密级状态：绝密(　　)　　秘密(　　)　　内部资料(　　)　　公开(√)

Rockchip TEE 安全 SDK 开发手册

(软件创新中心)

文件状态：	文件标识：	Rockchip TEE 安全 SDK 开发手册
［　］草稿	当前版本：	1.1
［√］正式发布	作　　者：	黄成就、张志杰
［　］正在修改	完成日期：	2017-5-8

版 本 历 史

版本号	作者	修改日期	修改说明
V1.0	黄成就	2016-12-9	初始版本
V1.1	张志杰	2017-5-8	更新相关说明

目 录

1. TRUSTZONE 简介 4

1.1 什么是 TRUSTZONE 4

1.2 TRUSTZONE 软硬件架构 5

1.2.1 硬件架构 5

1.2.2 软件架构 6

1.3 TRUSTZONE 与 TEE 7

2. TEE 环境 8

2.1 TEE 固件 8

2.2 TEE 库文件 9

3. CA/TA 开发与测试 9

3.1 目录介绍 9

3.2 编译开发说明 9

3.3 运行测试 TEE 环境 10

3.4 开发 CA/TA 10

4. TA 签名方法 10

4.1 签名 TA 过程 10

4.2 验证 TA 过程 11

5. TA 调试方法 11

6. 内存相关说明 13

7. 相关资料扩展 14

8. 注意事项 15

1. TrustZone 简介

1.1 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。

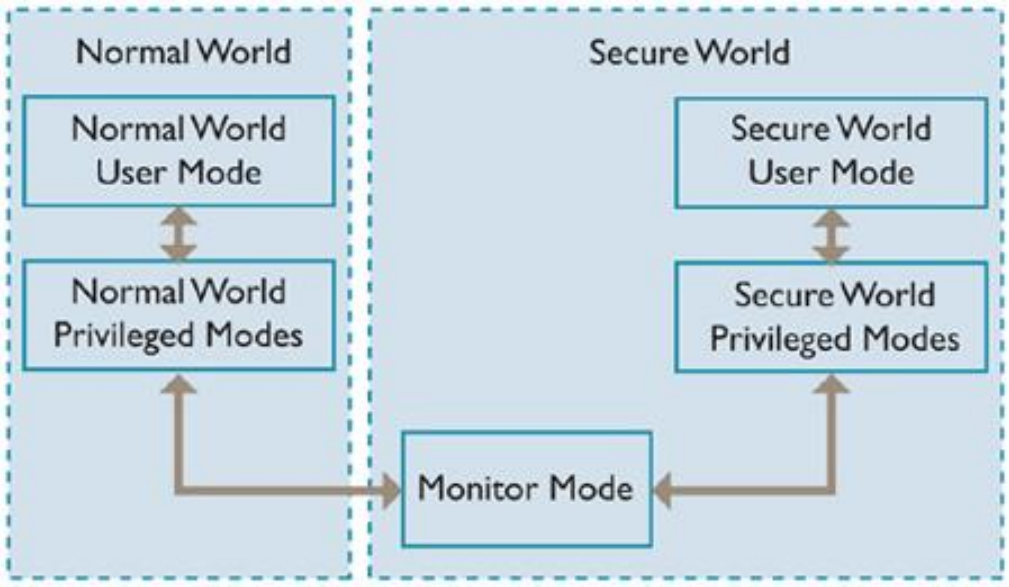
可信应用程序采用基 TrustZone 技术的 SoC（运行可信执行环境），与主 OS 分开，可防止软件/恶意软件攻击。TrustZone 可切换到安全模式，提供硬件支持的隔离。可信应用程序通常是可集装箱化的，如允许不同支付公司的可信应用程序共存于一台设备上。处理器支持 ARM TrustZone 技术是所有 Cortex-A 类处理器的基本功能，是通过 ARM 架构安全扩展引入的。这些扩展可在供应商、平台和应用程序中提供一致的程序员模型，同时提供真实的硬件支持的安全环境。

1.2 TrustZone 软硬件架构

1.2.1 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

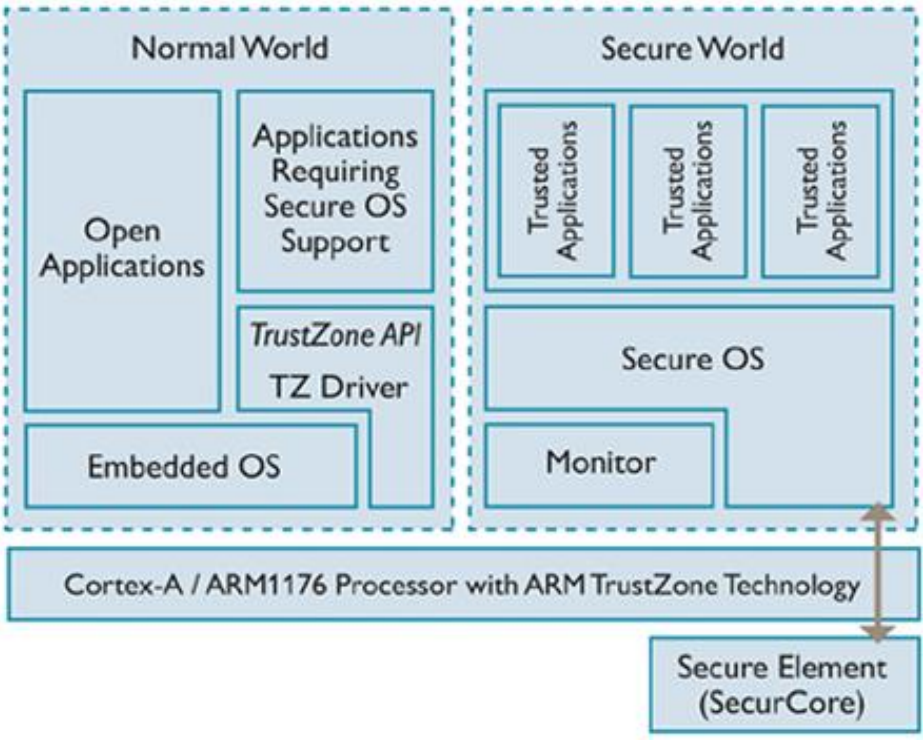
加的扩展，单个物理处理器内核能够以时间片的方式安全有效地同时从普通区域和安全区域执行代码。这样，便无需使用专用安全处理器内核，从而节省了芯片面积和能源，并且允许高性能安全软件与普通区域操作环境一起运行。

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。

1.2.2 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。

1.3 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。

2. TEE 环境

2.1 TEE 固件

TEE OS 的源码不开源，binary 位于安卓工程目录 u-boot/tools/rk_tools/bin 下。

1) arm v7 平台（RK312x，RK3288，RK3228）的 TEE binary 由工具 u-boot/tools/rk_tools/loaderimage 打包成固件 trust.img，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) arm v8 平台（RK3368，RK3399，RK3228H，RK3328）的 TEE binary 由工具 u-boot/tools/rk_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

2.2 TEE 库文件

TEE 环境相关组件在安卓工程目录 vendor/rockchip/common/security 下：

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. CA/TA 开发与测试

3.1 目录介绍

TEE CA/TA 开发环境在安卓工程目录 system/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-user_ta：存放编译 ta 所依赖的环境。

3.2 编译开发说明

cd system/rk_tee_user/

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 目录下 system/bin 中，rkdemo 与

rkdemo_storage 为 RK 编写的 demo 程序。

TA 是文件名为 uuid，后缀为.ta 的文件，编译后生成于 rk_tee_user/ta 下对应的文件夹中。

TA 文件需放置到设备的 system/lib/optee_armtz（注：若无 optee_armtz 目录，则需要新建）下，再执行 CA 程序。

3.3 运行测试 TEE 环境

1. adb shell 进入设备
2. libteec.so 放置到/system/lib 或 lib64 目录下，tee-supplicant，rkdemo 放置到/system/bin 目录下，8cccf200-2450-11e4-abe20002a5d5c52c.ta 放置到/system/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）
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 &
4. 运行 rkdemo，成功提示 PASS，失败提示 Fail：

   # rkdemo
5. 若 rkdemo 运行通过，则 TEE 环境正常，可进行 TEE 相关开发。
6. 同时，可利用 rkdemo_storage 测试 Secure Storage 环境是否正常。

3.4 开发 CA/TA

可参考 rkdemo。

4. TA 签名方法

4.1 签名 TA 过程

在编译 TA 时，编译脚本将使用 rk_tee_user/export-user_ta/keys 目录下的 default_ta.pem 密钥对 TA 镜像进行签名，该密钥为 pem 格式的 2048 长度 RSA 密钥，以下为编译脚本中签名 TA 过程：

SIGN = $(TA_DEV_KIT_DIR)/scripts/sign.py

TA_SIGN_KEY ?= $(TA_DEV_KIT_DIR)/keys/default_ta.pem

(q)(SIGN) --key $(TA_SIGN_KEY) --in $< --out $@

为防止客户 A 的 TA 应用运行在客户 B 的板子上，建议客户生成一个 2048 长度 RSA 密钥，替换 rk_tee_user/export-user_ta/keys 目录下的 default_ta.pem 密钥。

4.2 验证 TA 过程

在加载运行 TA 时，TEE OS 将验证 TA 的合法性，验证通过才能正常运行 TA 应用。由于客户替换了签名 TA 的密钥，则 TEE OS 中用于验证 TA 合法性的公钥也需要随之替换，客户可以使用工具替换 TEE binary（参考第一章第一节）中的公钥。

1) 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

使用客户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。

2) 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镜像，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。

5. TA 调试方法

当 TA 出现异常时会打印如下信息。

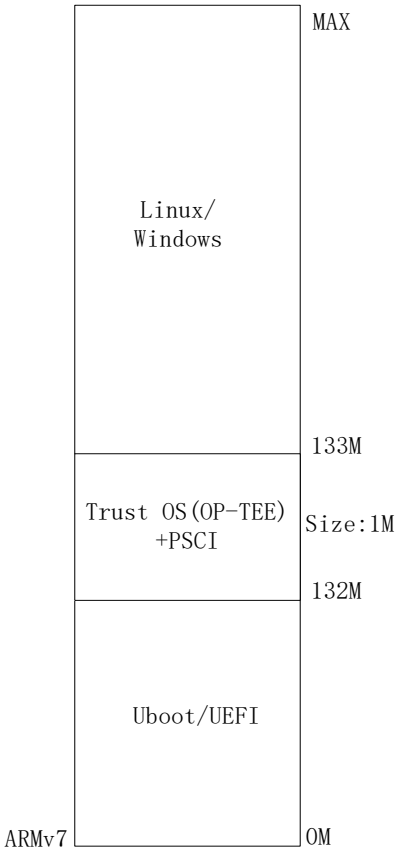
```
user TA data-abort at address 0x8888

 fsr 0x00000805  ttbr0 0x6846c46a  ttbr1 0x6846806a  cidr 0x2

 cpu #0          cpsr 0x00000030

 r0 0x60000013      r4 0x001007b8    r8 0x68471754   r12 0x000000ab

 r1 0x0000003a      r5 0x00200da9    r9 0x68415491    sp 0x00100720

 r2 0x00000031      r6 0x001005a0   r10 0x00000000    lr 0x0020265f

 r3 0x00008888      r7 0x00100728   r11 0x00000000    pc 0x00200104

Status of TA 8cccf200-2450-11e4-abe20002a5d5c52c (0x68467450) (active)
- load addr : 0x200000    ctx-idr: 2
- code area : 0x68700000 1048576
- stack: 0x68800000 stack:2048
DBG [0x0] TEE-CORE:get_fault_type:455: [abort] abort in User mode (TA will panic)
DBG [0x0] TEE-CORE:user_ta_enter:465: tee_user_ta_enter: TA panicked with code
0xdeadbeef
```

图中 pc 0x00200104 就是异常位置。进入 rkdemo 目录下，输入下面命令 arm-eabi-objdump －S 8cccf200-2450-11e4-abe20002a5d5c52c.elf | less 得到反汇编信息，由于 TA 的运行地址从 2M 位置开始，所以在反汇编信息中搜索 104（PC － 0x200000），得到如下图反汇编信息，图中红色就是异常位置。

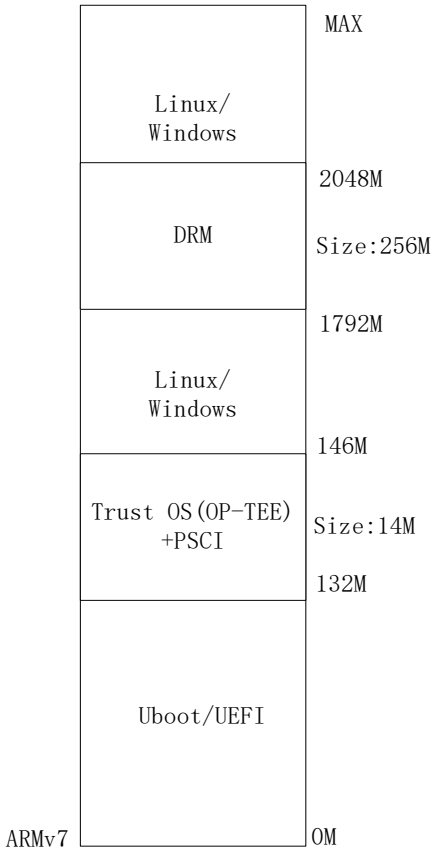
```
      e8:       4b3f            ldr     r3, [pc, #252]  ; (1e8
<TA_InvokeCommandEntryPoint+0x128>)
      ea:       447b            add     r3, pc
      ec:       9300            str     r3, [sp, #0]
      ee:       4b3f            ldr     r3, [pc, #252]  ; (1ec
<TA_InvokeCommandEntryPoint+0x12c>)
      f0:       447b            add     r3, pc
      f2:       4618            mov     r0, r3
      f4:       215e            movs    r1, #94 ; 0x5e
      f6:       2202            movs    r2, #2
      f8:       2301            movs    r3, #1
      fa:       f004 ff8d       bl      5018 <trace_printf>
        *(char*)0x8888 = '1';
      fe:       f648 0388       movw    r3, #34952      ; 0x8888
     102:       2231            movs    r2, #49 ; 0x31
     104:       701a            strb    r2, [r3, #0]
        IMSG("=========2=====");
     106:       4b3a            ldr     r3, [pc, #232]  ; (1f0
<TA_InvokeCommandEntryPoint+0x130>)
     108:       447b            add     r3, pc
     10a:       9300            str     r3, [sp, #0]
     10c:       4b39            ldr     r3, [pc, #228]  ; (1f4
<TA_InvokeCommandEntryPoint+0x134>)
     10e:       447b            add     r3, pc
     110:       4618            mov     r0, r3
     112:       2160            movs    r1, #96 ; 0x60
     114:       2202            movs    r2, #2
     116:       2301            movs    r3, #1
     118:       f004 ff7e       bl      5018 <trace_printf>
```

6. 内存相关说明

1）ARMv7 架构（RK312x，RK3288，RK3228）内存分配情况如下：



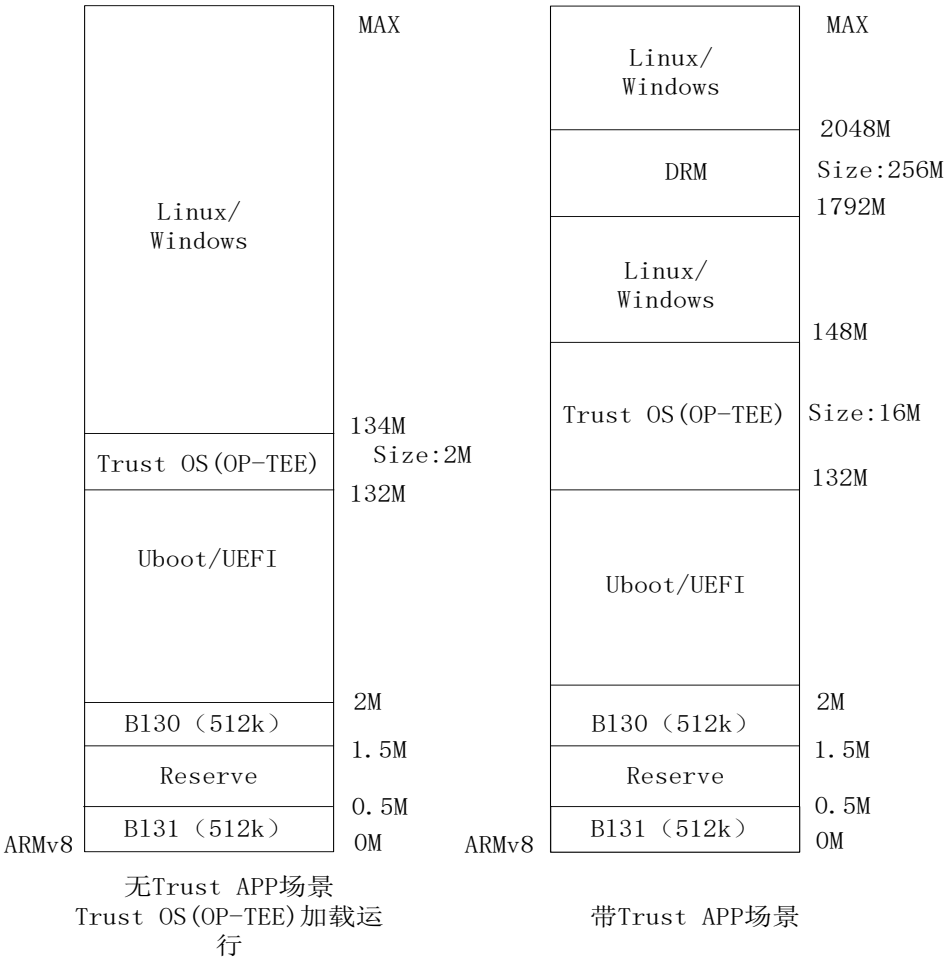
无Trust APP(如:DRM)场景



带Trust APP(如:DRM)场景

2）ARMv8 架构（RK3368，RK3399，RK3228H，RK3328）内存分配情况如下：

V8 架构下的的 BL30,BL31,BL32(Trust os)内存分配情况。



2）TEE 内存分配情况如下：

2M	TEE_RAM
12M	TA_RAM
2M	SHMEM

说明：1. 以上是 v8 的架构下 TEE 内存分布，TA 运行在 TA_RAM 区域，运行一个 TA 占用 4M 内存大小

2. 在 v7 的架构下，TEE 内存分布为 14M，TEE_RAM 和 SHMEM 区域分别是 1M。TA 运行在 TA_RAM ,运行一个 TA 占用 2M 内存大小

7. 相关资料扩展

1. ARM 官方 TrustZone 白皮书：

http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_tr

ustzone_security_whitepaper.pdf

2. GlobalPlatform 官方文档：

   https://www.globalplatform.org/specificationsdevice.asp

   该网站可下载 CA 开发 API 参考文档：TEE Client API Specification v1.0

   TA 开发 API 参考文档：TEE Internal Core API Specification v1.1

   以及其他架构方面参考文档。

8. 注意事项

1. 每次开机后需先在后台执行 tee-supplicant，然后 CA/TA 才可实现交互。
2. 开发新的 TA 时，TA 的 UUID 需采用标准的 UUID，可用 uuidgen 命令生成。
3. 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)