Finished. Total time: 0.110s
请按任意键继续. . .
```

串口上会有以下显示信息，表示烧写成功。

```
C910 Light#
C910 Light# fastboot usb 0
request 00000000ffe80200 was not queued to ep1in-bulk
request 00000000ffe80200 was not queued to ep1in-bulk
request 00000000ffe80200 was not queued to ep1in-bulk
Starting download of 122880 bytes
request 00000000ffe80200 was not queued to ep1in-bulk

downloading of 122880 bytes finished
request 00000000ffe80200 was not queued to ep1in-bulk
cmd_parameter: tf, imagesize: 122880
Flashing Raw Image
........ wrote 122880 bytes to 'tf'
request 00000000ffe80200 was not queued to ep1in-bulk
```

6. 安全启动



启动镜像：

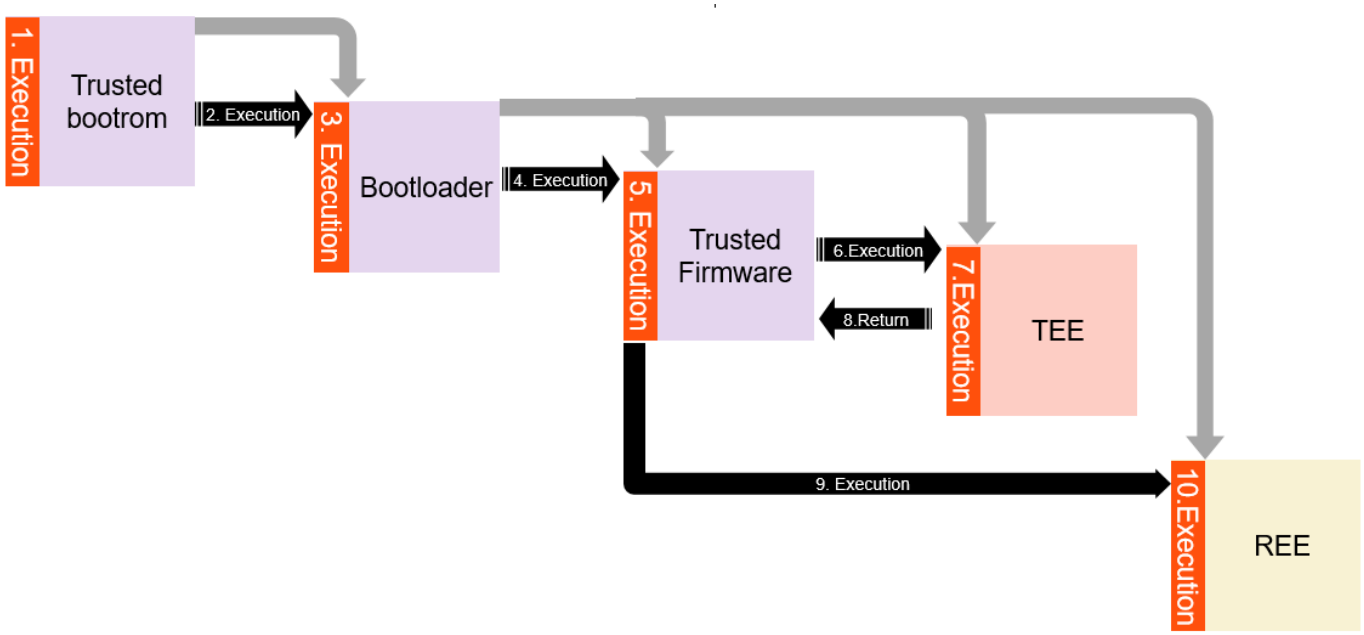
镜像类型	安全启动（验签COT）
u-boot-with-spl.bin	📎fastboot v0.8 20220325.rar
Trust_firmware.EXT4	
TEE.EXT4	
BOOT.EXT4	
rootfs.yocto.EXT4	

注意：

- 如果之前烧写过UBOOT， 需要通过以下命令在UBOOT CONSOLE下更新新的UBOOT环境变量和系统分区表。
  - env default -a -f
  - saveenv
  - run gpt_partition
- 以上镜像仅供测试使用。
- 整个启动路程为: BOOTROM->UBOOT->TF and TEE -> REE

6.1 非验签COT

非验签COT启动是指启动链路上子镜像不会被父镜像通过证书里的公钥进行验签。这种模式用于系统整体启动链路的测试，在这过程中不需要进行EFUSE烧写。启动后可以进入正常的安全子系统进行功能调用。



6.1.1 uboot文件bin生成

通过以下方式可以生成支持非验签COT的UBOOT镜像。

通过配置文件选择Light_A开发板或Light_B开发板

- light_c910_val_a_evb_sec_defconfig
- light_c910_val_b_sec_defconfig

```bash
$docker:git clone git@gitlab.alibaba-inc.com:thead-linux/u-boot.git -b light
$docker: make light_c910_val_a_evb_sec_defconfig
$docker: ./run.sh
```

详细参考安全工具用户手册里的2.4.2.1章节 – 生成uboot签名镜像。

注意：

- 在非COT模式下，uboot为了和COT模式下的运行地址做到统一，需要用 `imagesign.sh ia nor uboot v1.0` 命令进行签名后再进行烧写。
- 针对light B的板子，需要通过 `setenv fdt_file light-val-sec-b.dtb` 命令来修改需要加载的DTB文件

6.1.2 tf.ext4生成

详细参考安全工具用户手册里的2.4.3.2章节 – 生成tf.ext4

注意：

- 用于生成tf.ext4的原始镜像trust_firmware.bin不需要签名

6.1.3 tee.ext4文件生成

详细参考安全工具用户手册里的2.4.3.3章节 – 生成tee.ext4

注意：

- 用于生成tee.ext4的原始镜像tee.bin不需要签名

6.1.4 boot.ext4文件生成

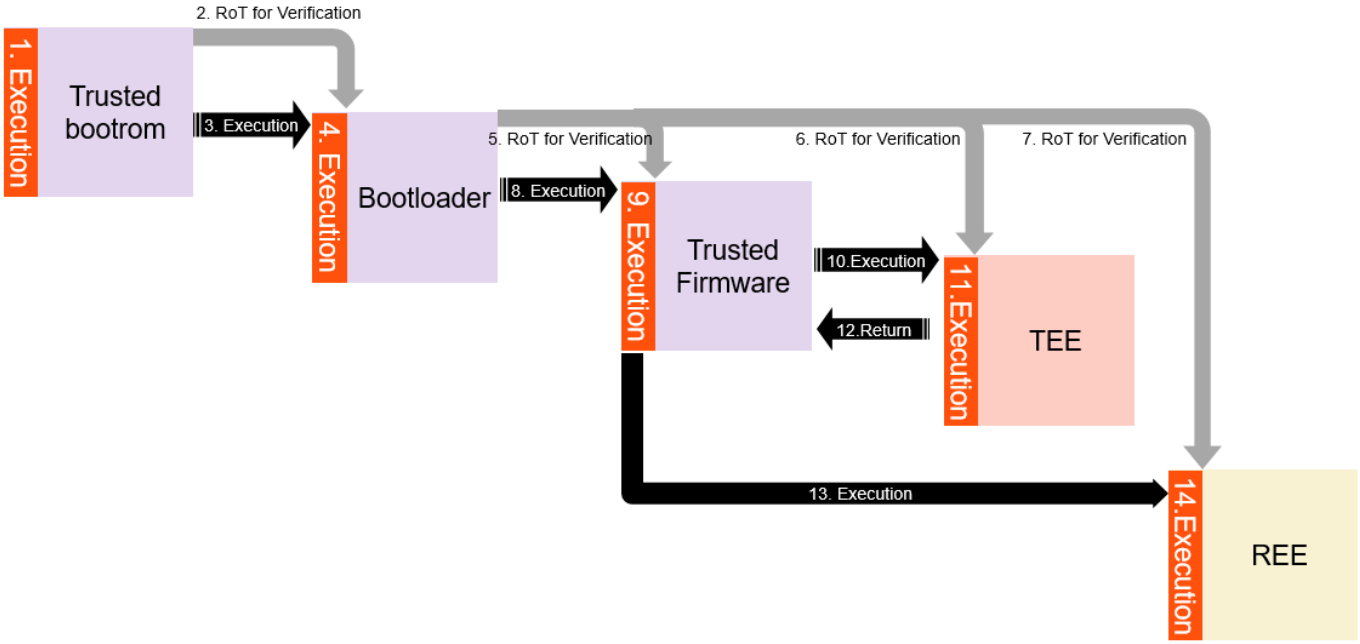
boot.ext4可以通过yocto编译打包生成，直接使用。

6.1.5 rootfs.yocto.ext4文件生成

rootfs.yocto.ext4可以通过yocto编译打包生成，直接使用。

6.2 验签COT

验签COT启动是指启动链路上子镜像被父镜像通过证书里的公钥进行验签。这种模式的系统启动，需要进行EFUSE烧写，包括安全启动相关的EFUSE字段， 以及对启动镜像要进行签名生成签名镜像。启动后可以进入正常的安全子系统进行功能调用。



6.2.1 uboot签名bin文件生成

详细参考安全工具用户手册里的2.4.2.1章节 – 生成uboot签名镜像。

注意：

- 在COT模式下，uboot必须要签名

6.2.2 tf签名EXT4文件生成

详细参考安全工具用户手册里的2.4.2.2章节 – 生成tf签名镜像。

详细参考安全工具用户手册里的2.4.3.2章节 – 生成tf.ext4

注意：

- 用于生成tf.ext4的原始镜像trust_firmware.bin必须要签名

6.2.3 tee签名EXT4文件生成

详细参考安全工具用户手册里的2.4.2.3章节 – 生成tee签名镜像。

详细参考安全工具用户手册里的2.4.3.3章节 – 生成tee.ext4

注意：

- 用于生成tf.ext4的原始镜像tee.bin必须要签名

6.2.4 boot.ext4文件生成

boot.ext4可以通过yocto编译打包生成，直接使用。

6.2.5 rootfs.yocto.ext4文件生成

rootfs.yocto.ext4可以通过yocto编译打包生成，直接使用。

7. 安全升级

7.1 uboot升级

7.1.1 版本规则

uboot镜像版本存储在eFuse空间Block10 BL1Version字段里，共有64个bit表示，一共支持64个安全版本升级。其版本规则说明如下：

- 版本取值范围(0~62)，版本范围外的数据均为无效版本。
- 版本更新每次只能加1。

注意：BL1version数据有64位组成，默认全为0， 表示初始版本v1.0。举例如下：

- bit0 = 1, 表示v2.0
- bit1 = 1, 表示v3.0
- bit2 = 1, 表示v4.0
- ...
- bit62=1, 表示v64.0

即从最低位扫描每个BIT，找到最后一个是1的BIT数，才能确认当前版本号。

当前版本	升级版本	版本升级	版本更新	升级结果	失败原因
1	2	安全版本升级	BL1Version.bit0=1	成功	–
1	3	安全版本升级	–	失败	版本跳跃
5	4	安全版本升级	–	失败	版本降级
10	11	安全版本升级	BL1Version.bit9=1	成功	–
1	1	安全版本升级	–	失败	版本一致

7.1.2 版本初始化

通过配置efuse里的BL1version字段来初始化到指定的版本。

7.1.3 升级文件制作

详细参考安全工具用户手册里的2.4.2.1章节 – 生成uboot签名镜像。
详细参考安全工具用户手册里的2.4.3.1章节 – 生成uboot.ext4文件。

7.2 TF升级

7.2.1 版本规则

TF镜像版本存储在RPMB空间Block0 offset16 TF_version字段里，可重复更新，版本格式为X.Y。其版本规则说明如下：

- X为安全版本，版本取值范围(0~255), 版本更新每次只能加1。
- Y为非安全版本，版本取值范围(0~255), 版本更新无要求。

当前版本	升级版本	版本升级	版本更新	升级结果	失败原因
1.2	1.1	非安全版本升级	TF_version=1.1	成功	–
1.2	1.5	非安全版本升级	TF_version=1.5	成功	–
3.5	4.4	安全版本升级	TF_version=4.4	成功	–
3.5	2.5	安全版本升级	TF_version=3.5	失败	版本降级
3.5	7.6	安全版本升级	TF_version=3.5	失败	版本跳级
1.0	1.0	安全版本升级	TF_version=1.0	失败	版本一致

7.2.2 版本初始化

RPMB里的数据具有防回滚，完整性和机密性。由于RPMB的访问具有安全性，需要在生产阶段烧写RPMB ACCESS KEY。

7.2.2.1 测试模式

为了开发测试的一致性，我们定义访问RPMB的KEY如下：

```c
/* the sample rpmb key is only used for testing */
const unsigned char emmc_rpmb_key_sample[32] = {0x33, 0x22, 0x11, 0x00, 0x77, 0x66, 0x55, 0x44, \
                                                0xbb, 0xaa, 0x99, 0x88, 0xff, 0xee, 0xdd, 0xcc, \
                                                0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, \
                                                0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff};
```

可以通过以下步骤将key写入RPMB，然后对TF镜像版本进行初始化。

步骤一：初始化RPMB key

```
C910 Light# mw 0x80000000 0x00112233
C910 Light# mw 0x80000004 0x44556677
C910 Light# mw 0x80000008 0x8899aabb
C910 Light# mw 0x8000000c 0xccddeeff
C910 Light# mw 0x80000010 0xFFFFFFFF
C910 Light# mw 0x80000014 0xFFFFFFFF
C910 Light# mw 0x80000018 0xFFFFFFFF
C910 Light# mw 0x8000001c 0xFFFFFFFF
```

步骤二： 烧写RPMB KEY

```
C910 Light# mmc rpmb key 0x80000000
Warning: Programming authentication key can be done only once !
         Use this command only if you are sure of what you are doing,
Really perform the key programming? <y/N> y
```

步骤三： 初始化TF镜像版本

TF_version 位于Block0 offset16， 而RPMB一个BLOCK size大小是256字节。以下命令是初始化TF_version为0x00000000.

```
C910 Light# mw 0x82000010 0x00000000
C910 Light# mmc rpmb write 0x82000000 0 1 0x80000000
C910 Light# mmc rpmb read 0x83000000 0 1
C910 Light# md 0x83000000
83000000: ffffffff ffffffff ffffffff ffffffff    ................
83000010: 00000000 ffffffff ffffffff ffffffff    ................
83000020: ffffffff ffffffff ffffffff fffffffd    ................
83000030: ffffffff ffffffff ffffffff ffffffff    ................
83000040: ffffffff ffffffff ffffffff ffffffff    ................
83000050: ffffffff ffffffff ffffffff ffffffff    ................
83000060: ffffffff ffffffff ffffffff ffffffff    ................
83000070: ffffffff ffffffff ffffffff ffffffff    ................
83000080: ffffffff ffffbfff ffffffff ffffffff    ................
83000090: ffffffff ffffffff fffffffd fffffdf     ................
830000a0: ffffffff ffffffff ffffffff ffffffff    ................
830000b0: ffffffff ffffffff ffffffff ffff7fff    ................
830000c0: ffffff7d ffffffff ffffffff ffffffff    }................
830000d0: ffffffff ffffffff ffffffff ffffffff    ................
830000e0: ffffffff ffffffff fffffdf ffffffff     ................
830000f0: ffffffff ffffffff ffffffff ffffffff    ................
```

通过回读RPMB数据，确认TF_version版本被设置成0x00000000

7.3.2.2 产品模式

参考Light芯片生产测试配置知道手册3.2章节完成RPMB ACCESS KEY烧写和RPMB内容初始化。

7.2.3 升级文件制作

详细参考安全工具用户手册里的2.4.2.2章节 – 生成trust_firmware.bin签名镜像。
详细参考安全工具用户手册里的2.4.3.2章节 – 生成tf.ext4文件。

7.3 TEE升级

7.3.1 版本规则

TEE镜像版本存储在RPMB空间Block0 offset0 TEE_version字段里，可重复更新，版本格式为X.Y。其版本规则说明如下：

- X为安全版本，版本取值范围(0~255), 版本更新每次只能加1。
- Y为非安全版本，版本取值范围(0~255), 版本更新无要求。

当前版本	升级版本	版本升级	版本更新	升级结果	失败原因
1.2	1.1	非安全版本升级	TEE_version=1.1	成功	–
1.2	1.5	非安全版本升级	TEE_version=1.5	成功	–
3.5	4.4	安全版本升级	TEE_version=4.4	成功	–
3.5	2.5	安全版本升级	TEE_version=3.5	失败	版本降级
3.5	7.6	安全版本升级	TEE_version=3.5	失败	版本跳级
1.0	1.0	安全版本升级	TEE_version=1.0	失败	版本一致

7.3.2 版本初始化

RPMB里的数据具有防回滚，完整性和机密性。由于RPMB的访问具有安全性，需要在生产阶段烧写RPMB ACCESS KEY。

7.3.2.1 测试模式

为了开发测试的一致性，我们定义访问RPMB的KEY如下：

C | 复制代码

```c
/* the sample rpmb key is only used for testing */
const unsigned char emmc_rpmb_key_sample[32] = {0x33, 0x22, 0x11, 0x00, 0x77, 0x66, 0x55, 0x44, \
                                                0xbb, 0xaa, 0x99, 0x88, 0xff, 0xee, 0xdd, 0xcc, \
                                                0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, \
                                                0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff};
```

可以通过以下步骤将key写入RPMB，然后对TF镜像版本进行初始化。

步骤一：初始化RPMB key

```
C910 Light# mw 0x80000000 0x00112233
C910 Light# mw 0x80000004 0x44556677
C910 Light# mw 0x80000008 0x8899aabb
C910 Light# mw 0x8000000c 0xccddeeff
C910 Light# mw 0x80000010 0xFFFFFFFF
C910 Light# mw 0x80000014 0xFFFFFFFF
C910 Light# mw 0x80000018 0xFFFFFFFF
C910 Light# mw 0x8000001c 0xFFFFFFFF
```

步骤二： 烧写RPMB KEY

```
C910 Light# mmc rpmb key 0x80000000
Warning: Programming authentication key can be done only once !
         Use this command only if you are sure of what you are doing,
Really perform the key programming? <y/N> y
```

步骤三： 初始化TF镜像版本

TF_version 位于Block0 offset0， 而RPMB一个BLOCK size大小是256字节。以下命令是初始化TF_version为0x00000000.

```
C910 Light# mw 0x82000000 0x00000000
C910 Light# mmc rpmb write 0x82000000 0 1 0x80000000
C910 Light# mmc rpmb read 0x83000000 0 1
C910 Light# md 0x83000000
83000000: 00000000 ffffffff ffffffff ffffffff    ................
83000010: 00000000 ffffffff ffffffff ffffffff    ................
83000020: ffffffff ffffffff ffffffff fffffffd    ................
83000030: ffffffff ffffffff ffffffff ffffffff    ................
83000040: ffffffff ffffffff ffffffff ffffffff    ................
83000050: ffffffff ffffffff ffffffff ffffffff    ................
83000060: ffffffff ffffffff ffffffff ffffffff    ................
83000070: ffffffff ffffffff ffffffff ffffffff    ................
83000080: ffffffff ffffbfff ffffffff ffffffff    ................
83000090: ffffffff ffffffff fffffffd fffffdf     ................
830000a0: ffffffff ffffffff ffffffff ffffffff    ................
830000b0: ffffffff ffffffff ffffffff ffff7fff    ................
830000c0: ffffff7d ffffffff ffffffff ffffffff    }................
830000d0: ffffffff ffffffff ffffffff ffffffff    ................
830000e0: ffffffff ffffffff fffffdf ffffffff     ................
830000f0: ffffffff ffffffff ffffffff ffffffff    ................
```

通过回读RPMB数据，确认TEE_version版本被设置成0x00000000

7.3.2.2 产品模式

参考Light芯片生产测试配置知道手册3.2章节完成RPMB ACCESS KEY烧写和RPMB内容初始化。

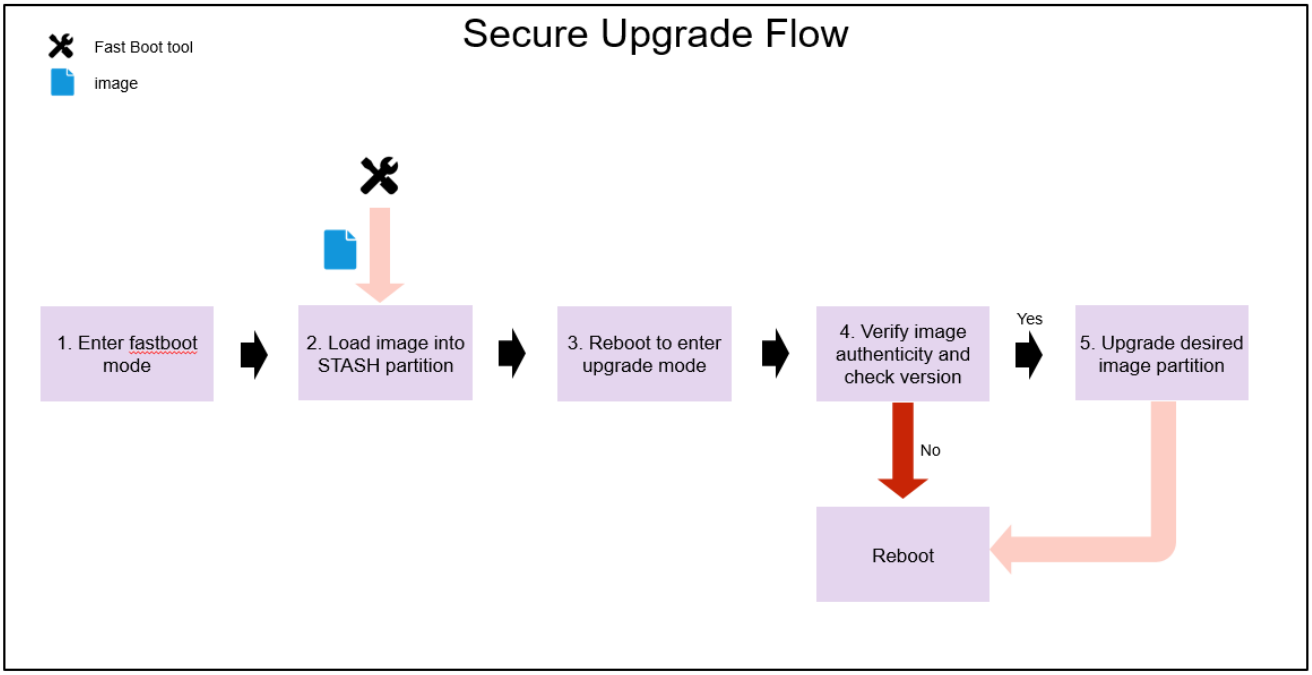
7.3.3 升级文件制作

详细参考安全工具用户手册里的2.4.2.3章节 – 生成tee.bin签名镜像。

详细参考安全工具用户手册里的2.4.332章节 – 生成tee.ext4文件。

7.4 镜像升级

7.4.1 升级流程



平头哥提供的升级方式机制统一采用uboot方式进行镜像烧录升级，如何进行uboot方式进行镜像烧写请参考5.2章节。

升级流程说明：

1. 进入uboot fastboot模式，等待上位机发送升级镜像。
2. 通过fastboot上位下发镜像，存储在stash分区，设置升级标志位后重启。
3. 系统重启后进入升级模式。
4. 系统验证镜像的真实性和完整性，以及检查版本号规则。如果不符合预期，进入6。
5. 如果符合预期，直接升级指定镜像的分区。
6. 清楚升级标志位后系统重启。

7.4.2 UBOOT镜像升级

第一步：详细参考本手册里的7.1.3章节 – 生成uboot镜像升级包uboot.ext4。

第二步：通过fastboot工具包里的light_fm_single_rank_uboot_upd.bat脚本利用UBOOT方式进行uboot镜像升级。

注意：

- 通过light_fm_single_rank_uboot.bat是可以直接对TF镜像进行更新，不会考虑版本检查等安全特性，一般适用于非安全升级。

7.4.3 TF镜像升级

第一步：详细参考本手册里的7.2.3章节 – 生成tf镜像升级包tf.ext4。

第二步：通过fastboot工具包里的light_fm_single_rank_tf_upd.bat脚本利用UBOOT方式进行TF镜像升级。

注意：

- 通过light_fm_single_rank_tf.bat是可以直接对TF镜像进行更新，不会考虑版本检查等安全特性，一般适用于非安全升级。

7.4.4 TEE镜像升级

第一步：详细参考本手册里的7.3.3章节 – 生成tee镜像升级包tee.ext4。

第二步：通过fastboot工具包里的light_fm_single_rank_tee_upd.bat脚本利用UBOOT方式进行TEE镜像升级。

注意：

- 通过light_fm_single_rank_tee.bat脚本可以直接对TEE镜像进行更新，不会考虑版本检查等安全特性，一般适用于非安全升级。

8. 安全存储

参考Global Platform storage APIs进行管理数据存储。

9. 安全调试

不支持

10. 安全计算

参考Global Platform Cryptographic APIs进行密码学算法调用。

11. 安全应用

安全子系统支持用户开发自己的TA和CA，并在release的SDK包里包含了demo作为参考模板。demo路径如下：

```
.
├── demo
│   ├── host
│   └── ta
├── doc
│   └── readme.txt
├── export
│   ├── ta-rv64
│   └── tee-client
├── pre-build
│   ├── boot
│   ├── ree-related
│   ├── tee-os
│   └── trust-firmware
└── tools

```

其中：

- demo/host 为CA的开发demo
- demo/ta 为TA的开发demo
- export/ta-rv64 为TA开发依赖的相关文件，如：头文件，库以及编译脚本
- export/tee-client 为CA开发依赖的相关文件

更详细的开发说明请参考安全应用SDK 开发手册。

12. 自测报告

12.1 Global Platform APIs测试

12.1.1 标准测试

xtest包含的测试用例位于optee_test/host/xtest目录下，其测试CA通过调用标准的Global Platform Client APIs进行开发调用，具体用例可以参考改目录下的测试源代码:

测试仓库： https://optee.readthedocs.io/en/latest/building/gits/optee_test.html

```
adbg              crypto_common.h  LICENSE        rand_stream.h
regression_4100.c  sdp_basic.c    stats.c         xtest_test.c
aes_perf.c        gp               Makefile       regression_1000.c
regression_5000.c  sdp_basic.h    stats.h         xtest_test.h
benchmark_1000.c  include          nist           regression_2000.c
regression_6000.c  sha_perf.c     xtest_helpers.c  xtest_uuid_helpers.c
benchmark_2000.c  install_ta.c     pkcs11_1000.c  regression_4000.c
```

采用默认的测试用例，xtest测试通过LOG如下：

```
26059 subtests of which 0 failed
89 test cases of which 0 failed
0 test cases were skipped
TEE test application done!
```

12.1.2 个性测试

用户可以更新或增加xtest测试用例，编译通过后通过NFS或其他办法放置到RootFS里，然后直接运行该xtest测试可执行文件。

12.2 安全启动

12.2.1 非校验COT模式

通过BOOTROM方式烧写所有镜像，执行light_fm_single_rank_system.bat烧写完所有镜像后，上电复位启动，可以启动Linux kernel表示启动成功

注意：非校验COT模式烧写的镜像不需要通过imagesign工具进行签名。同时,efuse不需要进行烧写。

12.2.2 校验COT模式

通过BOOTROM方式烧写所有镜像，执行light_fm_single_rank_system.bat烧写完所有镜像后，上电复位启动，可以启动Linux kernel表示启动成功

注意：校验COT模式烧写的镜像必须需要通过imagesign工具进行签名。同时,efuse需要进行烧写合理的安全配置参数。

12.3 安全升级

12.3.1 uboot镜像升级

uboot镜像升级通过uboot方式进行，设计利用签名工具对经镜像进行签名，然后转成EXT4格式文件。具体参考章节7.4.3 TF镜像升级。

第一步： 生成u-boot-with-spl.bin的签名版本，版本为v0.0

具体参考安全工具用户手册章节2.4.2.1 生成TF签名镜像，版本号为v0.0， 通过light_fm_single_rank_uboot.bat脚本进行烧写更新。

第二步： 初始化当前u-boot-with-spl..bin的版本为v0.0

具体参考章节7.1.2 版本初始化为v0.0

第三步：生成u-boot-with-spl..bin的签名版本，版本为v1.0

具体参考安全工具用户手册章节2.4.2.1 生成签名镜像，版本号为v1.0

第四步：升级u-boot-with-spl..bin为v1.0

执行执行light_fm_single_rank_uboot_upd.bat

12.3.2 TF镜像升级

TF镜像升级通过uboot方式进行，设计利用签名工具对经镜像进行签名，然后转成EXT4格式文件。具体参考章节7.4.3 TF镜像升级。

第一步： 生成trust_firmware.bin的签名版本，版本为v0.0

具体参考安全工具用户手册章节2.4.2.1 生成TF签名镜像，版本号为v0.0， 通过light_fm_single_rank_tf.bat脚本进行烧写更新。

第二步： 初始化当前trust_firmware.bin的版本为v0.0

具体参考章节7.2.2 版本初始化为v0.0

第三步：生成trust_firmware.bin的签名版本，版本为v1.0

具体参考安全工具用户手册章节2.4.2.1 生成TF签名镜像，版本号为v1.0

第四步：升级trust_firmware.bin为v1.0

执行执行light_fm_single_rank_tf_upd.bat

12.3.3 TEE镜像升级

TEE镜像升级通过uboot方式进行，设计利用签名工具对经镜像进行签名，然后转成EXT4格式文件。具体参考章节7.4.4 TEE镜像升级。

第一步： 生成tee.bin的签名版本，版本为v0.0

具体参考安全工具用户手册章节2.4.2.2 生成TF签名镜像，版本号为v0.0， 通过light_fm_single_rank_tee.bat脚本进行烧写更新。

第二步： 初始化当前tee.bin的版本为v0.0

具体参考章节7.3.2 版本初始化为v0.0

第三步：生成tee.bin的签名版本，版本为v1.0

具体参考安全工具用户手册章节2.4.2.1 生成TF签名镜像，版本号为v1.0

第四步：升级tee.bin为v1.0

执行执行light_fm_single_rank_tee_upd.bat

12.4 多核安全访问

TEE OS支持不同Core同时对其TA进行访问，也就是说不同的CA应用运行在不同的Core上。TEE OS可以同时处理不同Core的现场，从而完成相应的TA的调用。

在TEE OS配置里， 通过配置CFG_TEE_CORE_NB_CORE宏为4，确保TEE OS 最多支持4个Core。

通常情况下，处理器CPU Core的分配是操作系统按实际情况分配的，建议跑多个进程来确认。

Linux Kernel起来以后通过以下命令读取cpu信息以确认四个core 都已经起来。

Bash | 复制代码

```
root@light-fm-linux-v0:~# cat /proc/cpuinfo
processor       : 0
hart            : 0
isa             : rv64imafdcvsu
mmu             : sv39
cpu-freq        : 1.848Ghz
cpu-icache      : 64KB
cpu-dcache      : 64KB
cpu-l2cache     : 1MB
cpu-tlb         : 1024 4-ways
cpu-cacheline   : 64Bytes
cpu-vector      : 0.7.1

processor       : 1
hart            : 1
isa             : rv64imafdcvsu
mmu             : sv39
cpu-freq        : 1.848Ghz
cpu-icache      : 64KB
cpu-dcache      : 64KB
cpu-l2cache     : 1MB
cpu-tlb         : 1024 4-ways
cpu-cacheline   : 64Bytes
cpu-vector      : 0.7.1

processor       : 2
hart            : 2
isa             : rv64imafdcvsu
mmu             : sv39
cpu-freq        : 1.848Ghz
cpu-icache      : 64KB
cpu-dcache      : 64KB
cpu-l2cache     : 1MB
cpu-tlb         : 1024 4-ways
cpu-cacheline   : 64Bytes
cpu-vector      : 0.7.1

processor       : 3
hart            : 3
isa             : rv64imafdcvsu
mmu             : sv39
cpu-freq        : 1.848Ghz
cpu-icache      : 64KB
cpu-dcache      : 64KB
cpu-l2cache     : 1MB
```

```
46    cpu-tlb          : 1024 4-ways
47    cpu-cacheline    : 64Bytes
48    cpu-vector       : 0.7.1
49
```

12.5 安全应用SDK开发

Tsec_sdk(安全子系统SDK)集成了安全应用开发的demo。该demo给用户开发安全应用提供了参考模板，用户可以参考这个模板开发自己的安全应用。更详细的介绍可以参考安全应用SDK 开发手册：https://yuque.antfin.com/occ/rgizr3/gvzarh

编译和验证的操作方法如下：

第一步：编译CA

用进入目录./demo/host/ 后修改自己的工具链：

```
################################################################
## 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
```

然后，执行命令：

Bash | 复制代码

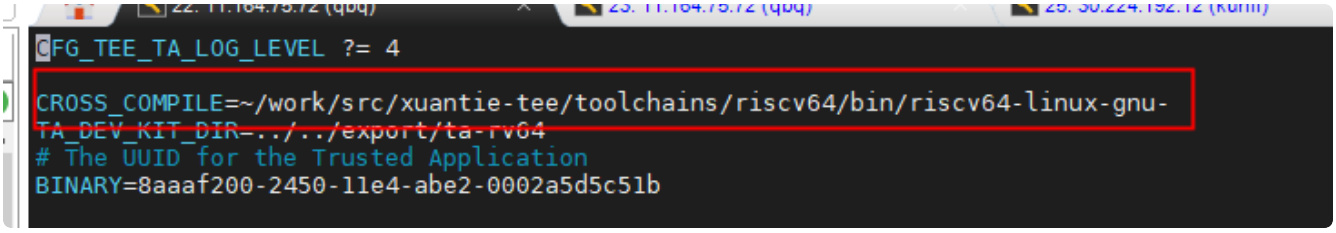
```
1    make
```

生成文件在当前目录：

```
qbq@docker-ubuntu18:host$ ll
total 36
drwxr-xr-x 2 qbq users  4096 Mar  1 17:07 ./
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  1 17:07 main.o
-rw-r--r-- 1 qbq users   957 Mar  1 15:06 Makefile
-rwxr-xr-x 1 qbq users 12576 Mar  1 17:07 optee_example_hello_world*
qbq@docker-ubuntu18:host$
```

第二步：编译TA

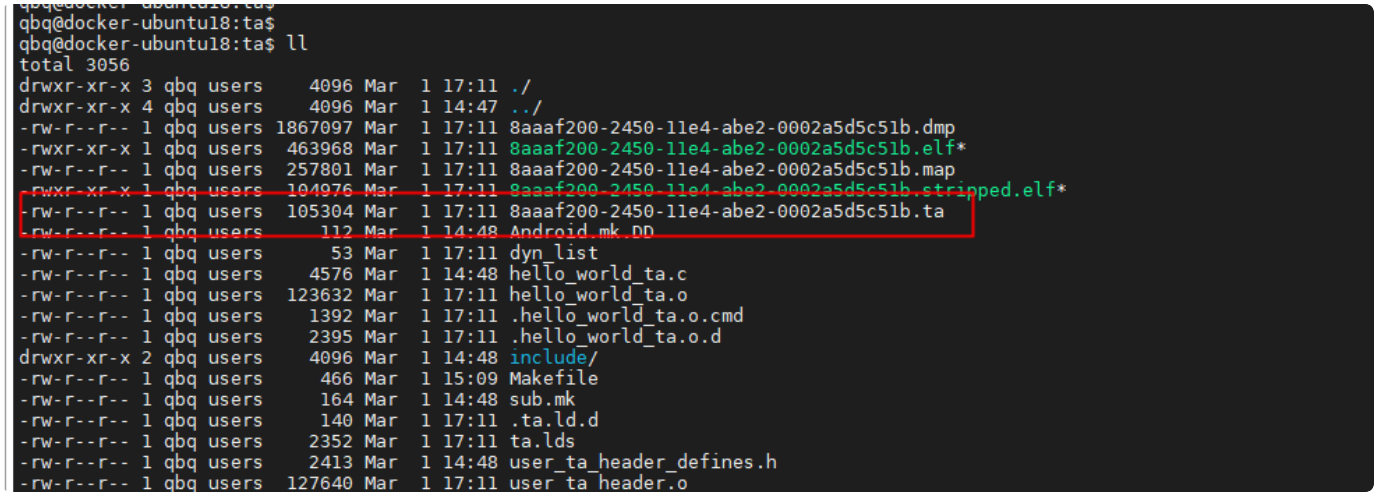
用进入目录./demo/ta/ 后修改自己的工具链：



然后，执行命令:

```bash
make
```

生成文件在当前目录：



第三步运行：

在light 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-supplicant 需要先执行tee-supplicant &）

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

```

12.6 安全存储

TODO， 介绍如何通过CA实现以下功能，贴上主要代码即可，注意格式。功能正确实现的证明说明书，比如在哪个目录下是有什么，因为我创建了XX文件，所以在某个目录下存在了XX文件。原来文件名是A，重命名B后， 在某目录下就出现了B文件。写重点 ，不要写对用户没有用的信息。

12.6.1 创建文件

```c
TEEC_Result create_secure_object(struct test_ctx *ctx, char *fileName)
{
    TEEC_Operation op;
    uint32_t origin;
    memset(&op, 0, sizeof(op));
    op.paramTypes = TEEC_PARAM_TYPES(TEEC_MEMREF_TEMP_INPUT,
                                     TEEC_MEMREF_TEMP_INPUT,
                                     TEEC_NONE, TEEC_NONE);
    ……
    op.params[0].tmpref.buffer = fileName;
    op.params[0].tmpref.size = strlen(fileName);
    //调用创建指令
    TEEC_InvokeCommand(&ctx->sess,TA_SECURE_STORAGE_CMD_CREATE,&op,
&origin);
    ……
}
```

第一步：将编写好的CA（optee_example_secure_storage）与虚拟机整体编译（make run 或make –f qemu.mk all），启动qemu虚拟机

```
QEMU 6.0.0 monitor - type 'help' for more information
(qemu) c
(qemu)
```

CA终端登录

```
Welcome to Buildroot, type root or test to login
buildroot login: root
#
```

第二步：启动CA执行创建文件

```
# optee_example_secure_storage
Prepare session with the TA

Test on object "obj1"
Create and load object in the TA secure storage
create fs : obj1 .
create obj1 success, 0x0
```

查看生成的文件是数字 2 命名的：

```
# ls /data/tee
0       1       2       dirf.db
```

调用创建文件功能指令，成功之后，传入的fileName：obj1，经过TEE处理，会在REE这边的/data/tee/目录下先生成dirf.db文件，dirf.db文件用于保存生成的安全文件的相关信息，有一个以数字（file_number属性）就是此次创建的安全文件名，这里是 2 号文件，后续也是通过先读取dirf.db文件来找到安全文件的。

12.6.2 更新文件

C 复制代码

```c
TEEC_Result write_secure_object(struct test_ctx *ctx, char *fileName,
char *data)
{
    TEEC_Operation op;
    uint32_t origin;
    memset(&op, 0, sizeof(op));
    op.paramTypes = TEEC_PARAM_TYPES(TEEC_MEMREF_TEMP_INPUT,
                                     TEEC_MEMREF_TEMP_INPUT,
                                     TEEC_NONE, TEEC_NONE);
    ……
    op.params[0].tmpref.buffer = fileName;
    op.params[0].tmpref.size = strlen(fileName);
    op.params[1].tmpref.buffer = data;
    op.params[1].tmpref.size = strlen(data);
    //调用写入文件指令
    TEEC_InvokeCommand(&ctx->sess, TA_SECURE_STORAGE_CMD_WRITE, &op,
&origin);
    ……
}
```

根据创建文件的流程，这里CA终端启动的是写文件指令：

```
Test on object "obj1"
Create and load object in the TA secure storage
create fs : obj1 .
create obj1 success, 0x0
write data : aaaaaaav in obj1.
Command write success: 0x0 / 4
```

在TEE这边根据fileName：obj1获取打开操作文件的句柄（这个过程经过了一系列的验证，最终允许获得），再通过该句柄进行数据的写入操作，每次更新安全文件会经过一系列安全操作，最终写入成功。

12.6.3 读取文件

```c
TEEC_Result read_secure_object(struct test_ctx *ctx, char *fileName,
char *data, size_t data_len)
{
    TEEC_Operation op;
    uint32_t origin;
    memset(&op, 0, sizeof(op));
    op.paramTypes = TEEC_PARAM_TYPES(TEEC_MEMREF_TEMP_INPUT,
                     TEEC_MEMREF_TEMP_OUTPUT,
                     TEEC_NONE, TEEC_NONE);
    ……
    op.params[0].tmpref.buffer = fileName;
    op.params[0].tmpref.size = strlen(fileName);
    op.params[1].tmpref.buffer = data;
    op.params[1].tmpref.size = data_len;
    //调用读取文件指令
    TEEC_InvokeCommand(&ctx->sess,TA_SECURE_STORAGE_CMD_READ,&op,
&origin);
    ……
}
```

根据创建文件的流程，这里CA终端启动的是读文件指令：

```
Read fs obj1.
Command read success: 0x0 / 4
read data : aaaaaaav from obj1.
```

在TEE这边根据fileName：obj1获取打开操作文件的句柄（这个过程经过了一系列的验证，最终允许获得），再通过该句柄进行数据的读取操作，每次更新安全文件会经过一系列安全操作，最终读取成功。

12.6.4 删除文件

```c
TEEC_Result delete_secure_object(struct test_ctx *ctx, char *fileName)
{
    EEC_Operation op;
    uint32_t origin;
    memset(&op, 0, sizeof(op));
    op.paramTypes = TEEC_PARAM_TYPES(TEEC_MEMREF_TEMP_INPUT,
                     TEEC_NONE, TEEC_NONE, TEEC_NONE);
    ……
    op.params[0].tmpref.buffer = fileName;
    op.params[0].tmpref.size = strlen(fileName);
    //调用删除文件指令
    TEEC_InvokeCommand(&ctx->sess, TA_SECURE_STORAGE_CMD_DELETE,&op,
&origin);
    ……
}
```

根据创建文件的流程，这里CA终端启动的是删除文件指令：

```
delete fs: obj1
Command delete success: 0x0 / 4

We're done, close and release TEE resources
# ls /data/tee
0       1       dirf.db
```

在TEE这边根据fileName：obj1获取打开操作文件的句柄（这个过程经过了一系列的验证，最终允许获得），再通过该句柄进行删除文件操作，通过dirf.db文件里的信息，最终成功把obj1对应的 2 号文件删除。

12.6.5 重命名文件

```c
TEEC_Result rename_secure_object(struct test_ctx *ctx, char* oldName,
char* newName)
{
    TEEC_Operation op;
    uint32_t origin;
    memset(&op, 0, sizeof(op));
    op.paramTypes = TEEC_PARAM_TYPES(TEEC_MEMREF_TEMP_INPUT,
                    TEEC_MEMREF_TEMP_INPUT, TEEC_NONE, TEEC_NONE);
    ……
    op.params[0].tmpref.buffer = oldName;
    op.params[0].tmpref.size = strlen(oldName);
    op.params[1].tmpref.buffer = newName;
    op.params[1].tmpref.size = strlen(newName);
    //调用文件重命名指令
    TEEC_InvokeCommand(&ctx->sess,TA_SECURE_STORAGE_CMD_RENAME,&op,
&origin);
    ……
}
```

根据创建文件的流程，这里CA终端启动的是文件重命名指令：

```
Command rename : 4
Rename newName obj2 success.
use newName: obj2, Read fs.
Command read success: 0x0 / 4
read data : aaaaaaav .

We're done, close and release TEE resources
# ls /data/tee
0       1       2       dirf.db
```

在TEE这边根据oldName：obj1获取打开操作文件的句柄（这个过程经过了一系列的验证，最终允许获得），再通过该句柄进行文件重命名操作，传入newName：obj2，最终成功重命名操作，可以根据通过新文件名obj2来读取数据就是原来的数据，安全文件名 2 不变。

最后：上面提到的安全存储功能，在TEE这边做了相对复杂的安全机制。如果需要进一步了解，可做进一步深入分析。比如创建安全文件的过程、创建dirf.db文件的过程，创建打开安全文件操作句柄的过程、打开dirf.db文件的过程、操作安全文件的过程、安全存储使用的KEY的产生、安全文件的格式，还有加密操作等等，这些细节上的分析。

12.6 安全算法

12.6.1 随机数

1. Enable HW RNG

- 在optee_os/core/arch/riscv/plat-light/conf.mk中修改
  - `CFG_THEAD_RAMBUS ?= y`
  - `CFG_THEAD_RAMBUS_RNG ?= y`

2. XTEST测试

Bash | 复制代码

```
root@light-fm-linux:~# tee-supplicant &
[2] 296
[1]   Done(127)               tee_supplicant
D/TC:? 0 tee_ta_init_session_with_context:611 Re-open TA 7011a688-ddde-4053-a5a9-7b3c4ddf13b8
D/TC:? 0 tee_ta_close_session:516 csess 0x1c0989f0 id 1
D/TC:? 0 tee_ta_close_session:535 Destroy session

root@light-fm-linux:~# xtest -t regression 4004
Test ID: 4004
D/TC:? 0 tee_ta_init_pseudo_ta_session:299 Lookup pseudo TA cb3e5ba0-adf1-11e0-998b-0002a5d5c51b
D/TC:? 0 ldelf_load_ldelf:95 ldelf load address 0x40002000
D/LD:  ldelf:134 Loading TS cb3e5ba0-adf1-11e0-998b-0002a5d5c51b
D/TC:? 0 ldelf_syscall_open_bin:147 Lookup user TA ELF cb3e5ba0-adf1-11e0-998b-0002a5d5c51b (Secure Storage TA)
Run test suite with level=0

TEE test application started overD/TC:? 0 ldelf_syscall_open_bin:151 res=0xffff0008
D/TC:? 0 ldelf_syscall_open_bin:147 Lookup user TA ELF cb3e5ba0-adf1-11e0-998b-0002a5d5c51b (REE)
 default TEE instance
##############################################################
#
# regression
D/TC:? 0 ldelf_syscall_open_bin:151 res=0
#
##############################################################

* regression_4004 Test TEE Internal API get random
o regression_4004.1 TEE get random
D/LD:  ldelf:168 ELF (cb3e5ba0-adf1-11e0-998b-0002a5d5c51b) at 0x4004d000
D/TC:? 0 tee_ta_close_session:516 csess 0x1c0989f0 id 1
D/TC:? 0 tee_ta_close_session:535 Destroy session
D/TC:? 0 destroy_context:312 Destroy TA ctx (0x1c098990)
  regression_4004.1 OK
  regression_4004 OK
+-----------------------------------------------------
Result of testsuite regression filtered by "4004":
regression_4004 OK
+-----------------------------------------------------
8 subtests of which 0 failed
1 test case of which 0 failed
88 test cases were skipped
```

```
41    TEE test application done!
42    [3]+  Done(127)                 tee_supplicant
```

12.6.2 哈希算法

1. Enable HW HASH

- 在optee_os/core/arc/riscv/plat-light/conf.mk中修改
  - `CFG_THEAD_RAMBUS ?= y`
  - `CFG_THEAD_RAMBUS_HASH ?= y`

2. XTEST测试

Bash | 复制代码

```
root@light-fm-linux:~# tee-supplicant &
[1] 292
D/TC:? 0 tee_ta_init_session_with_context:611 Re-open TA 7011a688-ddde-4053-a5a9-7b3c4ddf13b8
D/TC:? 0 tee_ta_close_session:516 csess 0x1c0999f0 id 1
D/TC:? 0 tee_ta_close_session:535 Destroy session
root@light-fm-linux:~# [   33.837887] lcd0_en: disabling
[   33.840962] lcd0_bias_en: disabling
[   33.844554] lcd1_en: disabling
[   33.847666] lcd1_bias_en: disabling
xtest -t regression 4004
Test ID: 4004
D/TC:? 0 tee_ta_init_pseudo_ta_session:299 Lookup pseudo TA cb3e5ba0-adf1-11e0-998b-0002a5d5c51b
D/TC:? 0 ldelf_load_ldelf:95 ldelf load address 0x40002000
D/LD:  ldelf:134 Loading TS cb3e5ba0-adf1-11e0-998b-0002a5d5c51b
D/TC:? 0 ldelf_syscall_open_bin:147 Lookup user TA ELF cb3e5ba0-adf1-11e0-998b-0002a5d5c51b (Secure Storage TA)
Run test suite wD/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000004
D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0x0
D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000004
D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0x0
ith level=0

TEE test application started over default TEE instance
#########D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000004
D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0x0
D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000004
D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0x0
D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000004
D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0x0
##############################################
#
# regression
#
#############D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000004
D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0x0
################################D/TC:? 0 ldelf_syscall_open_bin:151 res=0xffff0008
D/TC:? 0 ldelf_syscall_open_bin:147 Lookup user TA ELF cb3e5ba0-adf1-11e0-998b-0002a5d5c51b (REE)
#########
```

```
38
39  * regression_4004 Test TEE Internal API get random
40  o regression_4004.1 TEE get raD/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash
    alloc_ctx algo 0x50000004
41  D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0x0
42  D/TC:? 0 ldelf_syscall_open_bin:151 res=0
43  ndom
44  D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000004
45  D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0x0
46  D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000004
47  D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0x0
48  D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000004
49  D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0x0
50  D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000004
51  D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0x0
52  D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000004
53  D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0x0
54  D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000004
55  D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0x0
56  D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000004
57  D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0x0
58  D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000004
59  D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0x0
60  D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000004
61  D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0x0
62  D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000004
63  D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0x0
64  D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000004
65  D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0x0
66  D/LD:  ldelf:168 ELF (cb3e5ba0-adf1-11e0-998b-0002a5d5c51b) at
    0x40028000
67  D/TC:? 0 tee_ta_close_session:516 csess 0x1c0999f0 id 1
68  D/TC:? 0 tee_ta_close_session:535 Destroy session
69  D/TC:? 0 destroy_context:312 Destroy TA ctx (0x1c099990)
70    regression_4004.1 OK
71    regression_4004 OK
72  +-----------------------------------------------------
73  Result of testsuite regression filtered by "4004":
74  regression_4004 OK
75  +-----------------------------------------------------
76  8 subtests of which 0 failed
77  1 test case of which 0 failed
78  88 test cases were skipped
79  TEE test application done!
80  root@light-fm-linux:~# xtest -t regression 4001
81  Test ID: 4001
82  D/TC:? 0 tee_ta_init_pseudo_ta_session:299 Lookup pseudo TA cb3e5ba0-
    adf1-11e0-998b-0002a5d5c51b
```

```
D/TC:? 0 ldelf_load_ldelf:95 ldelf load address 0x40002000
D/LD:  ldelf:134 Loading TS cb3e5ba0-adf1-11e0-998b-0002a5d5c51b
D/TC:? 0 ldelf_syscall_open_bin:147 Lookup user TA ELF cb3e5ba0-adf1-
11e0-998b-0002a5d5c51b (Secure Storage TA)
D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000004
D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0x0
Run test suite with level=0

TD/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000004
D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0x0
D/TC:? 0 ldelf_syscall_open_bin:151 res=0xffff0008
D/TC:? 0 ldelf_syscall_open_bin:147 Lookup user TA ELF cb3e5ba0-adf1-
11e0-998b-0002a5d5c51b (REE)
EE test application started over default TEE instance
#########D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo
0x50000004
D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0x0
D/TC:? 0 ldelf_syscall_open_bin:151 res=0
################################################
#
# regression
#
########################################################

* regression_4001 Test TEE Internal API hash operations
D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000004
D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0x0
D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000004
D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0x0
D/LD:  ldelf:168 ELF (cb3e5ba0-adf1-11e0-998b-0002a5d5c51b) at
0x40064000
D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000001
D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0xFFFF0009
D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000001
D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0xFFFF0009
D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000002
D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0xFFFF0009
D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000002
D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0xFFFF0009

  regression_4001.1 OK
o regression_4001.2 HasD/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash
alloc_ctx algo 0x50000003
D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0xFFFF0009
D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000003
D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0xFFFF0009
h case 1 algo 0x50000002
```

```
125     regression_4001.2 OKD/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx
    algo 0x50000004
126   D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0x0
127   D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000004
128   D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0x0
129
130   o regression_4001.3 Hash case 2 algo 0x5000000D/TC:? 0
    drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000004
131   D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0x0
132   D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000004
133   D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0x0
134   3
135     regression_4001.3 OK
136   o regression_4001.4 HD/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx
    algo 0x50000005
137   D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0xFFFF0009
138   D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000005
139   D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0xFFFF0009
140   ash case 3 algo 0x50000004
141     regression_4001.4 D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx
    algo 0x50000006
142   D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0xFFFF0009
143   D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000006
144   D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0xFFFF0009
145   OK
146   o regression_4001.5 Hash case 4 algo 0x50000D/TC:? 0
    drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000007
147   D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0xFFFF0009
148   D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000007
149   D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0xFFFF0009
150   004
151     regression_4001.5 OK
152   o regression_4001.6D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx
    algo 0x50000007
153   D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0xFFFF0009
154   D/TC:? 0 drvcrypt_hash_alloc_ctx:18 hash alloc_ctx algo 0x50000007
155   D/TC:? 0 drvcrypt_hash_alloc_ctx:27 hash alloc_ctx ret 0xFFFF0009
156    Hash case 5 algo 0x50000005
157     regression_4001.D/TC:? 0 tee_ta_close_session:516 csess 0x1c0999f0 id
    1
158   D/TC:? 0 tee_ta_close_session:535 Destroy session
159   D/TC:? 0 destroy_context:312 Destroy TA ctx (0x1c099990)
160   6 OK
161   o regression_4001.7 Hash case 6 algo 0x50000006
162     regression_4001.7 OK
163   o regression_4001.8 Hash case 7 algo 0x50000007
164     regression_4001.8 OK
165   o regression_4001.9 Hash case 8 algo 0x50000007
```

```
  regression_4001.9 OK
  regression_4001 OK
+------------------------------------------------------
Result of testsuite regression filtered by "4001":
regression_4001 OK
+------------------------------------------------------
147 subtests of which 0 failed
1 test case of which 0 failed
88 test cases were skipped
TEE test application done!
root@light-fm-linux:~#
```

12.6.3 HMAC算法

1. Enable HW HMAC

- 在optee_os/core/arc/riscv/plat-light/conf.mk中修改
  - `CFG_THEAD_RAMBUS ?= y`
  - `CFG_THEAD_RAMBUS_MAC ?= y`

2. XTEST测试

Bash | 复制代码

```
oot@light-fm-linux:~# tee-supplicant &
[1] 281
D/TC:? 0 tee_ta_init_session_with_context:611 Re-open TA 7011a688-ddde-4053-a5a9-7b3c4ddf13b8
D/TC:? 0 tee_ta_close_session:516 csess 0x1c09a9f0 id 1
D/TC:? 0 tee_ta_close_session:535 Destroy session
root@light-fm-linux:~# xtest -t regression 4002
Test ID: 4002
D/TC:? 0 tee_ta_init_pseudo_ta_session:299 Lookup pseudo TA cb3e5ba0-adf1-11e0-998b-0002a5d5c51b
D/TC:? 0 ldelf_load_ldelf:95 ldelf load address 0x40002000
D/LD:  ldelf:134 Loading TS cb3e5ba0-adf1-11e0-998b-0002a5d5c51b
D/TC:? 0 ldelf_syscall_open_bin:147 Lookup user TA ELF cb3e5ba0-adf1-11e0-998b-0002a5d5c51b (Secure Storage TA)
Run test suite wU/TC: 000000001c09a0cc  79 b0 82 21 44 16 d9 f4  3a 9b e1 cb 6c 4a ba a5
U/TC: 000000001c09a0dc  43 5c 1a 1a 55 b8 ae 51  c3 a6 85 74 60 95 21 47
U/TC: 000000001c09a0ec  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
U/TC: 000000001c09a0fc  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
ith level=0

TEE test application started over default TEE insU/TC: 000000001c099f3c  79 b0 82 21 44 16 d9 f4  3a 9b e1 cb 6c 4a ba a5
U/TC: 000000001c099f4c  43 5c 1a 1a 55 b8 ae 51  c3 a6 85 74 60 95 21 47
U/TC: 000000001c099f5c  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
U/TC: 000000001c099f6c  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
D/TC:? 0 ldelf_syscall_open_bin:151 res=0xffff0008
D/TC:? 0 ldelf_syscall_open_bin:147 Lookup user TA ELF cb3e5ba0-adf1-11e0-998b-0002a5d5c51b (REE)
tance
##########################################################
#
# regression
#
#############D/TC:? 0 ldelf_syscall_open_bin:151 res=0
########################################

* regression_4002 Test TEE Internal API MAC operations
U/TC: 000000001c0991cc  79 b0 82 21 44 16 d9 f4  3a 9b e1 cb 6c 4a ba a5
U/TC: 000000001c0991dc  43 5c 1a 1a 55 b8 ae 51  c3 a6 85 74 60 95 21 47
U/TC: 000000001c0991ec  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
U/TC: 000000001c0991fc  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
U/TC: 000000001c09903c  79 b0 82 21 44 16 d9 f4  3a 9b e1 cb 6c 4a ba a5
U/TC: 000000001c09904c  43 5c 1a 1a 55 b8 ae 51  c3 a6 85 74 60 95 21 47
U/TC: 000000001c09905c  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
```

```
U/TC: 000000001c09906c  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
D/LD:  ldelf:168 ELF (cb3e5ba0-adf1-11e0-998b-0002a5d5c51b) at
0x40059000
o regression_4002.1 MAC case 0 algo 0x30000001
  regression_4002.1 OK
o regression_4002.2 MAC case 1 algo 0x30000001
  regression_4002.2 OK
o regression_4002.3 MAC case 2 algo 0x30000002
  regression_4002.3 OK
o regression_4002.4 MAC case 3 algo 0x30000002
  regression_4002.4 OK
o regression_4002.5 MAC case 4 algo 0x30000003
  regression_4002.5 OK
o regression_4002.6 MAC case 5 algo 0x30000003
  regression_4002.6 OK
o regression_4002.7 MAC U/TC: 000000001c09907c  51 57 45 52 54 59 00 00
 00 00 00 00 00 00 00 00
U/TC: 000000001c09908c  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
U/TC: 000000001c09909c  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
U/TC: 000000001c0990ac  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
case 6 algo 0x30000004
U/TC: 000000001c09907c  51 57 45 52 54 59 00 00  00 00 00 00 00 00 00 00
U/TC: 000000001c09908c  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
U/TC: 000000001c09909c  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
U/TC: 000000001c0990ac  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
  regression_4002.7 OK
o regression_4002.8 MAC U/TC: 000000001c09907c  51 57 45 52 54 59 00 00
 00 00 00 00 00 00 00 00
U/TC: 000000001c09908c  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
U/TC: 000000001c09909c  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
U/TC: 000000001c0990ac  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
case 7 algo 0x30000004
U/TC: 000000001c09907c  51 57 45 52 54 59 00 00  00 00 00 00 00 00 00 00
U/TC: 000000001c09908c  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
U/TC: 000000001c09909c  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
U/TC: 000000001c0990ac  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
  regression_4002.8 OK
o regression_4002.9 MAC U/TC: 000000001c09907c  6b 65 79 00 00 00 00 00
 00 00 00 00 00 00 00 00
U/TC: 000000001c09908c  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
U/TC: 000000001c09909c  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
U/TC: 000000001c0990ac  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
case 8 algo 0x30000004
U/TC: 000000001c09907c  6b 65 79 00 00 00 00 00  00 00 00 00 00 00 00 00
U/TC: 000000001c09908c  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
U/TC: 000000001c09909c  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
U/TC: 000000001c0990ac  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
  regression_4002.9 OK
```

```
 84  o regression_4002.10 MACU/TC: 000000001c09907c  6b 65 79 00 00 00 00 00
      00 00 00 00 00 00 00 00
 85  U/TC: 000000001c09908c  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
 86  U/TC: 000000001c09909c  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
 87  U/TC: 000000001c0990ac  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
 88   case 9 algo 0x30000004
 89  U/TC: 000000001c09907c  6b 65 79 00 00 00 00 00  00 00 00 00 00 00 00 00
 90  U/TC: 000000001c09908c  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
 91  U/TC: 000000001c09909c  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
 92  U/TC: 000000001c0990ac  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
 93    regression_4002.10 OK
 94  o regression_4002.11 MAC case 10 algo 0x30000005
 95    regression_4002.11 OK
 96  o regression_4002.12 MAC case 11 algo 0x30000005
 97    regression_4002.12 OK
 98  o regression_4002.13 MAC case 12 algo 0x30000006
 99    regression_4002.13 OK
100  o regression_4002.14 MAC case 13 algo 0x30000006
101    regression_4002.14 OK
102  o regression_4002.15 MAC case 14 algo 0x30000110
103    regression_4002.15 OK
104  o regression_4002.16 MAC case 15 algo 0x30000110
105    regression_4002.16 OK
106  o regression_4002.17 MAC case 16 algo 0x30000111
107    regression_4002.17 OK
108  o regression_4002.18 MAC case 17 algo 0x30000111
109    regression_4002.18 OK
110  o regression_4002.19 MAC case 18 algo 0x30000113
111    regression_4002.19 OK
112  o regression_4002.20 MAC case 19 algo 0x30000113
113    regression_4002.20 OK
114  o regression_4002.21 MAC case 20 algo 0x30000510
115    regression_4002.21 OK
116  o regression_4002.22 MAC case 21 algo 0x30000510
117    regression_4002.22 OK
118  o regression_4002.23 MAC case 22 algo 0x30000511
119    regression_4002.23 OK
120  o regression_4002.24 MAC case 23 algo 0x30000511
121    regression_4002.24 OK
122  o regression_4002.25 MAC case 24 algo 0x30000513
123    regression_4002.25 OK
124  o regression_4002.26 MAC case 25 algo 0x30000513
125    regression_4002.26 OK
126  o regression_4002.27 MAC case 26 algo 0x30000510
127    regression_4002.27 OK
128  o regression_4002.28 MAC case 27 algo 0x30000510
129    regression_4002.28 OK
130  o regression_4002.29 MAC case 28 algo 0x30000511
```

```
    regression_4002.29 OK
  o regression_4002.30 MAC case 29 algo 0x30000511
    regression_4002.30 OK
  o regression_4002.31 MAC case 30 algo 0x30000513
    regression_4002.31 OK
  o regression_4002.32 MAC case 31 algo 0x30000513
    regression_4002.32 OK
  o regression_4002.33 MAC case 32 algo 0x30000513
    regression_4002.33 OK
  o regression_4002.34 MAC case 33 algo 0x30000513
    regression_4002.34 OK
  o regression_4002.35 MAC case 34 algo 0x30000610
    regression_4002.35 OK
  o regression_4002.36 MAC case 35 algo 0x30000610
    regression_4002.36 OK
  o regression_4002.37 MAC case 36 algo 0x30000610
    regression_4002.37 OK
  o regression_4002.38 MAC case 37 algo 0x30000610
    regression_4002.38 OK
  o regression_4002.39 MAC case 38 algo 0x30000610
    regression_4002.39 OK
  o regression_4002.40 MAC case 39 algo 0x30000610
    regression_4002.40 OK
  o regression_4002.41 MAC case 40 algo 0x30000610
    regression_4002.41 OK
  o regression_4002.42 MAC case 41 algo 0x30000610
    regression_4002.42 OK
  o regression_4002.43 MAC case 42 algo 0x30000610
    regression_4002.43 OK
  o regression_4002.44 MAC case 43 algo 0x30000610
    regression_4002.44 OK
  o regression_4002.45 MAC case 44 algo 0x30000610
    regression_4002.45 OK
  o regression_4002.46 MAC case 45 algo 0x30000610
    regression_4002.46 OK
  o regression_4002.47 MAC case 46 algo 0x30000610
    regression_4002.47 OK
  o regression_4002.48 MAC case 47 algo 0x30000610
    regression_4002.48 OK
  o regression_4002.49 MAC case 48 algo 0x30000610
    regression_4002.49 OK
  o regression_4002.50 MAC case 49 algo 0x30000610
    regression_4002.50 OK
  o regression_4002.51 MAC case 50 algo 0x30000610
    regression_4002.51 OK
  o regression_4002.52 MAC case 51 algo 0x30000610
    regression_4002.52 OK
  o regression_4002.53 MAC case 52 algo 0x30000610
```

```
  regression_4002.53 OK
o regression_4002.54 MAC case 53 algo 0x30000610
  regression_4002.54 OK
o regression_4002.55 MAC case 54 algo 0x30000610
  regression_4002.55 OK
o regression_4002.56 MAC case 55 algo 0x30000610
  regression_4002.56 OK
o regression_4002.57 MAC case 56 algo 0x30000610
  regression_4002.57 OK
o regression_4002.58 MAC case 57 algo 0xf0000613
  regression_4002.58 OK
o regression_4002.59 MAC case 58 algo 0xf0000613
  regression_4002.59 OK
o regression_4002.60 MAC case 59 algo 0xf0000613
  regression_4002.60 OK
o regression_4002.61 MAC case 60 algo 0xf0000613
  regression_4002.61 OK
o regression_4002.62 MAC case 61 algo 0xf0000613
  regression_4002.62 OK
o regression_4002.63 MAC case 62 algo 0xf0000613
  regression_4002.63 OK
o regression_4002.64 MAC case 63 algo 0xf0000613
  regression_4002.64 OK
o regression_4002.65 MAC case 64 algo 0xf0000613
  regression_4002.65 OK
o regression_4002.66 MAC case 65 algo 0xf0000613
  regression_4002.66 OK
o regression_4002.67 MAC case 66 algo 0xf0000613
  regression_4002.67 OK
o regression_4002.68 MAC case 67 algo 0xf0000613
  regression_4002.68 OK
o regression_4002.69 MAC case 68 algo 0xf0000613
  regression_4002.69 OK
o regression_4002.70 MAC case 69 algo 0xf0000613
  regression_4002.70 OK
o regression_4002.71 MAC case 70 algo 0xf0000613
  regression_4002.71 OK
o regression_4002.72 MAC case 71 algo 0x30000007
  regression_4002.72 OK
o regression_4002.73 MAC case 72 algo 0x30000007
  regression_4002.73 OK
o regression_4002.74 MAC case 73 algo 0x30000007
  regression_4002.74 OK
o regression_4002.75 MAC case 74 algo 0x30000007
  regression_400D/TC:? 0 tee_ta_close_session:516 csess 0x1c09a9f0 id 1
D/TC:? 0 tee_ta_close_session:535 Destroy session
D/TC:? 0 destroy_context:312 Destroy TA ctx (0x1c09a990)
2.75 OK
```

```
  regression_4002 OK
+----------------------------------------------------
Result of testsuite regression filtered by "4002":
regression_4002 OK
+----------------------------------------------------
1988 subtests of which 0 failed
1 test case of which 0 failed
88 test cases were skipped
TEE test application done!
root@light-fm-linux:~# [   33.839807] lcd0_en: disabling
[   33.842878] lcd0_bias_en: disabling
[   33.846436] lcd1_en: disabling
[   33.849550] lcd1_bias_en: disabling

```

13. yocto开发测试

从v0.2版本，安全子系统通过yocto的release是以二进制文件的方式呈现，且此次release不带TA/CA的开发环境。用户可以通过参考“Light安全子系统快速上手指南”章节“2.2.2 硬件环境配置”和“2.2.3 操作步骤”运行release的二进制文件。

如果用户需要自行开发TA/CA甚至是TEEOS相关的代码。用户需要去申请相关仓库的权限。使用方法请参考开发文档中“Light安全子系统快速上手指南”。简单说明如下：

- TEE 仓库的下载

  请参考1.2， 1.3 章节
- yocto仓库的下载

  请参考1.4章节
- 编译运行

  请参考2.2 章节

说明：

如果用户需要了解更多yocto的相关操作命令，可以参考yocto用户手册 https://yuque.antfin-inc.com/occ/gbtat1/cat33o

14. 文档参考

- GPD_TEE_Specification https://yuque.antfin-inc.com/occ/rgizr3/ux4im7
- Light安全子系统快速上手指南 https://yuque.antfin-inc.com/occ/rgizr3/goonoo
- OPTEE 官网 https://optee.readthedocs.io/en/latest/architecture/index.html
- OPTEE 仓库 https://github.com/OP-TEE

- OPTEE_TEST:https://optee.readthedocs.io/en/latest/building/gits/optee_test.html
- yocto用户手册 https://yuque.antfin-inc.com/occ/gbtat1/cat33o