Rockchip TEE安全SDK 开发指南

发布版本：V1.87

日期：2022.06

文件密级：公开资料

瑞芯微电子股份有限公司

Fuzhou Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园A区18号

网址： www.rock-chips.com

客户服务电话： +86-591-83991906

客户服务传真： +86-591-83951833

客户服务邮箱： www.rock-chips.com

前言

概述

本文档主要介绍Rockchip TEE安全相关固件说明、TEE环境搭建、CA/TA开发测试、TA调试方法、TA签名方法以及注意事项。

读者对象

本文档（本指南）主要适用于以下工程师：

- 技术支持工程师
- 软件开发工程师

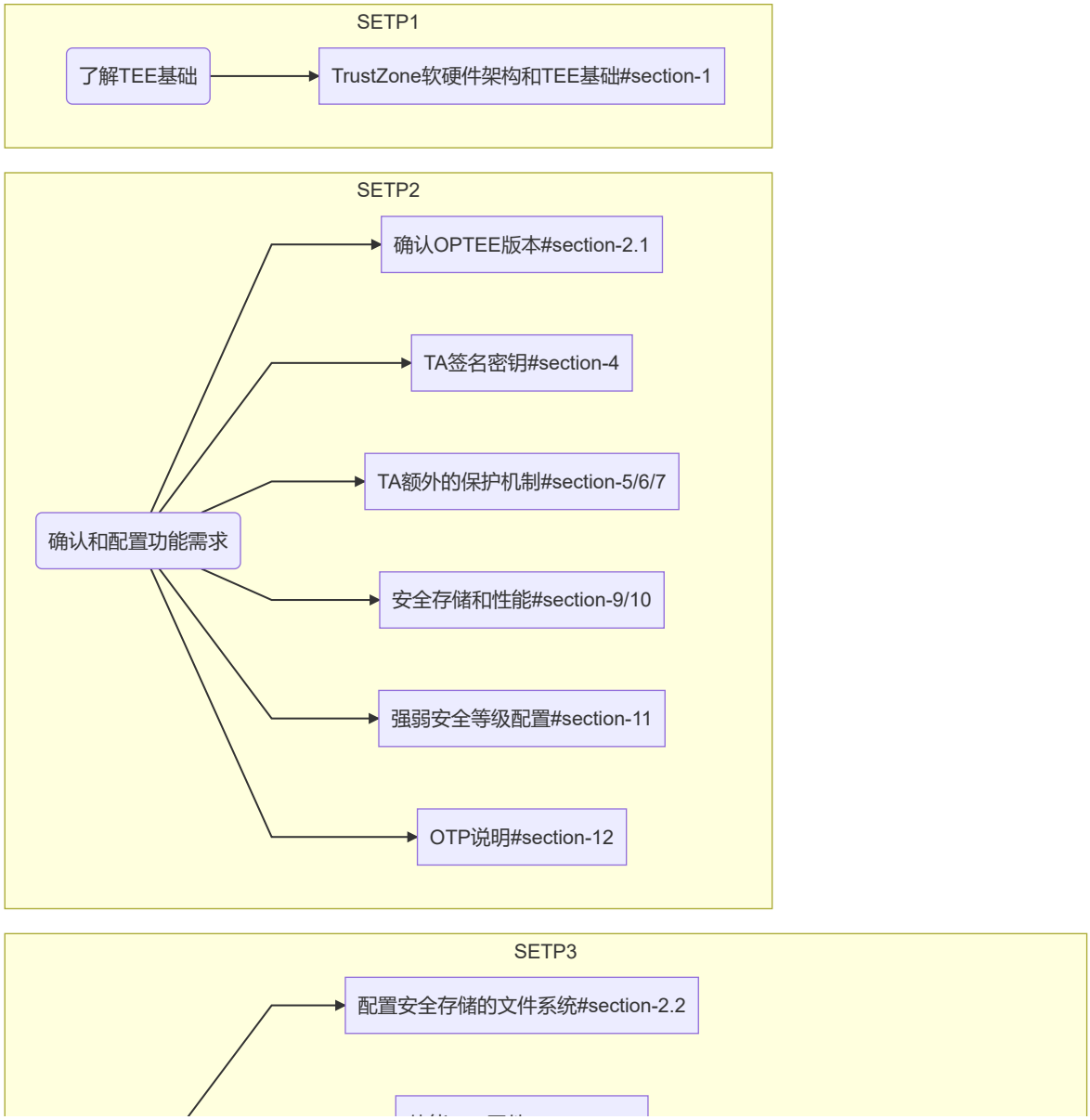
修订记录

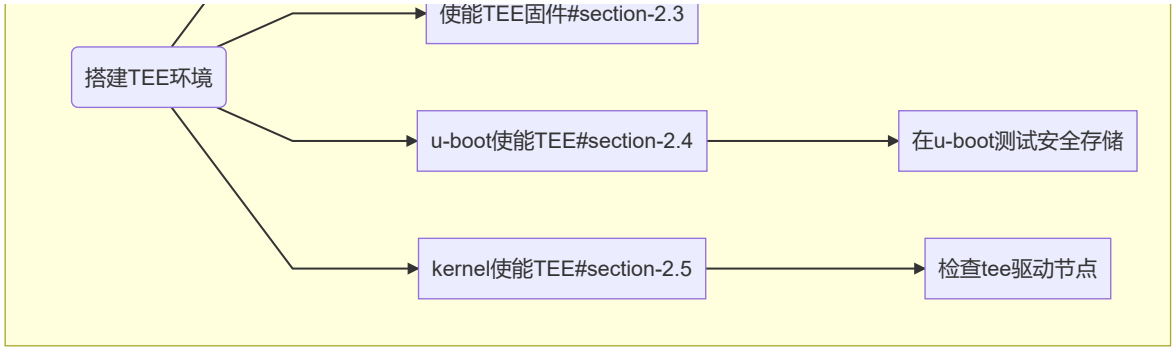
日期	版本	作者	修改说明
2018-4-26	V1.00	张志杰	初始版本
2019-3-18	V1.10	张志杰	新增uboot TEE环境说明；优化V1与V2版本区分说明
2019-6-4	V1.20	林平	新增安全存储说明
2019-7-4	V1.30	林平	修改安全存储说明
2019-7-11	V1.40	林平	新增parameter.txt说明；新增TEE相关内核节点说明
2019-8-8	V1.50	林平	新增编译rk_tee_user报错说明
2021-1-27	V1.60	林平	optee v1内核驱动变更说明
2021-3-4	V1.61	林平	新增rkfs unsupported说明
2021-5-13	V1.70	林平	新增内置TA到安全存储说明
2021-5-14	V1.71	王小滨	RK开发的CA/TA测试程序升级，对应更新文档的描述
2021-6-4	V1.72	林平	修改U-Boot中TEE相关宏说明
2021-6-4	V1.73	林平	新增加密TA说明
2021-6-17	V1.74	王小滨	新增REE FS TA的防回滚说明
2021-7-5	V1.75	林平	更新TA调试方法说明
2021-7-8	V1.76	林平	新增TA调试查看函数调用栈方法说明
2021-9-3	V1.77	林平	修改TA签名章节
2021-9-6	V1.78	林平	新增安全存储性能测试说明
2021-9-10	V1.79	张志杰	优化部分格式
2021-9-10	V1.80	王小滨	新增TA API说明章节
2021-10-12	V1.81	林平	修改安全存储性能测试说明
2021-10-15	V1.82	王小滨	在TA API章节新增API
2021-10-18	V1.83	张志杰	优化部分格式
2021-11-22	V1.84	王小滨	新增强弱安全等级可选方案的说明，更新CA/TA相关描述
2021-11-26	V1.85	王小滨	新增OTP说明章节，更新OTP API, 调整CA/TA等描述
2021-11-30	V1.86	王小滨	新增阅读指引
2022-06-22	V1.87	林平	新增 rk_tee_service 章节

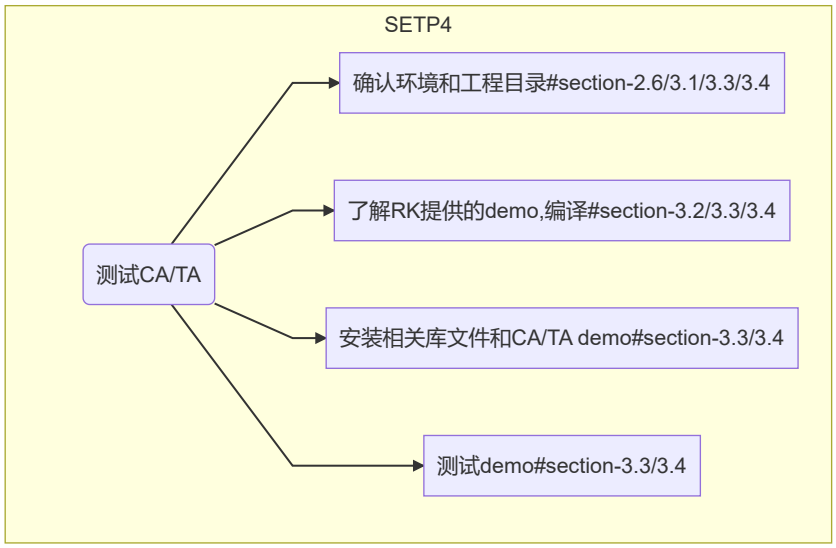
Rockchip TEE安全SDK 开发指南

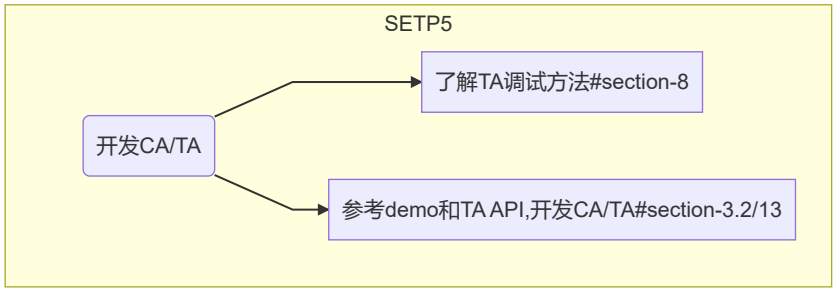
1. 阅读指引

下图介绍了文档结构，可作为开发者的阅读指引。









2. TrustZone简介

2.1 什么是TrustZone

ARM TrustZone技术是系统范围的安全方法，针对高性能计算平台上的大量应用，包括安全支付、数字版权管理(DRM)、企业服务和基于Web的服务。

TrustZone技术与Cortex™-A处理器紧密集成，并通过AMBA-AXI总线和特定的TrustZone系统IP块在系统中进行扩展。此系统方法意味着可以保护安全内存、加密块、键盘和屏幕等外设，从而可确保它们免遭软件攻击。

按照TrustZone Ready Program建议开发并利用TrustZone技术的设备提供了能够支持完全可信执行环境(TEE)以及安全感知应用程序和安全服务的平台。

智能手机和平板电脑等最新设备为消费者提供了基于扩展服务集的高价值体验，移动设备已发展为能够从Internet下载各种大型应用程序的开放软件平台。这些应用程序通常由设备OEM进行验证以确保质量，但并非可对所有功能进行测试，并且攻击者正在不断创建越来越多以此类设备为目标的恶意代码。

同时，移动设备处理重要服务的需求日益增加。从能够支付、下载和观看某一特定时段的最新好莱坞大片，到能够通过手机远程支付帐单和管理银行帐户，这一切都表明，新的商业模式已开始出现。

这些发展趋势已使手机有可能成为恶意软件、木马和rootkit等病毒的下一软件攻击目标。但是，通过应用基于ARM TrustZone技术的高级安全技术并整合SecurCore™防篡改元素，可开发出能够提供功能丰富的开放式操作环境和强大安全解决方案的设备。

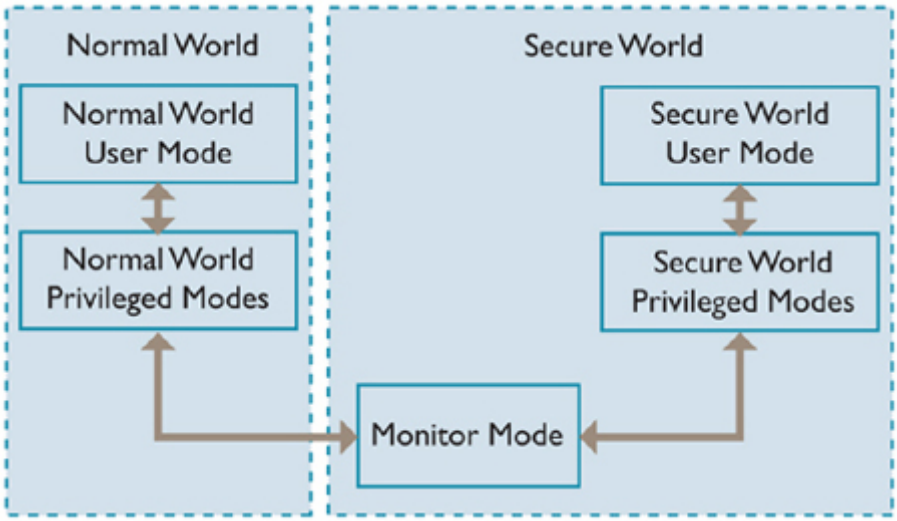
可信应用程序采用基TrustZone技术的SoC（运行可信执行环境），与主OS分开，可防止软件/恶意软件攻击。TrustZone可切换到安全模式，提供硬件支持的隔离。可信应用程序通常是可集装箱化的，如允许不同支付公司的可信应用程序共存于一台设备上。处理器支持ARM TrustZone技术是所有Cortex-A类处理器的基本功能，是通过ARM架构安全扩展引入的。这些扩展可在供应商、平台和应用程序中提供一致的程序员模型，同时提供真实的硬件支持的安全环境。

2.2 TrustZone软硬件架构

2.2.1 硬件架构

TrustZone硬件架构旨在提供安全框架，从而使设备能够抵御将遇到的众多特定威胁。TrustZone技术可提供允许SoC设计人员从大量可在安全环境中实现特定功能的组件中进行选择的基础结构，而不提供固定且一成不变的安全解决方案。

架构的主要安全目标是支持构建可编程环境， 以防止资产的机密性和完整性受到特定攻击。具备这些特性的平台可用于构建一组范围广泛的安全解决方案，而使用传统方法构建这些解决方案将费时费力。



可通过以下方式确保系统安全：隔离所有SoC硬件和软件资源，使它们分别位于两个区域（用于安全子系统的安全区域以及用于存储其他所有内容的普通区域）中。支持TrustZone的AMBA3 AXI™总线构造中的硬件逻辑可确保普通区域组件无法访问安全区域资源，从而在这两个区域之间构建强大边界。将敏感资源放入安全区域的设计，以及在安全的处理器内核中可靠运行软件可确保资产能够抵御众多潜在攻击，包括那些通常难以防护的攻击（例如，使用键盘或触摸屏输入密码）。通过在硬件中隔离安全敏感的外设，设计人员可限制需要通过安全评估的子系统的数目，从而在提交安全认证设备时节省成本。

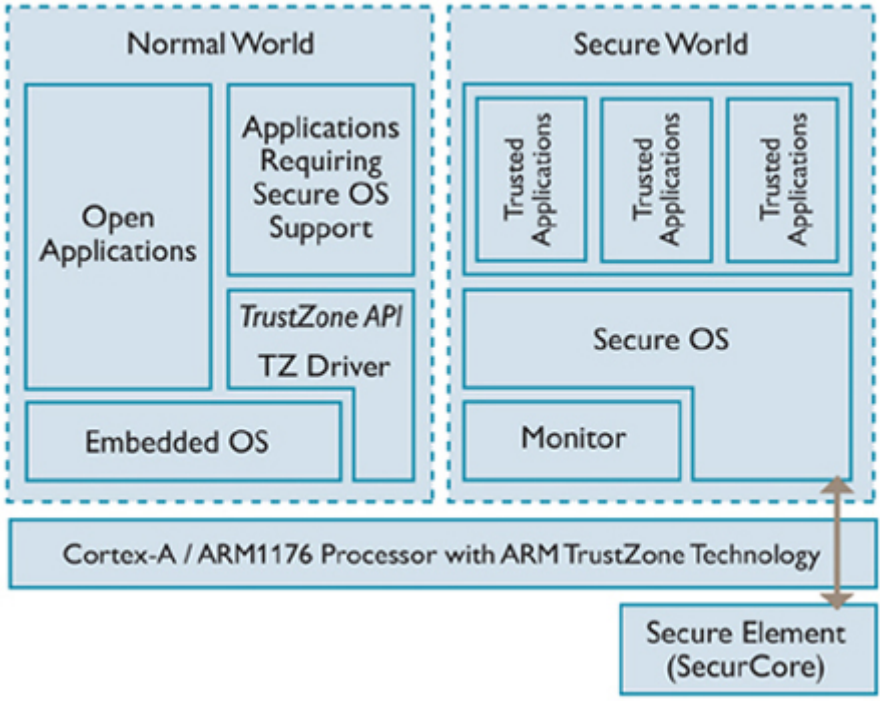
TrustZone硬件架构的第二个方面是在一些ARM处理器内核中实现的扩展。通过这些额外增加的扩展，单个物理处理器内核能够以时间片的方式安全有效地同时从普通区域和安全区域执行代码。这样，便无需使用专用安全处理器内核，从而节省了芯片面积和能源，并且允许高性能安全软件与普通区域操作环境一起运行。

更改当前运行的虚拟处理器后，这两个虚拟处理器通过新处理器模式（称为监视模式）来进行上下文切换。

物理处理器用于从普通区域进入监视模式的机制受到密切控制， 并且这些机制始终被视为监视模式软件的异常。要监视的项可由执行专用指令（安全监视调用 (SMC)指令）的软件触发，或由硬件异常机制的子集触发。可对IRQ、FIQ、外部数据中止和外部预取中止异常进行配置，以使处理器切换到监视模式。

在监视模式中执行的软件是实现定义的，但它通常保存当前区域的状态，并还原将切换到的区域位置的状态。然后，它会执行从异常返回的操作，以在已还原区域中重新启动处理过程。TrustZone硬件架构的最后一个方面是安全感知调试基础结构，它可控制对安全区域调试的访问，而不会削弱普通区域的调试可视化。

2.2.2 软件架构



在SoC硬件中实现安全区域要求在其中运行某些安全软件，并利用存储在其中的敏感资产。

可能有许多支持TrustZone的处理器内核上的安全区域软件堆栈可实现的软件架构。最高级的软件架构是专用安全区域操作系统；最简单的是放置在安全区域中的同步代码库。这两个极端架构之间有许多中间选项。

专用安全内核可能是一种复杂但强大的设计。它可模拟多个独立安全区域应用程序的并发执行、新安全应用程序的运行时下载以及完全与普通区域环境独立的安全区域任务。

这些设计与将在SoC中看到的软件堆栈非常类似，它们在非对称多处理(AMP)配置中使用两个单独的物理处理器。在每个虚拟处理器上运行的软件是独立的操作系统，并且每个区域使用硬件中断来抢占当前运行的区域和获得处理器时间。

使用将安全区域任务与请求这些任务的普通区域威胁相关联的通信协议的紧密集成设计可提供对称多处理(SMP)设计的许多优点。例如，在这些设计中，安全区域应用程序可继承它支持的普通区域任务的优先级。这将导致对媒体应用程序做出某些形式的软实时响应。

安全扩展是ARM架构的开放式组件，因此任何开发人员都可创建自定义安全区域软件环境，以满足其要求。

2.2.3 TrustZone与TEE

支付、网上银行、内容保护和企业身份验证之类的应用可通过利用TrustZone技术增强型设备所提供的三个关键要素来提高其完整性、功能和用户体验：

1. 面向软件的安全执行环境，可防止从富操作系统发起恶意软件攻击
2. 已知良好的硬件信任根，可在富操作领域检查数据和应用程序的完整性，确保安全环境不受到损害
3. 按需访问安全外设，如内存、键盘/触摸屏，甚至显示器

基于ARM TrustZone技术的设备与开放API相结合，提供了可信执行环境(TEE)，开发人员需要通过一种新型软件才能实现其功能和一致性：这种软件就是可信应用程序。典型可信应用程序可在普通区域和安全区域各包含部分代码，例如，处理关键存储和操控。TEE还提供了与其他可信应用程序的隔离，使多个可信服务可以共存。

TEE API的标准化（由GlobalPlatform管理）将会使服务提供商、运营商和OEM的可互操作可信应用程序和服务实现市场化。

ARM TrustZone技术无需单独的安全硬件来验证设备或用户的完整性。它通过在主手机芯片集中提供真正的硬件信任根来实现这一点。

为确保应用程序的完整性，TrustZone还提供了安全执行环境（即可信执行环境 (TEE)），在此环境中只有可信应用程序才能运行，从而防止遭到黑客/病毒/恶意软件形式的攻击。

TrustZone硬件提供了TEE与软件攻击媒介的隔离。硬件隔离可扩展为保护一直到物理外设（包括键盘/触摸屏等）的数据输入和输出。

正是具备了这些关键功能，采用TrustZone技术的芯片集提供了众多机会来重新定义用户可以访问的服务（更多、更好的服务），如何访问服务（更快、更轻松）以及在何处访问服务（随时随地）。

在大多数 Android设备上，Android Boot加载程序都不会验证设备内核的真实性。希望进一步控制其设备的用户可能会安装破解的Android内核来对设备进行root。破解的内核可让超级用户访问所有数据文件、应用程序和资源。一旦破解内核损坏，则会导致服务被拒绝。如果内核包含恶意软件，则将危害企业数据的安全性。

而Secure Boot可有效防止上述问题，Secure Boot是一种安全机制，它可防止在启动过程中加载未经授权的启动加载程序和内核。由值得信任的已知权威机构以加密方式签名的固件映像（如操作系统和系统组件）会被视为经过授权的固件。安全启动组件可以形成第一道防线，用以防范恶意软件对设备进行攻击。

3. TEE环境

3.1 平台说明

Rockchip平台中Android 7.1及更高版本SDK默认均支持TEE环境，Android7.1以下版本默认不支持TEE环境。Linux版本SDK默认不支持TEE环境。

Rockchip平台采用的TEE方案为OP-TEE，TEE API符合GlobalPlatform标准。

目前运行在rockchip平台上的OP-TEE有两个版本，OP-TEE V1与OP-TEE V2。

1.OP-TEE V1：RK312x、RK322x、RK3288、RK3328、RK322xh、RK3368、RK3399、RK3399Pro。

2.OP-TEE V2：RK3326、RK3308、RK1808、RV1109/RV1126、RK3566/RK3568和后续新平台。

两个版本在**TEE**库文件、**TA**文件、**Secure OS**固件方面均有不同，需根据具体平台采用不同版本**TEE**相关组件。

3.2 Parameter.txt说明

Parameter.txt文件记录了各镜像及分区的位置与大小信息，Rockchip的OP-TEE目前同时支持security与rpmb两种安全存储文件系统，具体使用哪种文件系统由TA中设置storageID参数来决定，若parameter.txt中没有定义security分区，则TA无法使用security安全存储文件系统，设置security分区的方法只需在parameter.txt中添加0x00002000@0x000xxxxx(security)即可，0x00002000表示大小4M，0x000xxxxx表示起始地址，具体根据实际的parameter.txt来修改。

3.3 TEE固件

TEE Secure OS的源码默认不开源，binary位于目录`u-boot/tools/rk_tools/bin`或`rkbin/bin`下。

1. ARMv7平台的TEE binary由工具`u-boot/tools/loaderimage`打包成固件trust.img，TEE binary的命名如下：

   ```
   <platform>_tee_[ta]_<version>.bin
   ```

   名称中带ta的为支持外部TA运行，不带ta则不支持运行外部TA。

2. ARMv8平台的TEE binary由工具`u-boot/tools/trust_merger`将BL31/BL32等bin打包成固件trust.img，TEE binary的命名如下：

   ```
   <platform>_bl32_<version>.bin
   ```

3. 若rkbin/RKTRUST/.ini中[BL32_OPTION]下SEC=0，则需要将其改成SEC=1，否则trust.img将不包含Secure OS，无法运行TEE相关服务。

3.4 U-Boot 中TEE驱动

目前一些安全的操作需要在U-Boot这级操作，比如读取一些数据必须需要OP-TEE帮忙获取。U-Boot里面实现了OP-TEE Client代码，可以通过该接口与OP-TEE通信。OP-TEE Client驱动在lib/optee_client目录下，API符合GP规范。目前尚不支持开发者在U-Boot中开发自己的CA/TA应用。

3.4.1 宏定义说明

`CONFIG_OPTEE_CLIENT`，U-Boot调用OP-TEE总开关。

`CONFIG_OPTEE_V1`，采用OP-TEE V1的平台使用。

`CONFIG_OPTEE_V2`，采用OP-TEE V2的平台使用。

`CONFIG_OPTEE_ALWAYS_USE_SECURITY_PARTITION`， 不开启该宏则安全存储会根据硬件来选择安全存储区域，emmc安全存储数据到rpmb，nand安全存储数据到security分区；开启该宏后安全存储数据固定存于security分区，不使用rpmb；使用rpmb安全性较security要高，但是若生产过程会更换设备emmc，由于rpmb的特性，rpmb中的数据是无法清除的（即使重烧固件），这会一定程度上影响工厂生产的便利性，是否开启该宏请根据便利性与安全性自行权衡；部分平台会默认开启该宏，若对安全性要求较高可自行去除该宏。

3.4.2 共享内存说明

U-Boot与OP-TEE通信时，数据需放在共享内存中，可以通过`TEEC_AllocateSharedMemory()`来申请共享内存，但各个平台共享内存大小不同，建议不超过1M，若超过则建议分割数据多次传递，使用完需调用`TEEC_ReleaseSharedMemory()`释放共享内存。

3.4.3 安全存储功能测试

3.4.3.1 测试步骤

按以下步骤，可执行安全存储功能测试。该测试用例将循环测试安全存储读写功能，测试程序会自动检查硬件，当硬件使用emmc时将测试rpmb与security分区两种安全存储方式，当硬件使用nand时只测试security分区安全存储。

1. 进入U-Boot串口命令行：设备串口连接PC，PC按住ctrl+c，启动设备，启动时设备检测到ctrl+c操作则会停在uboot。
2. 执行：以下指令启动测试。

```
=> mmc testsecurestorage
```

3.4.3.2 常见错误排查

```
"TEEC: Could not find device"
```

没有找到emmc或者nand设备，请检查U-Boot中驱动，或者硬件是否损坏。

```
"TEEC: Could not find security partition"
```

当采用security分区安全存储时，加密数据会存储在该分区，请检查parameter.txt中是否定义了security分区。

```
"TEEC: verify [%d] fail, cleanning ...."
```

第一次使用security分区进行安全存储时，或者security分区数据被非法篡改时出现，security分区会全部清空。

```
"TEEC: Not enough space available in secure storage !"
```

安全存储的空间不足，请检查存储的数据是否过大，或者存储的文件数量过多，或者之前是否存储过大量的数据而没有删除。

```
INF [0x0] TEE-CORE:storage_read_obj:201: Warning! head data not find!
ERR [0x0] TEE-CORE:storage_read_obj:210: cpu or emmc was replaced!
```

U-Boot启动过程中关键数据会调用TEE进行安全存储，关键数据经过TEE加密存储在security分区或者rpmb分区，而且加密密钥与CPU绑定；若更换CPU将导致关键数据无法正常解密，导致U-Boot启动失败；若更换emmc且emmc之前被使用过，security分区或者rpmb分区存在旧数据，也会出现该错误导致U-Boot启动失败；解决办法为需要清空security分区或者rpmb分区中的旧数据，若使用的是security分区则直接格式化emmc即可，若是使用rpmb则需要联系技术支持提供特殊固件清除rpmb中旧数据。

```
“optee check api revision fail”
```

U-Boot与TEE版本不匹配，U-Boot版本高于TEE版本，解决办法如下（二选一）：

1. 回退U-Boot版本至`cf13b78438 (tag: android-10.0-mid-rkr9) rockchip: spl: add rollback index check with otp`。
2. revert以下几个提交：

   `396e3049bd  rochchip: board: only map op-tee share memory as dcache enabled`

   `7a349fdcbd lib: optee_client: add optee initialize flag`

   `74eb602743 lib: optee_client: update to new optee msg for optee v1 platform`

   `102dfafc4a rochchip: board: map op-tee memory as dcache enabled`

正常情况下对外释放的SDK版本都是匹配的。

```
"optee api revision mismatch with u-boot/kernel, panic"
```

若在U-Boot启动阶段打印，则是U-Boot版本与TEE版本不匹配，U-Boot版本低于TEE版本，可升级U-Boot版本至`396e3049bd (tag: android-10.0-mid-rkr11, tag: android-10.0-mid-rkr10) rochchip: board: only map op-tee share memory as dcache enabled`及以上版本。

若在Android系统启动阶段打印，则升级`android/vendor/rockchip/common`版本至`8bc7bf97 (tag: android-10.0-mid-rkr10) vpu: librockit: add Rockit MetadataRetriever`及以上版本。

若在Linux系统启动阶段打印，则升级`linux/external/security/bin`版本至`f59085c optee_v1: lib: arm&arm64: update binary and library`及以上版本。

正常情况下对外释放的SDK版本都是匹配的。

3.5 TEE linux kernel驱动

TEE linux kernel驱动位于`security/optee_linuxdriver/`与`drivers/tee/`中。

3.5.1 OP-TEE V1

采用OP-TEE V1的芯片的驱动位于`security/optee_linuxdriver/`，默认均有开启。开启方法如下：

config中添加以下配置：

```
CONFIG_TEE_SUPPORT=y
```

目前我们将逐步废弃OP-TEE V1的TEE linux kernel驱动，OP-TEE V1平台将使用OP-TEE V2的TEE linux kernel驱动，若rkbin/bin目录下TEE binary文件名中 version >= v2.00，则还需要开启OP-TEE V2的TEE linux kernel驱动。

2020年8月份以后释放的Android10以及以上版本和Linux版本默认使用OP-TEE V2的kernel驱动。

3.5.2 OP-TEE V2

采用 OP-TEE V2 的芯片的驱动位于 `drivers/tee/` 下，开启方法如下：
确认对应平台 dtsi 中添加了如下节点：

```
firmware {
        optee: optee {
            compatible = "linaro,optee-tz";
            method = "smc";
            #status = "disabled";
        };
    };
```

各平台默认情况下都有添加该节点，但部分平台会设置 **status = "disabled";** 导致该驱动默认关闭，所以如果要开启optee驱动，只要去除 **status = "disabled";** 即可。

config 中添加以下两个配置：

```
CONFIG_TEE=y
CONFIG_OPTEE=y
```

3.5.3 确认驱动开启

若出现 `/dev/opteearmtz00` 节点，说明optee v1的TEE linux kernel驱动已开启；
若出现 `/dev/tee0` 和 `/dev/teepriv0` 节点，说明optee v2的TEE linux kernel驱动已开启。

3.6 TEE库文件

3.6.1 Android

TEE环境相关组件在Android工程目录`vendor/rockchip/common/security`或目录`hardware/rockchip/optee`下（包含V1与V2版本，需根据不同平台采用不同版本文件）：

1. lib：包含32bit与64bit平台编译出来的tee-supplicant、libteec.so以及keymaster/gatekeeper相关库文件。
2. ta：存放编译好的keymaster/gatekeeper等相关TA文件。

3.6.2 Linux

TEE环境相关组件在linux工程目录`external/security/bin`下（包含V1与V2版本，需根据不同平台采用不同版本文件）：

1. lib：包含32bit与64bit平台编译出来的tee-supplicant、libteec.so以及其他CA相关库文件。
2. ta：存放编译好的TA文件。

4. CA/TA开发与测试

4.1 环境配置

1. 若编译报错`No module named Crypto.Signature`，这是开发电脑没有安装python的算法库导致的，执行如下命令即可：

   ```
   pip uninstall Crypto
   pip uninstall pycrypto
   pip install pycrypto
   ```

2. 若出现如下错误：`ModuleNotFoundError: No module named 'Cryptodome'`
   开发主机上请安装python包：`pip3 install [--user] pycryptodomex`

4.2 CA/TA demo

RK提供一套CA/TA demo，目的是：

- 给开发者提供参考，或
- 直接用于测试TEE环境

CA/TA demo的源码在Android工程目录的`external/rk_tee_user`，或Linux工程目录的`external/security/rk_tee_user`，相关的编译说明见下文“CA/TA开发与测试”章节。

CA/TA demo包含“rktest”和“xtest”，其中“xtest”仅在Android环境的`external/rk_tee_user/v2`仓库支持，“xtest”是OPTEE开源的测试代码，包含较完整的测试项。通常的，如果用于测试TEE环境或者参考开发，用“rktest”基本能满足。下面介绍rktest的功能。

rktest demo的CA名为“rktest”，TA名为“1db57234-dacd-462d-9bb1ae79de44e2a5.ta”或“1db57234-dacd-462d-9bb1-ae79de44e2a5.ta”。运行CA程序时，需输入功能参数以选择执行对应的功能。可以输入CA程序名+空格+任意字符，log将提示可用的参数。当前rktest程序实现的测试功能见下表。

备注：测试程序只涉及常用的部分功能，不覆盖OPTEE支持的所有功能。

功能参数	功能	备注
transfer_data	测试CA与TA之间的参数传递	备注
storage	测试安全存储功能	
property	测试获取property	
crypto_sha	测试SHA算法	
crypto_aes	测试AED算法	
crypto_rsa	测试RSA加解密、签名验签	
otp_read	测试读OEM_S_OTP	测试程序默认隐藏otp测试项，如需开启，请在/host/rk_test/main.c中定义。otp相关特性见下文“OTP说明”章节
otp_write	测试写OEM_S_OTP	
otp_size	获取OEM_S_OTP的总大小	
otp_ns_read	测试读OEM_NS_OTP	
otp_ns_write	测试写OEM_NS_OTP	
trng	获取trng数据	

执行测试程序，指令如：

```
# rktest transfer_data
# rktest [command]
```

CA程序执行成功提示PASS，失败提示Fail。

备注：测试Secure Storage功能前，需要确保内核对应节点存在，`/dev/block/by-name/security`对应security分区；rpmb安全存储需要三个节点，`/dev/block/mmcblk%u`，`/dev/block/mmcblk%urpmb`，`/sys/class/mmc_host/mmc%u/mmc%u:0001/cid`，%u值为0 1 2任意一个；若节点不存在请链接到对应节点。

4.3 Android

4.3.1 目录介绍

TEE CA/TA开发环境在安卓工程目录`external/rk_tee_user`下：

1. Android.mk：其中决定了编译的工具和需要编译的ca 文件。
2. host：存放CA的相关源文件。
3. ta：存放TA的源文件。
4. export*：存放编译TA 所依赖的环境。

4.3.2 编译开发说明

若`external/rk_tee_user`目录下只有v1/ v2/两个目录，说明master分支代码已经合并到develop-next分支，master分支将被废弃，合并点为master分支`492f1cbf testapp: support new OP-TEE MSG`，执行如下命令开始编译。

```
#OP-TEE V1平台进入v1目录
cd external/rk_tee_user/v1
#OP-TEE V2平台进入v2目录
cd external/rk_tee_user/v2
rm -rf out/
./build.sh ta
mm
```

若`external/rk_tee_user`目录下没有v1/ v2/两个目录，说明依然使用两个分支，OP-TEE V1请先切换到master分支，OP-TEE V2请先切换到develop-next分支，执行如下命令开始编译。

```
cd external/rk_tee_user/
rm -rf out/
./build.sh ta (git log包含"Android.mk: remove build ta from android"则执行，否则不执行)
mm
```

编译成功后会得到相应的执行程序，执行程序分为CA（Client Application，运行在normal world）和TA（Trust Application，运行在secure world）。

- CA为普通执行文件，编译后生成于Android工程out目录中。
- TA是文件名为uuid，后缀为.ta的文件，编译后生成于rk_tee_user/ta、rk_tee_user/out/ta、rk_tee_user/v1/out/ta、rk_tee_user/v2/out/ta其中一个目录对应的文件夹中。

4.3.3 运行测试TEE环境

1. adb shell进入设备
2. 把相关TEE库文件、CA、TA安装到设备中。Android 7：libteec.so放置到`/system/lib`或`/system/lib64`目录下；tee-supplicant， CA程序放置到/system/bin目录下；创建`/system/lib/optee_armtz`目录，TA程序放置到`/system/lib/optee_armtz`目录下。

   Android 8及更高版本: libteec.so放置到`/vendor/lib`或`/vendor/lib64`目录下；tee-supplicant，CA程序放置到`/vendor/bin`目录下；创建`/vendor/lib/optee_armtz`目录，TA程序放置到`/vendor/lib/optee_armtz`目录下。

   （若开机tee-supplicant自启动，则tee-supplicant和libteec.so不用再push，系统中已有这两个文件；libteec.so和tee-supplicant注意区分OP-TEE V1与OP-TEE V2，注意区分32位和64位；

push后检查下tee-supplicant和CA程序是否有执行权限）

3. 若开机未自动运行tee-supplicant，则需手动root权限后台运行tee-supplicant：

```
# tee-supplicant &
```

若出现`tee_supp_rk_fs_init: unsupported`打印，说明parameter.txt中没有定义security分区，详情请参考2.2章节，若开发者只使用rpmb分区或REE文件系统进行安全存储，可以忽略该错误打印。

4. 运行CA，调用TA，测试TEE相关功能。rk_tee_user自带的rktest程序，可用于直接测试TEE基本功能，执行：

```
# rktest [command]
```

5. 若rktest的各指令运行通过，则TEE环境正常，可进行TEE相关开发。

   若运行报错，请先检查驱动及各组件；

   也可能是rk_tee_user版本与TEE OS版本不匹配导致，以下为常用匹配关系：

   - OP-TEE V1:

     rkbin/bin目录下TEE binary文件名中 version >= v2.00，

     对应`492f1cbf testapp: support new OP-TEE MSG`

     rkbin/bin目录下TEE binary文件名中 version < v2.00，

     对应`e8d7215d Android.mk: support build in android R`

     或者`466515ec add tools for user to resign TA`

   - OP-TEE V2:

     TEE启动阶段串口打印"OP-TEE version: 3.13.0"，对应`a566557 - v2: update to keep up with v3.13.0 of optee_test`

     TEE启动阶段串口打印"OP-TEE version: 3.6.0"，对应`1aa969e2 Android.mk: support build in android R`

     TEE启动阶段串口打印"OP-TEE version: 3.3.0"，对应`aa0a0c00 Android.mk: remove build ta from android`

     TEE启动阶段串口打印"OP-TEE version: 2.5.0"，对应`1ec9913a add tools for user to resign TA`

4.3.4 开发CA/TA

可参考CA，TA中的Makefile与头文件的UUID需要修改成新生成的UUID，可用uuidgen命令生成。

在每个TA的include目录下的头文件user_ta_header_defines.h中定义了堆栈的大小，堆的大小为32KB（TA_DATA_SIZE），栈的大小为2KB（TA_STACK_SIZE）。一般情况下最好不要去修改，若实在无法满足需求，可适当改大一些，堆的大小不要超过1MB，栈的大小不要超过64KB。

```
#define TA_STACK_SIZE                (2 * 1024)
#define TA_DATA_SIZE                 (32 * 1024)
```

4.4 Linux

4.4.1 目录介绍

TEE CA/TA开发环境在linux工程目录`external/security/rk_tee_user`下：

1. build.sh：编译执行脚本，编译说明请参考脚本中的注释。
2. Makefile：其中决定了编译的工具和需要编译的ca 文件。
3. host：存放CA的相关源文件以及对应Makefile。
4. ta：存放TA的源文件。
5. export*：存放编译TA 所依赖的环境。

4.4.2 编译开发说明

若`external/security/rk_tee_user`目录下只有v1/ v2/两个目录，说明master分支代码已经合并到develop-next分支，master分支将被废弃，合并点为master分支`492f1cbf testapp: support new OP-TEE MSG`，执行如下命令开始编译。

```
#OP-TEE V1平台进入v1目录
cd external/security/rk_tee_user/v1
#OP-TEE V2平台进入v2目录
cd external/security/rk_tee_user/v2
rm -rf out/
./build.sh 3232 （32位平台执行，CA 32bits，TA 32bits）
./build.sh 6432 （64位平台执行，CA 64bits，TA 32bits）
```

若`external/security/rk_tee_user`目录下没有v1/ v2/两个目录，说明依然使用两个分支，OP-TEE V1请先切换到master分支，OP-TEE V2请先切换到develop-next分支，执行如下命令开始编译。

```
cd external/security/rk_tee_user/
rm -rf out/
./build.sh 3232 （32位平台执行，CA 32bits，TA 32bits）
./build.sh 6432 （64位平台执行，CA 64bits，TA 32bits）
```

编译成功后会得到相应的执行程序，执行程序分为CA（Client Application，运行在normal world）和TA（Trust Application，运行在secure world）。

- CA为普通执行文件，编译后生成于rk_tee_user/out、rk_tee_user/v1/out、rk_tee_user/v2/out其中一个目录下对应的文件夹中。
- TA是文件名为uuid，后缀为.ta的文件，编译后生成于rk_tee_user/out/ta、rk_tee_user/v1/out/ta、rk_tee_user/v2/out/ta其中一个目录对应的文件夹中。

4.4.3 运行测试TEE环境

1. adb shell进入设备。
2. 把相关TEE库文件、CA、TA安装到设备中。libteec.so*等库文件放置到/lib或/lib64目录下；tee-supplicant，CA程序放置到/usr/bin目录下；创建/lib/optee_armtz目录，TA程序放置到/lib/optee_armtz目录下。

   （若开机tee-supplicant自启动，则tee-supplicant和libteec.so不用再push，系统中已有这两个文件；

libteec.so和tee-supplicant注意区分OP-TEE V1与OP-TEE V2，注意区分32位和64位；

push后检查下tee-supplicant和CA程序是否有执行权限）

3. 其他步骤，与Android平台相同，见上述“Android”章节。

4.4.4 开发CA/TA

可参考CA，TA中的Makefile与头文件的UUID需要修改成新生成的UUID，可用uuidgen命令生成。

在每个TA的include目录下的头文件`user_ta_header_defines.h`中定义了堆栈的大小，堆的大小为32KB（TA_DATA_SIZE），栈的大小为2KB（TA_STACK_SIZE）。一般情况下最好不要去修改，若实在无法满足需求，可适当改大一些，堆的大小不要超过1MB，栈的大小不要超过64KB。

```
#define TA_STACK_SIZE                (2 * 1024)
#define TA_DATA_SIZE                 (32 * 1024)
```

4.5 rk_tee_service

4.5.1 功能介绍

rk_tee_service 是基于TEE开发的安全功能服务，为开发者提供了常用的安全功能，简单清晰的对外接口极大便利了开发者使用。rk_tee_service 本质上是CA TA应用，因此请按照 “运行测试TEE环境” 章节测试TEE环境正常后再使用 rk_tee_service。

开发者可以直接调用 rk_tee_service 对敏感数据进行加解密，数据在TA端进行加解密，加解密密钥由TEE使用设备硬件唯一密钥HUK派生，因此每台设备的加解密密钥都不同，拷贝设备A的敏感数据到设备B上是无法正常解密的，保证敏感数据不被盗用。受限于共享内存的大小，建议单次加解密数据大小不要超过1M，大数据建议分多次进行加解密。

目前OP-TEE V2平台支持该功能，OP-TEE V1平台暂不支持。

4.5.2 组件

目前支持 Linux 平台和 Android 平台（Android 12及更高版本）。

组件	**Android** 目录	**Linux** 目录
librk_tee_service.so	hardware/rockchip/optee/v2/arm hardware/rockchip/optee/v2/arm64	external/security/bin/optee_v2/lib/arm external/security/bin/optee_v2/lib/arm64
rk_tee_service.h	hardware/rockchip/optee/v2/include	external/security/bin/optee_v2/include
4367fd45-4469-42a6-925d-3857b952704a.ta	hardware/rockchip/optee/v2/ta	external/security/bin/optee_v2/ta

Userspace 应用可以直接调用 librk_tee_service.so 动态库，函数的参数说明请参考 rk_tee_service.h，Linux平台请把4367fd45-4469-42a6-925d-3857b952704a.ta放置到 /lib/optee_armtz 目录下，Android平台请把4367fd45-4469-42a6-925d-3857b952704a.ta放置到 /vendor/lib/optee_armtz 目录下。

4.5.3 参考Demo

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "rk_tee_service.h"

int main(int argc, char *argv[])
{
    unsigned char plain[256];
    unsigned int plain_len;
    unsigned char cipher[256];
    unsigned int cipher_len;
    int res;

    memset((void *)plain, 0xab, sizeof(plain));
    cipher_len = 256;
    res = rk_encrypt_data(plain, sizeof(plain), cipher, &cipher_len);
    printf("res=0x%x cipher_len=%d\n", res, cipher_len);

    memset((void *)plain, 0, sizeof(plain));
    plain_len = 256;
    res = rk_decrypt_data(cipher, cipher_len, plain, &plain_len);
    printf("res=0x%x plain_len=%d\n", res, plain_len);

    return 0;
}
```

5. TA签名

5.1 原理

在编译TA时，编译脚本将自动使用rk_tee_user工程`export-user_ta/keys`目录或者`export-ta_arm32/keys`目录下的密钥对TA应用进行签名，该密钥为pem格式的2048长度RSA密钥，最终生成 .ta 格式的TA文件。

TEE binary内部保存着一份RSA公钥，在加载运行TA时，TEE OS将使用该公钥验证TA的合法性，验证通过才能正常运行TA应用，这将保证运行的TA都是合法。

5.2 替换公钥

为防止开发者A的TA应用运行在开发者B的板子上，建议开发者替换公钥。

开发者可以使用工具替换TEE binary中的公钥，所需工具在rk_tee_user工程tools/目录中。

- Linux下替换：

```
./change_puk  --teebin  <TEE binary>
```

该命令将自动生成一个2048长度的RSA密钥oemkey.pem并保存在当前目录下，并自动使用该密钥中的公钥替换TEE binary中的原始公钥。

```
./change_puk  --teebin  <TEE binary> --key oemkey.pem
```

使用开发者指定的密钥中的公钥来替换TEE binary中的原始公钥，密钥长度须2048长度。

- Windows下替换：

  打开Windows_change_puk.exe点击“生成oemkey.pem”按钮生成并保存密钥。

  选择刚刚生成的密钥和TEE binary，点击修改公钥。

  （由于Windows_change_puk.exe会调用BouncyCastle.Crypto.dll第三方库，请确保BouncyCastle.Crypto.dll与Windows_change_puk.exe在同一目录下）

替换公钥后，开发者需使用新的TEE binary（新公钥）替换rkbin/目录下的TEE binary（原始公钥），重新编译U-Boot，烧写新生成的trust.img固件，部分平台trust打包进uboot.img所以没有trust.img，开发者烧写uboot.img即可。

开发者需把之前工具生成的密钥或者指定的密钥重命名为oem_privkey.pem并替换rk_tee_user工程`export-user_ta/keys`目录或者`export-ta_arm32/keys`目录下密钥，重新编译CA TA应用，这样得到的TA应用就可以被TEE binary（新公钥）正确加载并运行，其他没有使用正确私钥签名的TA应用都视为非法TA不能运行。

6. 内置TA到安全存储

正常情况下TA开发完成后，TA文件会以明文形式存放于非安全文件系统中，部分对安全性要求较高的OEM厂商不希望TA以明文的形式暴露出去，为了支持OEM厂商这一需求，OP-TEE V2支持内置TA到安全存储（OP-TEE V1暂不支持）。

6.1 原理

CA端读取非安全文件系统中的TA文件，再把TA数据发送给OP-TEE OS，OP-TEE OS接收到TA数据会先校验TA的合法性，若TA是合法的则随机生成TA加密密钥，使用TA加密密钥对TA数据进行加密，再将密文TA数据和TA加密密钥进行安全存储，安全存储所使用的密钥由硬件唯一密钥派生，每台设备都不同。最后开发者需自行删除掉非安全文件系统中的TA文件，防止明文TA暴露。

上诉步骤完成后，CA可以正常调用TA应用，当CA调用安全存储中的TA时，OP-TEE OS会根据传入的uuid在安全存储中查找对应TA，若查找到对应TA则解密并加载运行TA，若没有查找到对应TA则会到非安全文件系统中查找对应TA。

6.2 参考实现

以下为CA端代码，开发者可以先读取TA文件，然后通过 install_ta 函数把TA数据发送到OP-TEE OS。

```
static void install_ta(void *buf, size_t blen)
{
```

```
    TEEC_Result res = TEEC_ERROR_GENERIC;
    uint32_t err_origin = 0;
    TEEC_UUID uuid = PTA_SECSTOR_TA_MGMT_UUID;
    TEEC_Operation op;
    TEEC_Context ctx = { };
    TEEC_Session sess = { };
    int i = 0;

    res = TEEC_InitializeContext(NULL, &ctx);
    if (res != TEEC_SUCCESS) {
        printf("TEEC_InitializeContext failed with code 0x%x\n", res);
        goto exit;
    }

    res = TEEC_OpenSession(&ctx, &sess, &uuid,
                   TEEC_LOGIN_PUBLIC, NULL, NULL, &err_origin);
    if (res != TEEC_SUCCESS) {
        printf("TEEC_Opensession failed with code 0x%x origin 0x%x\n",
               res, err_origin);
        goto exit;
    }

    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 = buf;
    op.params[0].tmpref.size = blen;


    res = TEEC_InvokeCommand(&sess, PTA_SECSTOR_TA_MGMT_BOOTSTRAP, &op,
                 &err_origin);
    if (res != TEEC_SUCCESS) {
        printf("TEEC_InvokeCommand failed with code 0x%x origin 0x%x\n",
               res, err_origin);
        goto exit;
    }
    printf("Installing TAs done\n");

exit:
    TEEC_CloseSession(&sess);
    TEEC_FinalizeContext(&ctx);

    return;
}
```

7. 加密TA

在上一章节 ”内置TA到安全存储“ 中介绍了避免暴露明文TA的一种方法，但是内置TA到安全存储会占用安全存储的空间，且加密TA的密钥是随机生成的，不是开发者自己的加密密钥，本章节将介绍另外一种加密TA的方法（OP-TEE V2支持，OP-TEE V1暂不支持）。

7.1 加密TA方法

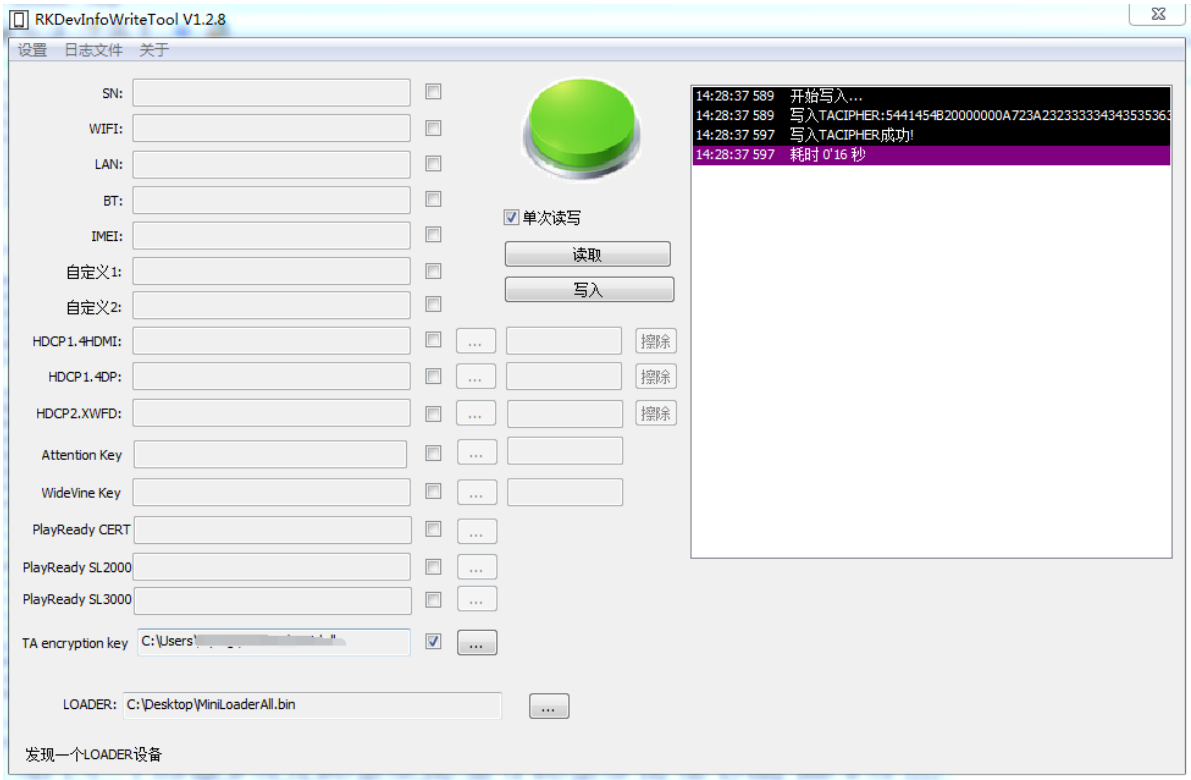
开发者需在 `export-ta_arm32/mk/link.mk` 中开启 `CFG_ENCRYPT_TA` 宏，同时修改 `TA_ENC_KEY` 为开发者自己的加密密钥，开启该宏后开发者在编译TA应用时脚本会自动对TA进行签名和加密。

7.2 烧写TA encryption key

开发者使用SDK工程RKTools目录下 `RKDevInfoWriteTool` 工具（版本号大于等于1.2.8）烧写TA encryption key。

使用前开发者需新建一个密钥文件，以16进制编辑格式打开该文件，在文件中编辑开发者的32字节加密密钥，打开RKDevInfoWriteTool工具，勾选 “TA encryption key”，点击按钮选择开发者新建的密钥文件，确认设备进入 LOADER模式后，点击 “写入” 按钮进行烧写，密钥最终会写入到设备OTP中（OTP区域一旦写入不可更改，所以一台设备密钥只能烧写一次）。

为防止TA encryption key泄漏，工具不支持读TA encryption key。



7.3 解密并运行TA

加密TA与明文TA在使用上完全一致，OP-TEE OS在加载TA过程会自动识别到TA是加密的，OP-TEE OS会自动读取OTP中TA encryption key并解密运行TA，该过程由OP-TEE OS自动完成。

8. REE FS TA防回滚

由“内置TA到安全存储”章节可知，OP-TEE V2版本既支持把TA明文存放在非安全文件系统（REE FS）中，也支持把TA存放在安全存储（Secure Storage）中。对于TA明文存放在REE FS的场景，OP-TEE V2版本支持TA防回滚，防止在REE非安全环境中TA版本降级。

8.1 使用TA防回滚

REE FS的TA防回滚功能始终开启，开发者可通过在Makefile定义TA的版本号来使用防回滚功能。

如果TA的Makefile始终不定义`CFG_TA_VERSION`，系统识别TA版本号为0，允许相同版本号的TA运行。

如果TA的Makefile定义当前`CFG_TA_VERSION`大于0，示例如下，则后续禁止该TA版本降级。

```
# unsigned integer format
CFG_TA_VERSION=1
```

9. TA调试方法

9.1 optee v1平台

TA出现异常时会打印如下信息。

```
user TA data-abort at address 0x2a

 esr 0x92000021  ttbr0 0x400000852fc00   ttbr1 0x00000000   cidr 0x0

 cpu #4          cpsr 0x20000130

#32位平台打印r0-r12, sp, lr, pc(402000a0)
#64位平台打印x0-x30, sp_el0, elr(00000000402000a0)

Status of TA 8cccf200-2450-11e4-abe20002a5d5c52c (0x85109b0) (active)
- load addr : 0x40200000    ctx-idr: 4
- code area : 0x9200000 2097152
- stack: 0x9400000 stack:2048
TEEC_InvokeCommand failed with code 0xffff3024 origin 0x3
```

异常打印中pc或elr就是异常发生的地址，load addr是TA加载到内存中的运行虚拟地址，所以elr - load addr = 0x402000a0 - 0x40200000 = 0xa0就是异常代码在TA中的偏移地址。

在TA文件同级目录下有个uuid.dmp文件，该文件是TA的反汇编文件，记录了TA中各函数的偏移地址，打开uuid.dmp文件并查找 a0，其中 `a0:   681b         ldr r3, [r3, #0]` 就是异常偏移地址，`testapp_ta.c:97` 表示异常代码在testapp_ta.c第97行。

```
/home/xxx/android/vendor/optee_test/ta/testapp/testapp_ta.c:97
      98:   6823        ldr r3, [r4, #0]
      9a:   2202        movs    r2, #2
      9c:   2161        movs    r1, #97 ; 0x61
      9e:   4820        ldr r0, [pc, #128]  ; (120
<TA_InvokeCommandEntryPoint+0xa0>)
      a0:   681b        ldr r3, [r3, #0]
      a2:   4478        add r0, pc
      a4:   3033        adds    r0, #51 ; 0x33
      a6:   9301        str r3, [sp, #4]
      a8:   4b1e        ldr r3, [pc, #120]  ; (124
<TA_InvokeCommandEntryPoint+0xa4>)
```

```
      aa:   447b        add r3, pc
      ac:   9300        str r3, [sp, #0]
      ae:   2301        movs    r3, #1
      b0:   f002 fbc0   bl  2834 <trace_printf>
```

9.2 optee v2平台

TA出现异常时会打印如下信息。

```
E/TC:? 0 User mode data-abort at address 0x2a (translation fault)
E/TC:? 0  esr 0x92000005  ttbr0 0x20000084a7020   ttbr1 0x00000000   cidr 0x0
E/TC:? 0  cpu #1          cpsr 0x20000130

#32位平台打印r0-r12, sp, lr, pc(c00870a4)
#64位平台打印x0-x30, sp_el0, elr(00000000c00870a4)

E/LD:  region  0: va 0xc0004000 pa 0x08600000 size 0x002000 flags rw-s (ldelf)
E/LD:  region  1: va 0xc0006000 pa 0x08602000 size 0x008000 flags r-xs (ldelf)
E/LD:  region  2: va 0xc000e000 pa 0x0860a000 size 0x001000 flags rw-s (ldelf)
E/LD:  region  3: va 0xc000f000 pa 0x0860b000 size 0x004000 flags rw-s (ldelf)
E/LD:  region  4: va 0xc0013000 pa 0x0860f000 size 0x001000 flags r--s
E/LD:  region  5: va 0xc0014000 pa 0x08625000 size 0x001000 flags rw-s (stack)
E/LD:  region  6: va 0xc0015000 pa 0x09201000 size 0x002000 flags rw-- (param)
E/LD:  region  7: va 0xc0087000 pa 0x00001000 size 0x009000 flags r-xs [0]
E/LD:  region  8: va 0xc0090000 pa 0x0000a000 size 0x00c000 flags rw-s [0]
E/LD:   [0] 8cccf200-2450-11e4-abe2-0002a5d5c52c @ 0xc0087000
E/LD:  Call stack:
E/LD:   0xc00870a4
E/LD:   0xc0088b21
E/LD:   0xc008d507
E/LD:   0xc008716c
```

异常打印中pc或elr就是异常发生的地址，region 0 到 region 8 是TA代码段在内存中的地址，这里异常地址 0xc00870a4 属于 region 7 代码段，所以elr - region 7 : va = 0xc00870a4 - 0xc0087000 = 0xa4 就是异常代码在TA中的偏移地址。

在TA文件同级目录下有个uuid.dmp文件，该文件是TA的反汇编文件，记录了TA中各函数的偏移地址，打开uuid.dmp文件并查找 a4，其中 `a4:  681b  ldr r3, [r3, #0]` 就是异常偏移地址，`testapp_ta.c:101` 表示异常代码在testapp_ta.c第101行。

```
/home/xxx/rk_px30_linux/external/optee_test/ta/testapp/testapp_ta.c:101
      9c:   6823        ldr r3, [r4, #0]
      9e:   2202        movs    r2, #2
      a0:   4d28        ldr r5, [pc, #160]  ; (144
<TA_InvokeCommandEntryPoint+0xc4>)
      a2:   4e29        ldr r6, [pc, #164]  ; (148
<TA_InvokeCommandEntryPoint+0xc8>)
      a4:   681b        ldr r3, [r3, #0]
      a6:   447d        add r5, pc
      a8:   447e        add r6, pc
      aa:   3533        adds    r5, #51 ; 0x33
      ac:   4628        mov r0, r5
```

```
      ae:   9600        str r6, [sp, #0]
      b0:   9301        str r3, [sp, #4]
      b2:   2301        movs    r3, #1
      b4:   f000 f912   bl  2dc <trace_printf>
```

9.3 查看调用栈

若开发者调试代码时不仅仅要查看程序异常地址，还想查看整个函数调用栈，optee v2平台提供了脚本export-ta_arm32\scripts\symbolize.py脚本实现该功能。optee v1平台暂不支持。

第一步，设置脚本需要的编译器路径，开发者请根据自己本地编译器路径自行调整。

```
#32位TA执行
export PATH=/home1/hisping/rk_px30_linux/prebuilts/gcc/linux-x86/arm/gcc-linaro-6.3.1-2017.05-x86_64_arm-linux-gnueabihf/bin:$PATH
export CROSS_COMPILE=arm-linux-gnueabihf-

#64位TA执行
export PATH=/home1/hisping/rk_px30_linux/prebuilts/gcc/linux-x86/aarch64/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin:$PATH
export CROSS_COMPILE=aarch64-linux-gnu-
```

第二步，执行脚本，其中-d参数指向开发者编译出的TA目录。

```
./export-ta_arm32/scripts/symbolize.py -d out/ta/testapp/
```

第三步，脚本会等待开发者输入异常打印信息，复制并粘贴异常打印信息，可以得到如下结果，其中Call stack就是函数调用栈。

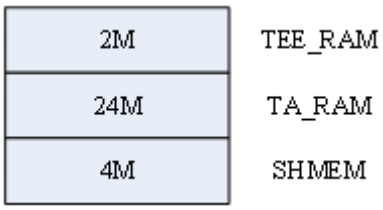
```
I/TA: Hello Test App!
E/TC:? 0
E/TC:? 0 User mode data-abort at address 0x2a (translation fault)
E/TC:? 0  esr 0x92000005  ttbr0 0x20000084a7020   ttbr1 0x00000000   cidr 0x0
E/TC:? 0  cpu #1          cpsr 0x20000130
E/TC:? 0  x0  00000000000069ee x1  0000000000000062
E/TC:? 0  x2  0000000000000002 x3  000000000000002a
E/TC:? 0  x4  00000000c0014f30 x5  00000000c0014f40
E/TC:? 0  x6  00000000c0074080 x7  00000000c0074308
E/TC:? 0  x8  00000000c00742e8 x9  00000000c0014f30
E/TC:? 0  x10 0000000000000065 x11 00000000c007f2d8
E/TC:? 0  x12 0000000000000773 x13 00000000c0014f00
E/TC:? 0  x14 00000000c006cb0d x15 0000000000000000
E/TC:? 0  x16 0000000000000000 x17 0000000000000000
E/TC:? 0  x18 0000000000000000 x19 0000000000000000
E/TC:? 0  x20 0000000000000000 x21 0000000000000000
E/TC:? 0  x22 0000000000000000 x23 0000000000000000
E/TC:? 0  x24 0000000000000000 x25 0000000000000000
E/TC:? 0  x26 0000000000000000 x27 0000000000000000
E/TC:? 0  x28 0000000000000000 x29 0000000000000000
E/TC:? 0  x30 0000000000000000 elr 00000000c006b0a4
E/TC:? 0  sp_el0 00000000c0014f80
E/LD:  Status of TA 8cccf200-2450-11e4-abe2-0002a5d5c52c
```

```
E/LD:   arch: arm
E/LD:  region  0: va 0xc0004000 pa 0x08600000 size 0x002000 flags rw-s (ldelf)
E/LD:  region  1: va 0xc0006000 pa 0x08602000 size 0x008000 flags r-xs (ldelf)
E/LD:  region  2: va 0xc000e000 pa 0x0860a000 size 0x001000 flags rw-s (ldelf)
E/LD:  region  3: va 0xc000f000 pa 0x0860b000 size 0x004000 flags rw-s (ldelf)
E/LD:  region  4: va 0xc0013000 pa 0x0860f000 size 0x001000 flags r--s
E/LD:  region  5: va 0xc0014000 pa 0x08625000 size 0x001000 flags rw-s (stack)
E/LD:  region  6: va 0xc0015000 pa 0x09201000 size 0x002000 flags rw-- (param)
E/LD:  region  7: va 0xc006b000 pa 0x00001000 size 0x009000 flags r-xs [0]
.ta_head .text .rodata .ARM.extab .ARM.exidx .dynsym .dynstr .hash
E/LD:  region  8: va 0xc0074000 pa 0x0000a000 size 0x00c000 flags rw-s [0]
.dynamic .got .rel.got .data .bss .rel.dyn
E/LD:   [0] 8cccf200-2450-11e4-abe2-0002a5d5c52c @ 0xc006b000
(out/ta/testapp/8cccf200-2450-11e4-abe2-0002a5d5c52c.elf)
E/LD:  Call stack:
E/LD:   0xc006b0a4 TA_InvokeCommandEntryPoint at ta/testapp/testapp_ta.c:98
E/LD:   0xc006cb0d entry_invoke_command at
/home/zhangzj/secure/optee_3.6.0/optee_os/lib/libutee/arch/arm/user_ta_entry.c:35
7
E/LD:   0xc00714f3 __ta_entry_c at export-ta_arm32/src/user_ta_header.c:48
E/LD:   0xc006b158 __ta_entry at export-ta_arm32/src/ta_entry_a32.S:20
```

10. 内存相关说明

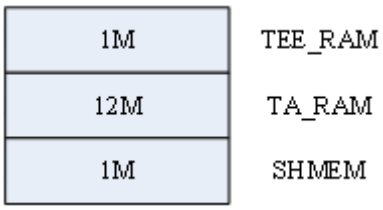
10.1 OP-TEE V1

ARMv8架构芯片中TEE内存分配情况如下：



说明：占用内存总共30M，Secure OS运行在TEE_RAM，TA运行在 TA_RAM ，共享内存占用4M空间。

ARMv7架构芯片中TEE内存分配情况如下：



说明：占用内存总共14M, Secure OS运行在TEE_RAM，TA运行在TA_RAM，共享内存占用1M空间。

10.2 OP-TEE V2

各平台大小不固定，并且运行大小可能会调整，这里就不统一说明了。

11. 安全存储

11.1 分区

1. 安全存储是OPTEE OS重要的功能之一，一般用于存储用户重要数据，数据经过OPTEE OS加密存储于security分区或者rpmb分区或者REE文件系统，具体存储于哪个分区由CA传递参数告知TA，由TA负责存储。
2. Uboot端安全存储，uboot端optee client代码存放于`lib/optee_client`目录下，一般情况下使用emmc时安全存储存于rpmb，使用nand时安全存储存于security分区，当emmc的rpmb不可用时可以开启`CONFIG_OPTEE_ALWAYS_USE_SECURITY_PARTITION`强制使用security分区，相关宏定义如下：

   `CONFIG_OPTEE_CLIENT`，U-Boot调用OP-TEE总开关。

   `CONFIG_OPTEE_V1`，采用OP-TEE V1的平台使用。

   `CONFIG_OPTEE_V2`，采用OP-TEE V2的平台使用。

   `CONFIG_OPTEE_ALWAYS_USE_SECURITY_PARTITION`， 当emmc的rpmb不能用，才开这个宏，默认不开。

   Uboot端不支持开发者开发自己的TA应用，但是可以开发自己的CA调用OPTEE OS内部静态TA存储开发者自己的数据，开发者只需要设置对应的文件名和数据即可进行安全存储，细节请参考lib/optee_client/OpteeClientInterface.c中的函数。
3. Android端安全存储，定义`PRODUCT_PROPERTY_OVERRIDES += ro.tee.storage=auto`代表根据硬件来选择安全存储区域，emmc使用rpmb，nand使用security；

   定义`PRODUCT_PROPERTY_OVERRIDES += ro.tee.storage=rpmb`代表使用rpmb

   定义`PRODUCT_PROPERTY_OVERRIDES += ro.tee.storage=rkss`代表使用security分区
4. 在进行安全存储写入数据时请确保设备不发生断电，虽然我们做了断电保护测试，但不保证文件系统完整性，所以建议开发者减少对重要数据的写入次数，确保操作安全数据时环境的稳定性。

11.2 性能测试

OPTEE V1平台，RK3399 Linux平台，CPU定频1200000，DDR定频200000000，进行测试：

存储区域	数据大小	创建文件	写数据	读数据	删除文件
Linux文件系统	30K	16ms	67ms	61ms	19ms
Linux文件系统	4K	17ms	23ms	13ms	7ms
Linux文件系统	1K	18ms	16ms	7ms	6ms
Linux文件系统	32	23ms	16ms	7ms	7ms
security分区	30K	97ms	181ms	54ms	277ms
security分区	4K	101ms	74ms	14ms	101ms
security分区	1K	104ms	56ms	7ms	64ms
security分区	32	103ms	55ms	7ms	73ms
rpmb分区	30K	20ms	233ms	10ms	7ms
rpmb分区	4K	20ms	36ms	3ms	6ms
rpmb分区	1K	22ms	14ms	2ms	6ms
rpmb分区	32	27ms	8ms	2ms	6ms

OPTEE V2平台，RK356x Linux平台，CPU定频1416000，DDR定频324000000，进行测试：

存储区域	数据大小	创建文件	写数据	读数据	删除文件
Linux文件系统	30K	17ms	28ms	3ms	8ms
Linux文件系统	4K	17ms	11ms	1ms	8ms
Linux文件系统	1K	18ms	9ms	1ms	8ms
Linux文件系统	32	19ms	8ms	1ms	7ms
security分区	30K	12ms	12ms	4ms	12ms
security分区	4K	12ms	3ms	1ms	11ms
security分区	1K	13ms	2ms	1ms	11ms
security分区	32	15ms	3ms	1ms	14ms
rpmb分区	30K	23ms	287ms	16ms	5ms
rpmb分区	4K	24ms	50ms	7ms	6ms
rpmb分区	1K	23ms	22ms	5ms	6ms
rpmb分区	32	30ms	12ms	5ms	5ms

12. 强弱安全等级可选的方案

12.1 方案的适用范围

方案适用于：RK3588系列。

12.2 注意事项

在使用本方案之前，开发者需注意以下事项。

- 请在首次下载固件之前，确认uboot的配置项CONFIG_OPTEE_SECURITY_LEVEL已按需求配置。芯片只支持一次配置，后续无法修改。
- 如果采用“强安全方案2”，请先用工具“RKDevInfoWriteTool(V1.3.0及以上版本)”，下载OEM HUK。

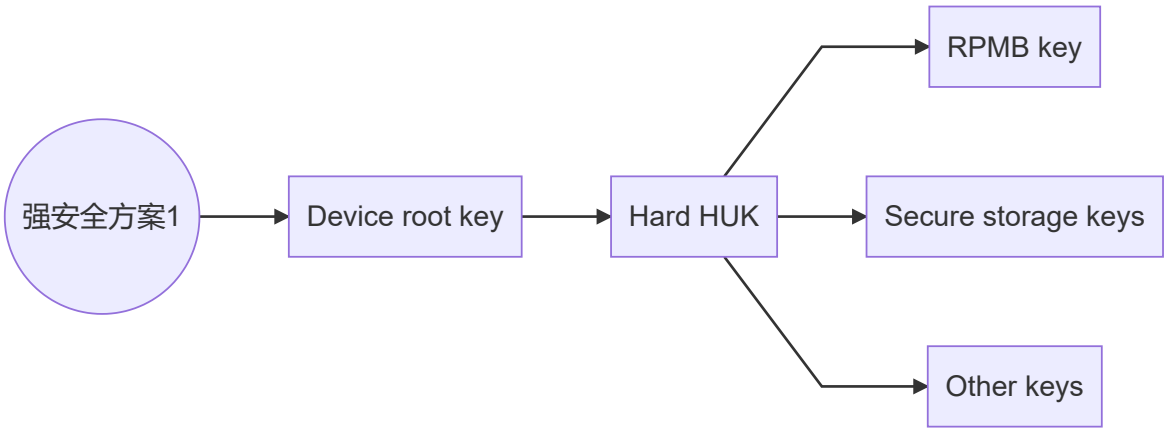
12.3 方案说明

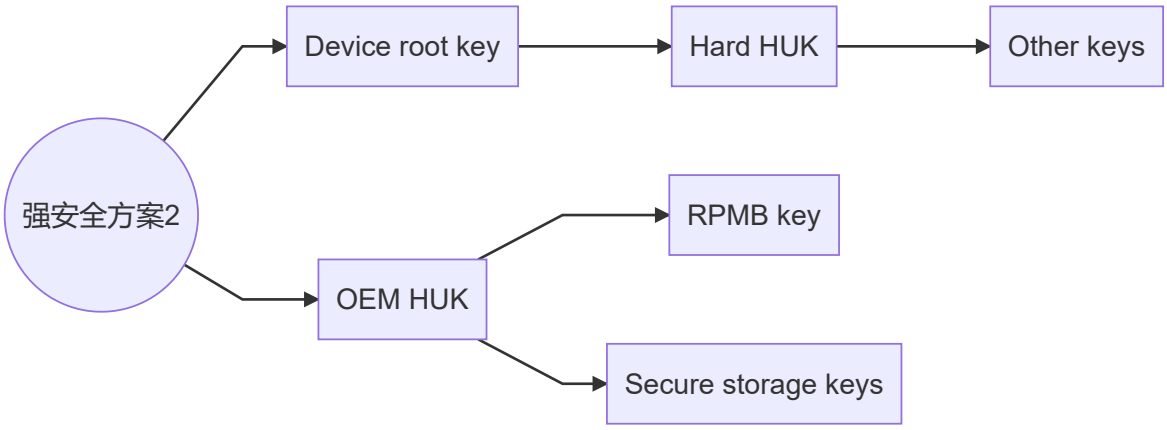
方案支持由开发者配置OPTEE的安全等级，不同的安全等级对eMMC/安全存储的保护强度不同。

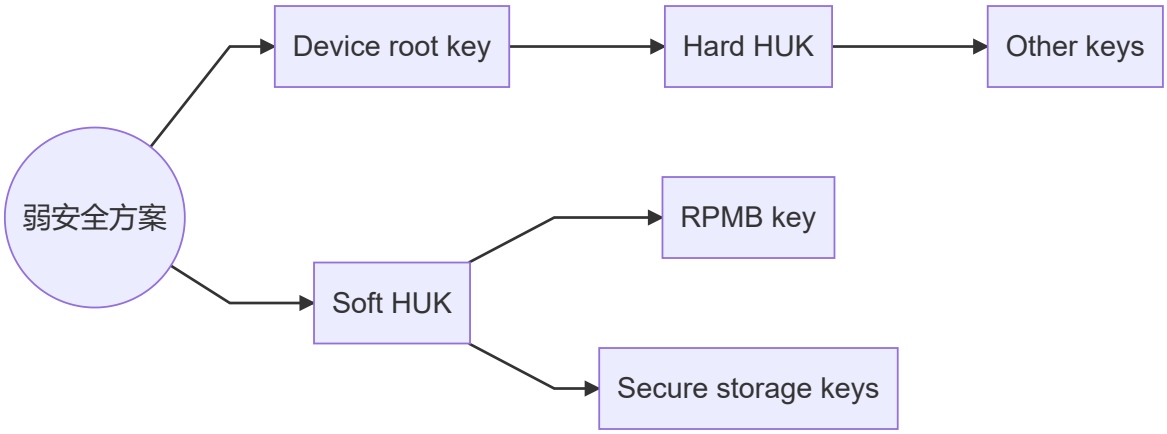
安全等级方案	说明	**CONFIG_OPTEE_SECURITY_LEVEL (uboot的配置项)**
强安全方案1	主控和eMMC/安全存储的数据强绑定，更换主控后，需同步更换eMMC芯片和擦除其他安全存储数据	2
强安全方案2	主控和eMMC/安全存储的数据弱绑定，由开发者定义的OEM HUK来派生eMMC/安全存储的保护密钥，更换主控后，如果下载相同的OEM HUK，可继续使用原有的eMMC芯片和其他安全存储数据	1
弱安全方案	主控和eMMC/安全存储的数据不绑定，更换主控后，可继续使用原有的eMMC芯片和其他安全存储数据	0或不配置

以上不同安全等级的差异，源于eMMC和安全存储相关密钥的差异。下面介绍本方案相关的密钥。

- HUK：用于派生RPMB key、Secure storage keys以及其他密钥。不同安全的等级，HUK不同，详见下图。
  Hard HUK是基于Device root key派生的，一机一密。
  OEM HUK是由开发者定义的，存储在安全OTP中。
  Soft HUK是Rockchip定义的，所有芯片为同一个密钥，固化在固件的密钥。
- RPMB key：eMMC芯片的存储保护密钥。
- Secure storage keys：安全存储保护密钥。







13. OTP说明

OTP指One Time Programmable Memory，OTP区域支持多次读取，但只能写入一次。
RK支持以下OTP类型。

OTP类型	说明	支持的平台
OEM_HR_OTP	支持硬件可读的安全OTP。当不配置时默认只允许安全世界读写，安全世界内存可访问读写的数据；当配置为仅硬件可读时，限制所有软件无法访问读写的数据，数据不出现在安全和非安全世界内存中。这通常可用于保护密钥等敏感数据，达到密钥与CPU软件隔离的目的。以上配置的动作也是由OTP机制控制只能配一次。下面`OTP API`章节提供了相关操作接口。	rk3588系列
OEM_S_OTP	安全OTP，只允许安全世界读写。下面`OTP API`章节提供了相关操作接口。	rk3308，rk3326，rk3358，rk3568，rk3588，rv1126
OEM_NS_OTP	普通安全等级的OTP，目前分为两种：1是通过CA调用OPTEE内部TA进行访问，当前总大小固定为64Byte；2是在REE环境通过其他方式直接访问。若无特殊说明，本文OEM_NS_OTP指的是第1种。	rk3308，rk3326，rk3358

14. TA API说明

14.1 概述

RK提供以下TA API，有两个目的：

- 供开发者参考常用的GlobalPlatform TEE Internal Core API的用法
- 供开发者直接使用API

14.2 API的返回值

API的返回值有：

- TEE_SUCCESS: 成功
- TEE_ERROR_BAD_PARAMETERS: 参数错误
- 其他错误: 见`tee_api_defines.h`

14.3 API说明

14.3.1 Crypto API

14.3.1.1 rk_crypto_malloc_ctx

```
crypto_ctx_t *rk_crypto_malloc_ctx(void);
```

功能

申请crypto操作句柄资源。

参数

- 无

14.3.1.2 rk_crypto_free_ctx

```
void rk_crypto_free_ctx(crypto_ctx_t **ctx);
```

功能

释放crypto操作句柄，完整的算法执行完毕，需执行本函数以释放资源。

参数

- ctx: 操作句柄

14.3.1.3 rk_hash_crypto

```
TEE_Result rk_hash_crypto(uint8_t *in, uint8_t *out, uint32_t in_len,
                          uint32_t out_len, uint32_t algo);
```

功能

消息摘要算法。如需分段，可使用`rk_hash_begin/update/finish`接口。

参数

- in: 输入数据
- in_len: 输入数据长度
- out: 输出数据
- out_len: 输出数据长度
- algo: 算法类型，支持`TEE_ALG_MD5, TEE_ALG_SHA1, TEE_ALG_SHA224, TEE_ALG_SHA256, TEE_ALG_SHA384, TEE_ALG_SHA512`

14.3.1.4 rk_hash_begin

```
TEE_Result rk_hash_begin(crypto_ctx_t *ctx, uint32_t algo);
```

功能

消息摘要分段算法，初始化操作。

参数

- ctx: 操作句柄

- algo: 算法类型，支持`TEE_ALG_MD5, TEE_ALG_SHA1, TEE_ALG_SHA224, TEE_ALG_SHA256, TEE_ALG_SHA384, TEE_ALG_SHA512`

14.3.1.5 rk_hash_update

```
TEE_Result rk_hash_update(crypto_ctx_t *ctx, uint8_t *in, uint32_t in_len);
```

功能

消息摘要分段算法，对输入的数据，在内部计算摘要。

参数

- ctx: 操作句柄
- in: 输入数据
- in_len: 输入数据长度

14.3.1.6 rk_hash_finish

```
TEE_Result rk_hash_finish(crypto_ctx_t *ctx, uint8_t *in, uint8_t *out,
                          uint32_t in_len, uint32_t *out_len);
```

功能

消息摘要分段算法，计算最后一段数据并输出所有数据的摘要。

参数

- ctx: 操作句柄
- in: 输入数据
- in_len: 输入数据长度
- out: 输出数据
- out_len: 输出数据长度

14.3.1.7 rk_cipher_crypto

```
TEE_Result rk_cipher_crypto(uint8_t *in, uint8_t *out, uint32_t len,
                            uint8_t *key, uint32_t key_len, uint8_t *iv,
                            uint32_t algo, TEE_OperationMode mode);
```

功能

对称加解密算法接口。如需分段，可使用`rk_cipher_begin/update/finish`接口。

参数

- in: 输入数据
- len: 输入数据的长度
- out: 输出数据
- key: algo对应的密钥
- key_len: 密钥长度，不同的algo，支持的密钥长度略有不同

- iv: 初始向量
- algo: 算法类型，支持以下算法(OP-TEE V1版本不支持SM算法)

```
TEE_ALG_AES_ECB_NOPAD
TEE_ALG_AES_CBC_NOPAD
TEE_ALG_AES_CTR
TEE_ALG_AES_CTS
TEE_ALG_AES_XTS
TEE_ALG_SM4_ECB_NOPAD
TEE_ALG_SM4_CBC_NOPAD
TEE_ALG_SM4_CTR
TEE_ALG_DES_ECB_NOPAD
TEE_ALG_DES_CBC_NOPAD
TEE_ALG_DES3_ECB_NOPAD
TEE_ALG_DES3_CBC_NOPAD
```

- mode: 加密或解密模式

14.3.1.8 rk_set_padding

```
TEE_Result rk_set_padding(crypto_ctx_t *ctx, int padding);
```

功能

设置对加解密数据的填充模式。

参数

- ctx: 操作句柄
- padding: 支持的模式见`rk_padding_t`

14.3.1.9 rk_cipher_begin

```
TEE_Result rk_cipher_begin(crypto_ctx_t *ctx, uint8_t *key, uint32_t key_len,
                           uint8_t *iv, uint32_t algo, TEE_OperationMode mode);
```

功能

对称加解密分段算法的初始化操作。

参数

- ctx: 操作句柄
- key: 密钥
- key_len: 密钥长度，不同的algo，支持的密钥长度略有不同
- iv: 初始向量
- algo: 算法类型，支持以下算法(OP-TEE V1版本不支持SM算法)

```
TEE_ALG_AES_ECB_NOPAD
TEE_ALG_AES_CBC_NOPAD
TEE_ALG_AES_CTR
TEE_ALG_AES_CTS
TEE_ALG_AES_XTS
TEE_ALG_SM4_ECB_NOPAD
TEE_ALG_SM4_CBC_NOPAD
TEE_ALG_SM4_CTR
TEE_ALG_DES_ECB_NOPAD
TEE_ALG_DES_CBC_NOPAD
TEE_ALG_DES3_ECB_NOPAD
TEE_ALG_DES3_CBC_NOPAD
```

- mode: 加密或解密模式

14.3.1.10 rk_cipher_update

```
TEE_Result rk_cipher_update(crypto_ctx_t *ctx, uint8_t *in, uint8_t *out,
                            uint32_t in_len, uint32_t *out_len);
```

功能

对称加解密分段算法，对输入的数据进行加解密。

参数

- ctx: 操作句柄
- in: 待加解密的输入数据
- in_len: 输入数据长度
- out: 输出数据
- out_len: 输出数据长度

14.3.1.11 rk_cipher_finish

```
TEE_Result rk_cipher_finish(crypto_ctx_t *ctx, uint8_t *out, uint32_t *out_len);
```

功能

对称加解密分段算法，完成加解密操作。

参数

- ctx: 操作句柄
- out: 输出数据
- out_len: 输出数据长度

14.3.1.12 rk_ae_begin

```
TEE_Result rk_ae_begin(crypto_ctx_t *ctx, uint8_t *key, uint32_t key_len,
                       uint8_t *iv, uint32_t iv_len,
                       uint32_t add_len, uint32_t tag_len,
                       uint32_t payload_len, uint32_t algo, TEE_OperationMode
mode);
```

功能

AES-CCM或AES-GCM算法的初始化操作。

参数

- ctx: 操作句柄
- key: 密钥
- key_len: 密钥长度，支持16，24，32
- iv: 初始向量
- iv_len: 向量长度
- add_len: AES-CCM时的ADD长度
- tag_len: tag长度（bit），AES-GCM支持128, 120, 112, 104, 96; AES-CCM支持128, 112, 96, 80, 64, 48, 32
- payload_len: AES-CCM时的payload长度
- algo: 算法类型，支持`TEE_ALG_AES_GCM, TEE_ALG_AES_CCM`
- mode: 加密或解密模式

14.3.1.13 rk_ae_update

```
TEE_Result rk_ae_update(crypto_ctx_t *ctx, uint8_t *in, uint8_t *out,
                        uint32_t in_len, uint32_t *out_len,
                        rk_ae_update_type_t is_aad);
```

功能

AES-CCM或AES-GCM算法，对输入的数据进行加解密。

参数

- ctx: 操作句柄
- in: 待加解密的输入数据
- in_len: 输入数据长度
- out: 输出数据
- out_len: 输出数据长度
- is_add: 是否有AAD(Additional Authentication Data)

14.3.1.14 rk_ae_finish

```
TEE_Result rk_ae_finish(crypto_ctx_t *ctx, uint8_t *in, uint8_t *out,
                        uint8_t *tag, uint32_t in_len,
                        uint32_t *out_len, uint32_t *tag_len);
```

功能

AES-CCM或AES-GCM算法，完成加解密操作。

参数

- ctx: 操作句柄
- in: 待处理的最后一段输入数据
- in_len: 输入数据长度
- out: 输出数据
- out_len: 输出数据长度
- tag: 输出数据缓冲区填充的tag
- tag_len: tag长度

14.3.1.15 rk_gen_rsa_key

```
TEE_Result rk_gen_rsa_key(rsa_key_t *rsa_key, uint32_t key_len,
                          uint64_t public_exponent);
```

功能

随机生成RSA公私钥对。

参数

- rsa_key: 生成的RSA公私钥对
- key_len: RSA密钥长度（字节），支持32，64，96，128，192，256，384，512
- public_exponent: 指数，支持3，65537

14.3.1.16 rk_rsa_crypto

```
TEE_Result rk_rsa_crypto(uint8_t *in, uint8_t *out, uint32_t len,
                         rsa_key_t *key, uint32_t algo, TEE_OperationMode mode);
```

功能

RSA加密、解密算法。也可使用`rk_rsa_begin/finish`接口实现。

参数

- in: 输入数据
- len: 输入数据长度
- out: 输出数据
- key: RSA密钥
- algo: RSA算法填充模式，支持以下

  ```
  TEE_ALG_RSAES_PKCS1_V1_5
  TEE_ALG_RSAES_PKCS1_OAEP_MGF1_SHA1
  TEE_ALG_RSAES_PKCS1_OAEP_MGF1_SHA224
  TEE_ALG_RSAES_PKCS1_OAEP_MGF1_SHA256
  TEE_ALG_RSAES_PKCS1_OAEP_MGF1_SHA384
  TEE_ALG_RSAES_PKCS1_OAEP_MGF1_SHA512
  TEE_ALG_RSA_NOPAD
  ```

- mode: 加解密模式，`TEE_MODE_ENCRYPT`或`TEE_MODE_DECRYPT`

14.3.1.17 rk_rsa_sign

```
TEE_Result rk_rsa_sign(uint8_t *digest, uint8_t *signature, uint32_t digest_len,
                       uint32_t *signature_len, rsa_key_t *key,
                       uint32_t salt_len, uint32_t algo, TEE_OperationMode mode);
```

功能

RSA签名、验签。也可使用`rk_rsa_begin/finish`接口实现。

参数

- digest: 摘要值
- signature: 签名输出的值，或者输入待验签的值
- digest_len: 摘要长度
- signature_len: 签名值长度
- key: rsa密钥
- salt_len: salt的字节数，此参数是可选的。如果不存在，salt长度等于摘要长度
- algo: 算法，详见GPD_TEE_Internal_Core_API_Specification，Table 6-4
- mode: 模式，`TEE_MODE_SIGN`或`TEE_MODE_VERIFY`

14.3.1.18 rk_set_sign_mode

```
TEE_Result rk_set_sign_mode(crypto_ctx_t *ctx, unsigned int mode);
```

功能

设置RSA签名模式，对数据签名或对摘要签名。

参数

- ctx: 操作句柄
- mode: 数据`SIGN_DATA`或摘要`SIGN_DIGEST`

14.3.1.19 rk_rsa_begin

```
TEE_Result rk_rsa_begin(crypto_ctx_t *ctx, rsa_key_t *key,
                        uint32_t algo, TEE_OperationMode mode);
```

功能

RSA加密、解密、签名、验签算法的初始化操作。

参数

- ctx: 操作句柄
- key: RSA密钥
- algo: 算法填充模式，不同的mode允许的algo不同，详见GPD_TEE_Internal_Core_API_Specification，Table 6-4
- mode: 算法类型，支持`TEE_MODE_ENCRYPT, TEE_MODE_DECRYPT, TEE_MODE_SIGN, TEE_MODE_VERIFY`

14.3.1.20 rk_rsa_finish

```
TEE_Result rk_rsa_finish(crypto_ctx_t *ctx, uint8_t *in, uint8_t *out,
                         uint32_t in_len, uint32_t *out_len, uint32_t salt_len);
```

功能

在`rk_rsa_begin`之后执行的RSA加密、解密、签名、验签算法。

参数

- ctx: 操作句柄
- in: 输入数据
- in_len: 输入数据长度
- out: 输出数据
- out_len: 输出数据长度
- salt_len: salt的字节数，此参数是可选的

14.3.1.21 rk_gen_ec_key

```
TEE_Result rk_gen_ec_key(ec_key_t *ec_key, uint32_t key_len, uint32_t curve);
```

功能

随机生成ECC密钥对。

参数

- ec_key: 生成的ECC公私钥对
- key_len: 密钥长度（bit），支持192，224，256，384，521
- curve: ECC曲线，支持的曲线见`tee_api_defines.h`

14.3.1.22 rk_ecdh_genkey

```
TEE_Result rk_ecdh_genkey(uint8_t *private, uint8_t *publicx, uint32_t *publicy,
                          uint32_t algo,  uint32_t curve,
                          uint32_t keysize, uint8_t *out);
```

功能

进行ECDH，协商对称密钥。

参数

- private: ECC私钥
- publicx: ECC公钥X坐标
- publicy: ECC公钥Y坐标
- algo: 支持`TEE_ALG_ECDH_P192, TEE_ALG_ECDH_P224, TEE_ALG_ECDH_P256, TEE_ALG_ECDH_P384, TEE_ALG_ECDH_P521`
- curve: ECC曲线，支持`TEE_ECC_CURVE_NIST_P192, TEE_ECC_CURVE_NIST_P224, TEE_ECC_CURVE_NIST_P256, TEE_ECC_CURVE_NIST_P384, TEE_ECC_CURVE_NIST_P521`
- keysize: 密钥长度（bit），支持192，224，256，384，521
- out: 输出的对称密钥

14.3.1.23 rk_ecdsa_sign

```
TEE_Result rk_ecdsa_sign(uint8_t *digest, uint8_t *signature,
                         uint32_t digest_len, uint32_t *signature_len,
                         ec_key_t *key, uint32_t algo, TEE_OperationMode mode);
```

功能

ECDSA签名、验签算法。也可使用`rk_ecdsa_begin/finish`接口实现。

参数

- digest: 输入的摘要
- signature: 输出的签名值，或者输入的待验签的值
- digest_len: 摘要长度
- signature_len: 签名值长度
- key: ECC密钥
- algo: 支持`TEE_ALG_ECDSA_P224, TEE_ALG_ECDSA_P256, TEE_ALG_ECDSA_P384, TEE_ALG_ECDSA_P521`
- mode: 支持`TEE_MODE_SIGN, TEE_MODE_VERIFY`

14.3.1.24 rk_ecdsa_begin

```
TEE_Result rk_ecdsa_begin(crypto_ctx_t *ctx, ec_key_t *key,
                          uint32_t algo,TEE_OperationMode mode);
```

功能

ECDSA签名、验签算法的初始化操作。

参数

- ctx: 操作句柄
- key: ECC密钥
- algo: 支持`TEE_ALG_ECDSA_P224, TEE_ALG_ECDSA_P256, TEE_ALG_ECDSA_P384, TEE_ALG_ECDSA_P521`
- mode: 支持`TEE_MODE_SIGN, TEE_MODE_VERIFY`

14.3.1.25 rk_ecdsa_finish

```
TEE_Result rk_ecdsa_finish(crypto_ctx_t *ctx, uint8_t *in, uint8_t *out,
                           uint32_t in_len, uint32_t *out_len);
```

功能

ECDSA签名、验签算法，对输入的摘要进行签名，或者对输入的摘要、签名值进行验签。

参数

- ctx: 操作句柄
- in: 输入的摘要
- out: 输出的签名值，或者输入的待验签的值
- in_len: 摘要长度

- out_len: 签名值长度

14.3.1.26 rk_sm2_pke

```
TEE_Result rk_sm2_pke(uint8_t *in, uint32_t in_len, uint8_t *out,
                      uint32_t *out_len, ec_key_t *key,
                      uint32_t algo, TEE_OperationMode mode);
```

功能

SM2加密、解密。OP-TEE V1版本不支持该接口。

参数

- in: 输入待加密解密的数据
- in_len: 输入数据长度
- out: 输出数据
- out_len: 输出数据长度
- key: SM2密钥
- algo: 算法，支持`TEE_ALG_SM2_PKE`
- mode: 模式，支持`TEE_MODE_ENCRYPT`和`TEE_MODE_DECRYPT`

14.3.1.27 rk_sm2_dsa_sm3

```
TEE_Result rk_sm2_dsa_sm3(uint8_t *digest, uint32_t digest_len,
                          uint8_t *signature, uint32_t *signature_len,
                          ec_key_t *key, uint32_t algo, TEE_OperationMode mode);
```

功能

SM2签名或验签。OP-TEE V1版本不支持该接口。

参数

- digest: SM3摘要
- digest_len: SM3摘要长度，固定为32
- signature: 输出的签名数据，或者输入的待验证的签名数据
- signature_len: 签名数据的长度
- key: SM2密钥
- algo: 算法，支持`TEE_ALG_SM2_DSA_SM3`
- mode: 签名`TEE_MODE_SIGN`或验签`TEE_MODE_VERIFY`

14.3.1.28 rk_sm2_kep_genkey

```
TEE_Result rk_sm2_kep_genkey(rk_sm2_kep_parms *kep_parms, uint8_t *share_key,
                             uint32_t share_key_len, uint8_t *conf_out);
```

功能

基于SM2的ECDH算法。OP-TEE V1版本不支持该接口。

参数

- kep_parms: SM2密钥信息，包含A的私钥和B的公钥
- share_key: 输出ECDH协商的对称密钥
- share_key_len: share_key长度
- conf_out: 用于校验的会话信息

14.3.1.29 rk_mac_crypto

```
TEE_Result rk_mac_crypto(uint8_t *in, uint8_t *out, uint32_t in_len,
                         uint32_t *out_len, uint8_t *key, uint32_t key_len,
                         uint8_t *iv, uint32_t algo);
```

功能

MAC计算。如需分段，可使用`rk_mac_begin/update/finish`接口。

参数

- in: 输入数据
- in_len: 输入数据长度
- out: 输出数据
- out_len: 输出数据长度
- key: MAC密钥
- key_len: 密钥长度
- iv: 初始向量
- algo: MAC算法类型, (OP-TEE V1版本不支持SM算法)， `TEE_ALG_HMAC_MD5, TEE_ALG_HMAC_SHA1, TEE_ALG_HMAC_SHA256, TEE_ALG_AES_CMAC, TEE_ALG_HMAC_SM3`

14.3.1.30 rk_mac_begin

```
TEE_Result rk_mac_begin(crypto_ctx_t *ctx, uint8_t *key, uint32_t key_len,
                        uint8_t *iv, uint32_t algo);
```

功能

MAC分段算法，初始化操作。

参数

- ctx: 操作句柄
- key: MAC密钥
- key_len: 密钥长度
- iv: 初始向量
- algo: MAC算法类型, (OP-TEE V1版本不支持SM算法)， `TEE_ALG_HMAC_MD5, TEE_ALG_HMAC_SHA1, TEE_ALG_HMAC_SHA256, TEE_ALG_AES_CMAC, TEE_ALG_HMAC_SM3`

14.3.1.31 rk_mac_update

```
TEE_Result rk_mac_update(crypto_ctx_t *ctx, uint8_t *in, uint32_t in_len);
```

功能

MAC分段算法，积累输入数据，用于计算MAC。

参数

- ctx: 操作句柄
- in: 输入数据
- in_len: 输入数据长度

14.3.1.32 rk_mac_finish

```
TEE_Result rk_mac_finish(crypto_ctx_t *ctx, uint8_t *in, uint8_t *mac,
                         uint32_t in_len, uint32_t *mac_len, rk_mac_mode_t mode);
```

功能

MAC分段算法，输入最后一段数据计算MAC，或者输入最后一段数据和预期的MAC值，输出校验结果。

参数

- ctx: 操作句柄
- in: 最后一段输入数据
- in_len: 输入数据长度
- mac: 当`mode=RK_MAC_SIGN`时输出MAC值，当`mode=RK_MAC_VERIFY`时，输入预期的MAC值
- mac_len: MAC值的长度
- mode: 见“mac”

14.3.1.33 rk_hkdf_genkey

```
TEE_Result rk_hkdf_genkey(uint8_t *ikm, uint32_t ikm_len,
                          uint8_t *salt, uint32_t salt_len,
                          uint32_t *info, uint32_t info_len,
                          uint32_t algo, uint32_t okm_len, uint8_t *okm);
```

功能

HKDF密钥派生函数。

参数

- ikm: 输入的密码
- ikm_len: 密码长度
- salt: 输入的salt
- salt_len: salt长度
- info: 输入的info
- info_len: info长度
- algo: 支持`TEE_ALG_HKDF_MD5_DERIVE_KEY, TEE_ALG_HKDF_SHA1_DERIVE_KEY, TEE_ALG_HKDF_SHA224_DERIVE_KEY, TEE_ALG_HKDF_SHA256_DERIVE_KEY, TEE_ALG_HKDF_SHA384_DERIVE_KEY, TEE_ALG_HKDF_SHA512_DERIVE_KEY`
- okm_len: 输入的派生的密钥长度
- okm: 派生的密钥

14.3.1.34 rk_pkcs5_pbkdf2_hmac

```
TEE_Result rk_pkcs5_pbkdf2_hmac(uint8_t *password, uint32_t password_len,
                                uint8_t *salt, uint32_t salt_len,
                                uint32_t iterations, uint32_t algo,
                                uint32_t key_len, uint8_t *out_key);
```

功能

通过指定的salt和iteration count以及password，派生密钥。

参数

- password: 输入的密码
- password_len: 密码长度
- salt: 输入的salt
- salt_len: salt长度
- iterations: 输入的iteration count值
- algo: 支持`TEE_ALG_PBKDF2_HMAC_SHA1_DERIVE_KEY`
- key_len: 输入的派生的密钥长度
- out_key: 派生的密钥

14.3.2 TRNG API

14.3.2.1 rk_get_trng

```
TEE_Result rk_get_trng(uint8_t *buffer, uint32_t size);
```

功能

获取硬件随机数。
仅部分平台支持，如果不支持该接口，将返回TEE_ERROR_NOT_SUPPORTED。

参数

- buffer: 返回的硬件随机数，buffer内存空间必须不小于size
- size: 待获取的随机数长度

14.3.3 OTP API

14.3.3.1 rk_otp_size

```
TEE_Result rk_otp_size(uint32_t otp_type, uint32_t *otp_size);
```

功能

获取OTP的总大小，包括已写和未写的大小。支持OEM_HR_OTP和OEM_S_OTP。

参数

- otp_type: OTP类型，支持TYPE_OEM_HR_OTP和TYPE_OEM_S_OTP
- otp_size: 返回的OTP的总大小

14.3.3.2 rk_otp_read

```
TEE_Result rk_otp_read(uint32_t otp_type, uint32_t offset, uint8_t *data,
              uint32_t len);
```

功能

读取OTP数据。支持OEM_HR_OTP和OEM_S_OTP。

参数

- otp_type: OTP类型，支持TYPE_OEM_HR_OTP和TYPE_OEM_S_OTP
- offset: 待读取的OTP区域的位置偏移
- data: 返回的OTP数据，data内存空间必须不小于len
- len: 待读取的长度

14.3.3.3 rk_otp_write

```
TEE_Result rk_otp_write(uint32_t otp_type, uint32_t offset, uint8_t *data,
               uint32_t len);
```

功能

写数据到OTP。支持OEM_HR_OTP和OEM_S_OTP。

参数

- otp_type: OTP类型，支持TYPE_OEM_HR_OTP和TYPE_OEM_S_OTP
- offset: 待写入到OTP区域的位置偏移
- data: 待写入到OTP的数据
- len: data长度

15. 相关资料扩展

ARM官方TrustZone

https://developer.arm.com/ip-products/security-ip/trustzone

GlobalPlatform官网：

https://globalplatform.org/

该网站可下载CA开发API参考文档：`TEE Client API Specification`

TA开发API参考文档：`TEE Internal Core API Specification`

以及其他架构方面参考文档。