UG10158

i.MX Android安全用户指南

版本android-14.0.0_2.2.0—2024年10月18日 **用户指南**

文档信息

信息	内容
关键词	Android、i.MX、android-14.0.0_2.2.0
摘要	本指南介绍了如何对i.MX Android软件支持的安全功能进行个性化定制。

1 前言

1.1 关于本文档

本指南介绍了如何对i.MX Android软件支持的安全功能进行个性化定制，概述了i.MX Android的安全功能，并重点介绍了如何配置和使用这些安全功能。

发布的代码可以嵌入到集成了或未集成Trusty OS的镜像，而本文中与Trusty OS相关的内容只适用于集成了Trusty OS的镜像。

不同的SoC可能具有不同的安全硬件模块。本文提供了相关的详细信息。特定SoC的用户只需关注与该SoC有关的信息。如果未明确指定SoC类型，请参阅适用于所有SoC的通用说明。

1.2 恩智浦安全免责声明

- 恩智浦在i.MX系列的文档中记录了如何正确配置安全IP。
- 无法提供一种通用的Trusty OS安全配置来满足所有用户的需求，并且Trusty OS的安全性会在每个版本中逐步增强。
- 因此，用户应根据其特定的安全要求自定义Trusty OS安全配置，以锁定和保护其最终产品。
- 恩智浦提供Trusty OS，包括其Android空间安全模块，作为开源软件支持，而非量产型安全实现方案。
- 使用Google AOSP Trusty OS源代码而非恩智浦Trusty OS源代码可能会对i.MX平台支持的功能和安全级别产生影响。

1.3 约定

本文使用了以下约定：

- 软件代码以`Consolas`字体显示。
- `${MY_ANDROID}`表示i.MX Android源代码根目录。
- `${MY_TRUSTY}` 表示i.MX Trusty OS源代码根目录。

2 i.MX Android安全功能概述

2.1 安全相关硬件模块介绍

安全功能基于安全相关代码，这些代码需要进行一些加密计算来保护安全数据。要保证安全性，只允许安全相关代码在部分硬件资源上运行。因此，这些硬件资源被称为安全相关硬件模块。i.MX平台上有一些安全硬件模块，它们与Trusty OS协同工作以保证安全：

- CAAM：加密加速和保证模块是片上系统（SoC）的一个硬件组件，为加密算法、数据包封装和解封装以及其它加密操作提供安全保证和硬件加速。
- TrustZone：Arm TrustZone创建了一个隔离的安全环境，可用于为系统提供机密性和完整性。它能够保护用于身份验证等不同用例的高价值代码和数据。它常用于为可信执行环境（如Trusty OS）提供安全边界。
- TZASC：TrustZone地址空间控制器是一种符合高级微控制器总线架构（AMBA）规范的SoC外设。它是一种高性能、区域优化的地址空间控制器，用于保护可信执行环境中的安全敏感软件和数据，防止在平台上运行的软件受到潜在威胁。

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

- CSU：中央安全单元（CSU）设置总线主设备和总线从设备之间的访问控制策略，能够将外设划分到不同的安全域中。
- RDC：资源域控制器（RDC）支持将目的存储映射位置（如外设和存储器）隔离到单个内核、总线主设备或一组内核和总线主设备。
- xRDC：在i.MX 8QuadMax和i.MX 8QuadXPlus上，扩展资源域控制器（xRDC）取代了之前i.MX处理器中的RDC和TrustZone组件（CSU、TZASC等）。
  i.MX 8QuadMax和i.MX 8QuadXPlus SoC包含Cortex-A和Cortex-M CPU的组合，它们通常在不同的软件环境下以非对称模式运行。为了防止这些软件环境之间无意的相互干扰，SoC包含xRDC来强制隔离。xRDC的工作方式类似于Arm的TrustZone。来自主设备的事务会带有用户侧带信息，表明来自哪个功能域，访问控制逻辑根据此信息允许/禁止对外设/存储器的访问。
- TRDC：可信资源域控制器（TRDC）取代了之前i.MX处理器中的RDC和TrustZone组件（CSU、TZASC等）。
  TRDC具有与xRDC相似的功能。该芯片使用基于域的资源控制架构来实现Cortex-A平台、Cortex-M核和其它总线主设备之间的存储器/外设资源共享和隔离。此架构的一个关键特征是，无论是Cortex-A平台、Cortex-M平台还是ELE，都可以对资源域控制访问列表进行编程，以实现客户定义的策略。任何外设都可以分配到任意域。TRDC支持多个区域，对每个存储空间都具有灵活的访问权限控制。
- 系统管理器（SM）是一种在系统控制处理器（SCP）上运行的低级系统功能，用于支持电源域、时钟、复位、传感器、引脚等的隔离和管理。它通常在Cortex-M处理器上运行，并为许多底层硬件功能提供抽象。系统管理器的主要目的是支持隔离在SoC中不同内核上运行的软件。它通过独占访问控制电源、时钟、复位和PMIC等关键资源，并向这些客户端提供RPC接口，从而实现隔离。这样，系统管理器能够为这些共享的关键资源提供访问控制、仲裁和聚合策略。
- AHAB/HABv4：高级高可靠性启动（AHAB）和高可靠性启动（HABv4）通过加密操作来支持对镜像的身份验证，以防止在芯片启动过程中执行非法软件。有关如何使用HAB验证镜像的详细信息，请参见第2.1章。
- SCU：系统控制器单元（SCU）仅适用于i.MX 8QuadMax和i.MX 8QuadXPlus平台。它由一个Cortex-M4处理器和一组外设与接口组成，用于连接外部PMIC并控制内部子系统。SCU Cortex-M4是第一个启动芯片的处理器。SCU专用于：
  - 启动管理
  - 电源管理
    - 通过与外部PMIC通讯进行外部电源管理
    - 所有子系统的内部电源管理
  - 时钟和复位管理
  - I/O配置和多路复用
  - 资源分区/访问控制
- SECO：安全控制器子系统（SECO）仅适用于i.MX 8QuadMax和i.MX 8QuadXPlus平台。它管理多个安全硬件模块（CAAM、SNVS、OTP、ADM等），以执行加密加速并确保整个系统的安全。
- ELE：EdgeLock安全飞地（ELE）在嵌入式安全飞地的基础上，将平台安全服务引入SoC。无源安全元件子系统基本上是保护自己，然后执行加密和密钥管理功能，而ELE则增加了新的基于信任的服务，如SoC安全启动、调试域、SoC安全IP和的分布式子系统安全状态管理、熔丝的低功耗高级信任根管理和生命周期开始安全决策、支持优化的快速启动（汽车）、认证、篡改检测和响应以及其它功能。

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

- eMMC RPMB：RPMB是eMMC器件中的独立物理分区，专为安全数据存储而设计。对RPMB的每次访问都经过身份验证，允许主机通过身份验证和重放保护将数据存储到该区域。
  在Trusty OS中，RPMB分区作为安全存储区进行管理，用于存储所有关键数据，如锁定/解锁状态、回滚索引等。

下表列出了i.MX 8QuadMax、i.MX 8QuadXPlus、i.MX 8M Mini和i.MX 8M Quad平台上的模块：

表1. i.MX 8QuadMax、i.MX 8QuadXPlus、i.MX 8M Mini和i.MX 8M Quad平台上的模块

模块	i.MX 8QuadMax和8Quad XPlus	i.MX 8M Quad、8M Mini、8M Nano和8M Plus	i.MX 8ULP	i.MX 95
CAAM	Y	Y	Y	N
TZASC	N	Y	N	N
CSU	N	Y	N	N
RDC	N	Y	N	N
xRDC	Y	N	Y	N
TRDC	N	N	N	Y
系统管理器（System Manager）	N	N	N	Y
AHAB	Y	N	Y	Y
HABv4	N	Y	N	N
SCU	Y	N	N	N
SECO	Y	N	N	N
ELE	N	N	Y	Y
eMMC	Y	Y	Y	Y

2.2 i.MX 8 SoC配置的Trusty OS安全建议

2.2.1 安全CSU配置

中央安全单元（CSU）管理SoC上外设访问的系统安全策略。CSU允许可信代码使用8个安全访问权限级别，为每个外设设置单独的安全访问权限。CSU可根据编程策略在总线事务期间分配总线主设备的安全访问权限。CSU有两个主要的安全相关功能：

- 外设访问策略：访问每个外设都需要适当的总线主设备权限和身份。
- 主设备权限策略：CSU覆盖总线主设备权限信号（安全/非安全）。

在i.MX 8M平台上，Trusty OS和其它软件组件（如ATF、SPL和可能的U-Boot（如果在安全环境中运行））可以访问CSU寄存器，并配置或覆盖外设访问和主设备权限策略。默认情况下，安全代码（CSU驱动程序）可能具有非安全CSU配置。应根据最终应用程序/终端产品的安全要求，将CSU配置从默认的非安全配置改为安全配置。

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

- 应适当设置控制外设访问策略的CSU_CSLn寄存器和控制主设备权限策略的CSU_SA寄存器。
- 所有CSU寄存器配置应使用适当的位域锁定，以防止在运行时对CSU进行任何修改。
- CSU可能需要在电源状态转换时重新配置，这取决于i.MX平台和低功耗支持情况。

注:

- *自 imx-android-11.0.0_2.2.0以来，Trusty OS和 ATF已经创建了一些默认的 CSU配置以供参考。*
- *有关更多信息，请参阅相应《SoC i.MX安全参考手册》中的 CSU章节和“安全集成”章节。*
- *提供了《i.MX安全检查清单》，应仔细阅读，以确保最终产品得到正确保护。*

2.2.2 安全TZASC配置

TrustZone地址空间控制器（TZASC）在可信执行环境中保护安全敏感的软件和数据，防止在平台上运行的软件受到潜在威胁。i.MX 8M平台使用TZC-380控制器。

在i.MX 8M平台上，Trusty OS和其他软件组件（如ATF、SPL和可能的U-Boot（如果在安全环境中运行））可以访问TZASC寄存器，并配置或覆盖存储区域。默认情况下，安全代码可以具有非安全TZASC配置。应根据最终应用程序/终端产品的安全要求，将TZASC配置从默认的非安全配置修改为安全配置：

- 设置存储区域以满足安全要求。
- 区域0应设置为安全，以防止通过别名存储范围访问安全存储区。
- 存储区域配置必须锁定，以防止运行时修改。
- 根据平台的不同，TZASC可能需要在电源状态转换时进行重新配置。

注:

- *从imx-android-11.0.0_1.0.0开始，默认ATF具有将 Trusty OS存储区域配置为受保护TZASC的参考代码。*
- *有关更多信息，请参阅《SoC i.MX安全参考手册》和《CoreLink TrustZone地址空间控制器TZC-380技术参考手册》。*
- *提供了《i.MX安全检查清单》，应仔细阅读，以确保最终产品得到正确保护。*

2.2.3 安全OCRAM配置

OCRAM（片上RAM）是嵌入在SoC上的一个小型存储器。OCRAM的大小根据i.MX SoC的不同而变化。与DRAM一样，OCRAM存储器可以通过配置由TrustZone保护。OCRAM支持TrustZone和非TrustZone访问，并且可以选择配置为仅限TrustZone的访问区域。CSU可用于设置OCRAM存储器的访问权限，并且具有可编程锁定位，以防止运行时修改。

在i.MX 8M平台上，Trusty OS和其它软件组件（如ATF、SPL和可能的U-Boot（如果在安全环境中运行））可以配置或覆盖外设访问和主设备权限策略。然而，安全和非安全环境都可以访问GPR寄存器。应根据最终应用程序/最终产品的安全要求，将OCRAM配置从默认的非安全配置修改为安全配置：

- 查看安全CSU配置建议。
- 必须设置CSU以保护OCRAM。

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

- 必须设置相应的GPR寄存器，以便TrustZone能够保护OCRAM。
- 用相应的锁定位锁定GPR和CSU寄存器，以防止运行时修改。

注：

- *i.MX 8M Plus、8M Quad和8M Nano在ATF中已进行了此配置。*
- *有关更多信息，请参阅《SoC i.MX安全参考手册》中的CSU章节和“安全集成”章节。*
- *提供了《i.MX安全检查清单》，应仔细阅读，以确保最终产品得到正确保护。*

2.2.4 安全RDC配置

资源域控制器（RDC）支持将目的存储映射位置（如外设和存储器）隔离到单个内核、总线主设备或一组内核和总线主设备。RDC提供了一种机制，允许启动时配置代码通过给内核、总线主设备、外设和存储区域分配域标识符来建立资源域。配置完成后，总线事务会被监测，以限制由内核和总线主设备发起的对其各自外设和存储器的访问。

在i.MX 8M平台上，安全和非安全环境都可以访问RDC寄存器。应根据最终应用程序/最终产品的安全要求，将RDC配置从默认的非安全配置修改为安全配置：

- 查看默认和非安全RDC配置的代码。
- 根据您的安全要求设置RDC配置。
- 确保设置不会与CSU和AIPSTZ配置冲突。
- 用相应的锁定位锁定RDC设置，以防止运行时修改。

注：

- *有关更多信息，请参阅《SoC i.MX安全参考手册》中的RDC章节和“安全集成”章节。*
- *提供了《i.MX安全检查清单》，应仔细阅读，以确保最终产品得到正确保护。*

2.2.5 安全AIPSTZ配置

AIPSTZ是一种外设，可用作AHB总线与带有低带宽IP从设备（IPS）总线的外设之间的桥梁。AIPSTZ桥为主设备和外设提供可编程的访问保护。它允许覆盖主设备的权限级别，强制其使用用户模式权限，并允许将主设备指定为可信或不可信。

在i.MX 8M平台上，安全和非安全环境都可以访问AIPSTZ寄存器。应根据最终应用程序/最终产品的安全要求，将AIPSTZ配置从默认的非安全配置修改为安全配置：

- 查看默认和非安全AIPSTZ配置的代码。
- 根据您的安全要求设置AIPSTZ配置。
- 确保设置不会与CSU和RDC配置冲突。
- 用相应的锁定位锁定AIPSTZ设置，以防止运行时修改。

注：

- *有关更多信息，请参阅《SoC i.MX参考手册》和《SoC i.MX安全参考手册》中的AIPSTZ章节。*
- *提供了《i.MX安全检查清单》，应仔细阅读，以确保最终产品得到正确保护。*

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

2.2.6 SCU/SCFW

在i.MX 8QuadMax和8QuadXPlus平台上，在系统控制单元（SCU）上运行的系统控制固件（SCFW）维护电源、时钟和安全组件，如xRDC和TZASC。其它软件组件可以通过SCFW API与SCU进行通讯。

默认情况下，Trusty OS运行在安全分区中运行，这在ATF中进行了配置。有关更多信息，请参阅SCFW API手册。

2.3 安全框架

i.MX Android/Android Automotive安全框架包括安全增强型U-Boot、Android/Android Auto、i.MX Trusty OS和相关硬件。

安全增强型U-Boot提供了Android验证启动模块、密钥预处理接口和安全存储代理。

Android验证启动对最终用户保证通过安全增强型U-Boot加载和启动的软件的完整性。这是由谷歌定义的，更多详细信息可以在https://source.android.com/security/verifiedboot/avb找到。

密钥预处理接口提供RPMB密钥、密钥认证和AVB密钥预处理接口。这些接口可用于将密钥注入芯片以确保其安全。

安全存储代理（Secure Storage Proxy）是Trusty OS安全存储服务的客户端。它有助于通过SoC IP访问RPMB安全存储设备。

Android/Android Auto平台基于谷歌的设计，集成了Keymaster HAL、Gatekeeper HAL和安全存储代理。

Keymaster HAL使用Trusty支持的版本，支持Keymaster V2和Keymaster V3 API。有关Keymaster的更多信息，请参阅https://source.android.com/security/keystore。

Gatekeeper也使用Trusty支持的Gatekeeper HAL。有关Gatekeeper的更多信息，请参阅 https://source.android.com/security/authentication/gatekeeper。

i.MX Trusty OS基于谷歌发布的Trusty OS，集成了安全的TA和服务。Trusty OS是i.MX Android/Android Auto平台整体安全性的一个重要模块。

Trusty OS提供了trusty-ipc，用于实现安全环境和非安全环境之间的通讯。Trusty OS具有面向CAAM的硬件驱动程序，用于keyblob计算和安全算法加速。

下图所示为这些组件之间的逻辑关系。

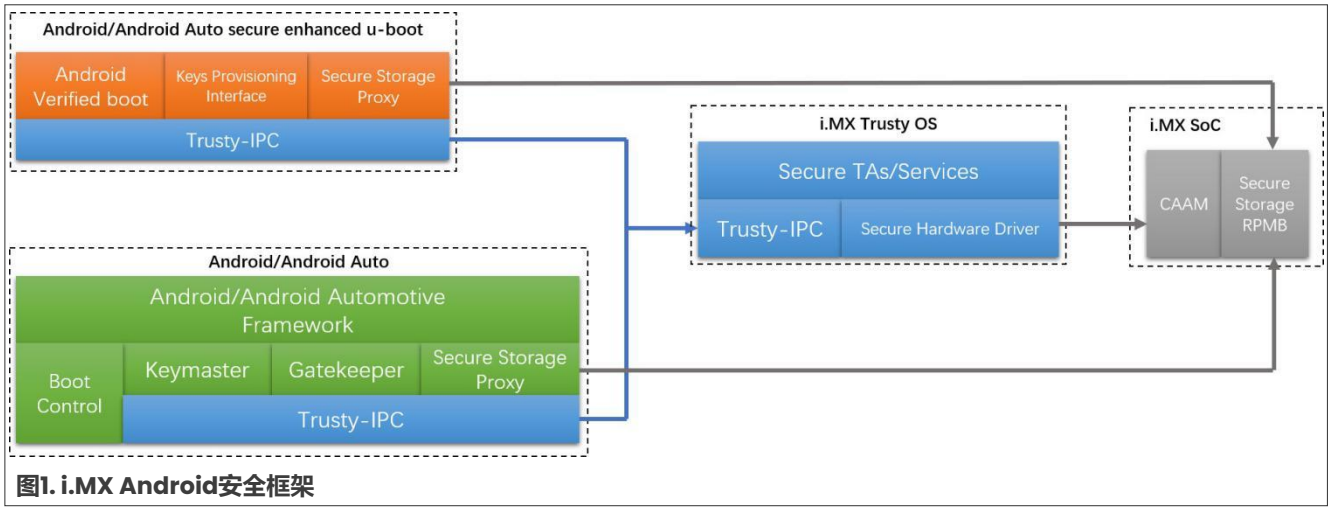


图1. i.MX Android安全框架

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

下图所示为i.MX Android/Android Auto安全信任链。

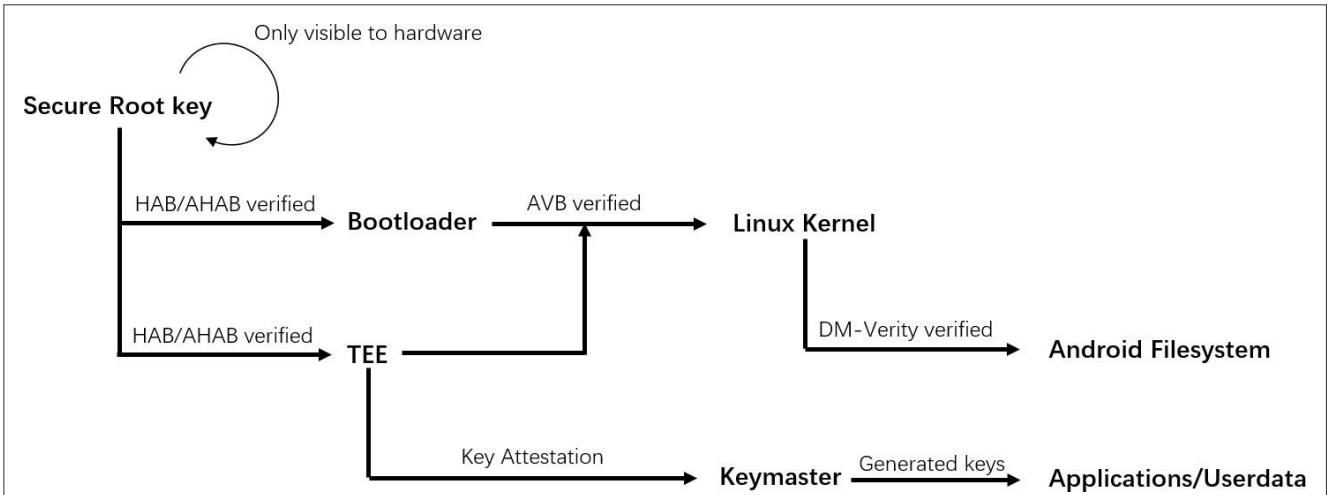


图2. i.MX Android安全信任链

安全根密钥被烧写到i.MX芯片中的一次性可编程（OTP）eFuse硬件中，并作为解决方案的根信任。CAAM使用它来生成其它密钥。在信任链中，HAB/AHAB、AVB和DM-Verity在不同级别上用于验证特定的镜像或加密用户数据。

在上电后，启动流程开始，U-Boot和Trusty OS由ROM代码加载。它们首先由ROM代码使用HAB/AHAB进行验证。只有通过验证后才能执行。U-Boot加载Linux内核，并在跳转到Linux内核之前使用AVB进行验证。Linux内核挂载Android文件系统。从Android文件系统访问的数据将由DM-Verity进行验证，以确保完整性。这些功能形成了安全链。

3 自定义i.MX Android安全功能

3.1 使用HAB验证镜像

i.MX系列应用处理器在片上ROM中提供了高可靠性启动（HAB）功能。ROM负责加载初始镜像并在执行镜像之前对其进行验证。

由于新架构的原因，启动i.MX 8/9系列芯片需要多个固件和软件镜像。恩智浦定义了“容器”来组织这些镜像。适用于i.MX 8Quad、i.MX 8ULP和i.MX 95芯片的AHAB可以识别“容器”的格式并验证容器中的镜像。对于i.MX 8M芯片，这些镜像以扁平镜像树（FIT）的格式存储，并具有适当的镜像向量表（IVT）集。适用于i.MX 8M的HABv4能够识别这种格式并验证镜像。默认情况下，启用HAB验证功能，并且i.MX芯片处于开启阶段，因此HAB验证失败不会阻止启动流程。关闭芯片后，只有正确签名的镜像才能执行。

本节介绍了适用于i.MX 8Quad、i.MX 8ULP和i.MX 95系列芯片的AHAB以及适用于i.MX 8M系列芯片的HABv4。

3.1.1 使用AHAB进行安全启动

AHAB与“容器”紧密相关。有关“容器”的详细信息，请参阅相关芯片的参考手册。根据参考手册，多个固件和软件组件的哈希值存储在容器头中。下面介绍的容器签名流程将在容器中嵌入一个SRK表，并对容器进行签名。签名流程中描述的信息用于在启动期间验证容器。

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

对于i.MX 8QuadMax MEK和i.MX 8QuadXPlus MEK，“容器”用于组织SECO固件、SCU固件、MCU软件、Arm Trusted Firmware、Trusty OS和U-Boot。对于i.MX 8ULP和i.MX 95，“容器”用于组织ELE固件、Upower固件、MCU软件、Arm Trusted Firmware、Trusty OS和U-Boot。

i.MX 8Quad上的SECO固件或i.MX 8ULP和i.MX 95上的ELE固件始终位于第一个容器中。当U-Boot启用SPL时，使用3个容器来组织镜像。如果未启用SPL，则使用两个容器。下面以i.MX 8QuadMax为例介绍镜像的不同布局。

i.MX 8Quad平台的UUU U-Boot镜像未启用SPL，因此使用了两个容器。第一个容器包含SECO固件，该固件以二进制文件形式提供并由恩智浦签名。第二个容器包含SCU固件、Arm Trusted Firmware和U-Boot。它在构建时构造，并附加到第一个容器的末尾。以`u-boot-imx8qm-mek-uuu.imx`为例，其高层布局结构如下图所示。

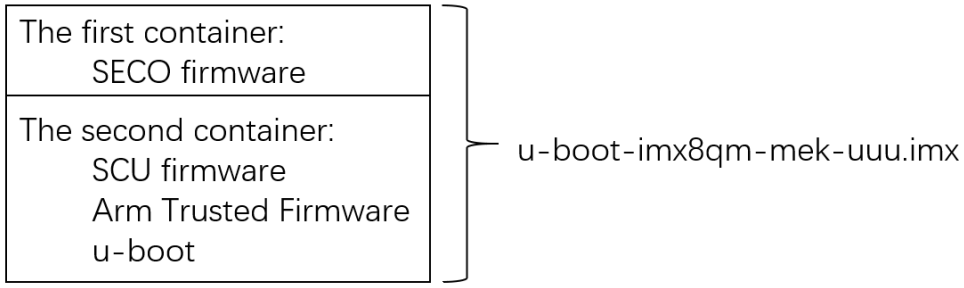


图3. 未启用SPL时的布局结构

与i.MX 8Quad不同，i.MX 8ULP和i.MX 95的UUU U-Boot镜像启用了SPL，因此包含3个容器。

i.MX 8Quad上要烧写到电路板上的普通U-Boot镜像启用了SPL，因此使用了3个容器。第一个容器包含SECO固件。第二个容器在构建时构造，并附加到第一个容器。该容器包含SPL。第三个容器也是在构建时构造的，包含U-Boot本身。默认情况下，构建时构造的两个容器未签名。如果启用了双引导加载程序，则这3个容器位于两个文件中。对于单引导加载程序情况，以标准Android镜像的`u-boot-imx8qm.imx`为例。下图所示为布局结构。

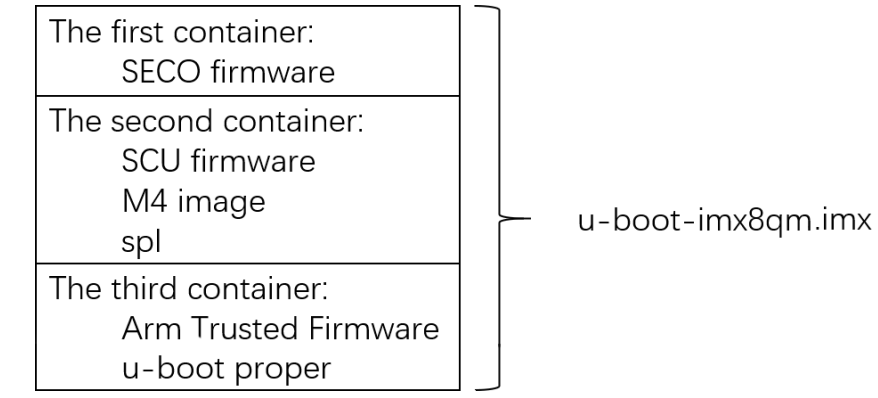


图4. 单引导加载程序情况下，启用SPL时的布局结构

对于双引导加载程序情况，以Android镜像的`spl-imx8qm-trusty-dual.bin`和`bootloader-imx8qm-trusty-dual.img`为例。第一个容器和第二个容器合并为一个镜像，第三个容器组合为另一个镜像。下图所示为布局结构。

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

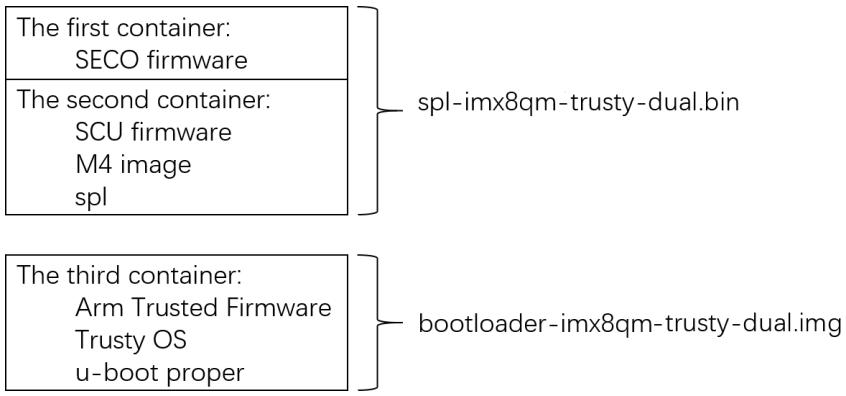


图5. 双引导加载程序情况下，启用SPL时的布局结构

与i.MX 8Quad一样，i.MX 8ULP和i.MX 95上的普通U-Boot镜像也有3个容器。单引导加载程序和双引导加载程序的布局相似。

要对在构建Android镜像的过程中构造的容器进行签名，请执行以下步骤：

1. 从恩智浦官网下载代码签名工具（CST）。使用以下命令解压缩工具包：

```
$ tar zxvf cst-3.1.0.tgz
```

2. 生成AHAB PKI树。解压缩工具包后，进入`${CST}/keys/`目录，并执行以下命令：

```
$ ./ahab_pki_tree.sh
```

然后根据此脚本的输出输入一些参数。示例如下：

```
Do you want to use an existing CA key (y/n)?: n
Do you want to use Elliptic Curve Cryptography (y/n)?: y
Enter length for elliptic curve to be used for PKI tree:
Possible values p256, p384, p521: p384
Enter the digest algorithm to use: sha384
Enter PKI tree duration (years): 5
Do you want the SRK certificates to have the CA flag set? (y/n)?: n
```

成功执行上述命令后，私钥位于`keys/`目录下，公钥证书位于`crts/`目录下。

3. 生成AHAB SRK表。

进入`${CST}/crts/`目录，并执行以下命令：

```
$ ../linux64/bin/srktool -a -s sha384 -t SRK_1_2_3_4_table.bin \
-e SRK_1_2_3_4_fuse.bin -f 1 -c \
SRK1_sha384_secp384r1_v3_usr_crt.pem,\
SRK2_sha384_secp384r1_v3_usr_crt.pem,\
SRK3_sha384_secp384r1_v3_usr_crt.pem,\
SRK4_sha384_secp384r1_v3_usr_crt.pem
```

此外，对于像i.MX 8ULP和i.MX 95这样只有8字SRK熔丝的平台，请使用`SHA256`从`SRK_1_2_3_4_table.bin`重新生成`SRK_1_2_3_4_fuse.bin`：

```
$ openssl dgst -binary -sha256 SRK_1_2_3_4_table.bin > SRK_1_2_3_4_fuse.bin
```

成功执行该命令后，SRK表及其SHA512或SHA256值将生成并分别保存在`${CST}/crts/`下的两个文件中。

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

SRK表在对容器进行签名的过程中被嵌入到容器中。因此，在启动期间，它可以用于验证签名。如果签名通过身份验证，则可以信任固件和软件镜像的哈希值来验证相应的固件和软件。SRK表的哈希值将被烧录到OTP eFuse硬件中，作为“安全根密钥”。它用于验证嵌入在容器中的SRK表。

在${CST}/keys/和${CST}/crts/中生成的文件非常重要。如果SRK哈希值被烧录到芯片中，然后芯片从开启状态转变为关闭状态，那么电路板只能通过使用这些文件签名的镜像启动。

建议为不同的芯片系列使用专用的CST包副本，因为此流程将在同一目录中生成同名文件。

4. 构建Android镜像以构造要签名的容器。

   要使用AHAB在SPL中验证镜像，请在U-Boot代码的相应defconfig文件中启用CONFIG_AHAB_BOOT配置。默认情况下不启用这些配置。以imx8qm_mek_android_trusty_dual_defconfig为例：

```
 diff --git a/configs/imx8qm_mek_android_trusty_dual_defconfig b/configs/
imx8qm_mek_android_trusty_dual_defconfig
 index 7a8a7a3c1c5..d441bded80d 100644
 --- a/configs/imx8qm_mek_android_trusty_dual_defconfig
 +++ b/configs/imx8qm_mek_android_trusty_dual_defconfig
 @@ -214,3 +214,4 @@ CONFIG_ATTESTATION_ID_MANUFACTURER="nxp"
  CONFIG_ATTESTATION_ID_MODEL="MEK-MX8Q"
  CONFIG_SHA256=y
  CONFIG_DUAL_BOOTLOADER=y
 +CONFIG_AHAB_BOOT=y
```

   mkimage_imx8是用于构造容器的工具。它在构造容器时在标准输出中输出容器的布局信息。构建Android镜像时，保存构建系统的日志信息。例如，执行以下命令：

```
$ ./imx-make.sh -j12 2>&1 | tee make_android.txt
```

   在构建过程中，构建系统输出信息保存在make_android.txt中。上述3个示例布局在对镜像签名时可以分为两类：由mkimage_imx8直接生成的镜像文件和并非由mkimage_imx8直接生成的镜像文件。由于mkimage_imx8在其生成的文件中输出容器布局信息，如果镜像文件是由mkimage_imx8直接生成的，则可以直接使用布局信息参数。如果最终镜像是由mkimage_imx8生成的中间文件组装而成的，则布局信息参数在使用前需要正确处理。

   如果启用了SPL，像u-boot-imx8qm.imx这样的单引导加载程序镜像将和由mkimage_imx8生成的两个文件组装在一起。事实上，在双引导加载程序情况下，这两个文件是分开的，而在单引导加载程序情况下，一个文件将以1KB对齐边界附加到另一个文件之后。

5. 获取文件中容器的布局信息。

   对容器进行签名时需要布局信息。具体而言，这是文件中容器和容器签名块的偏移信息。代码签名工具使用这些偏移值在文件中定位容器。布局信息可以在刚刚生成的make_android.txt中找到。根据以下原则，可以轻松地在文件中找到要签名的容器的偏移值。

   前面提到的带有待签名容器的生成文件是基于不同的U-Boot defconfig文件构建的。在${MY_ANDROID}/device/nxp/${PLATFORM}/${BOARD}/BoardConfig.mk中可以找到具有不同U-Boot defconfig文件的各种U-Boot编译目标。以i.MX 8QuadMax MEK和i.MX 8QuadXPlus MEK为例，U-Boot配置如下，并附有一些注释。

```
# bootloader-imx8qm.img and spl-imx8qm.bin
TARGET_BOOTLOADER_CONFIG := imx8qm:imx8qm_mek_androidauto_trusty_defconfig
# bootloader-imx8qm-secure-unlock.img and spl-imx8qm-secure-unlock.bin
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

```
TARGET_BOOTLOADER_CONFIG += imx8qm-secure-
unlock:imx8qm_mek_androidauto_trusty_secure_unlock_defconfig
# bootloader-imx8qxp.img and spl-imx8qxp.bin
TARGET_BOOTLOADER_CONFIG += imx8qxp:imx8qxp_mek_androidauto_trusty_defconfig
# u-boot-imx8qxp-c0.imx
TARGET_BOOTLOADER_CONFIG += imx8qxp-
c0:imx8qxp_mek_androidauto_trusty_defconfig
# bootloader-imx8qxp-secure-unlock.img and spl-imx8qxp-secure-unlock.bin
TARGET_BOOTLOADER_CONFIG += imx8qxp-secure-
unlock:imx8qxp_mek_androidauto_trusty_secure_unlock_defconfig
# u-boot-imx8qm.imx
TARGET_BOOTLOADER_CONFIG := imx8qm:imx8qm_mek_android_defconfig
# u-boot-imx8qxp.imx
TARGET_BOOTLOADER_CONFIG += imx8qxp:imx8qxp_mek_android_defconfig
# bootloader-imx8qm-dual.img and spl-imx8qm-dual.bin
TARGET_BOOTLOADER_CONFIG += imx8qm-dual:imx8qm_mek_android_dual_defconfig
# bootloader-imx8qxp-dual.img and spl-imx8qxp-dual.bin
TARGET_BOOTLOADER_CONFIG += imx8qxp-dual:imx8qxp_mek_android_dual_defconfig
# u-boot-imx8qm-hdmi.imx
TARGET_BOOTLOADER_CONFIG += imx8qm-hdmi:imx8qm_mek_android_hdmi_defconfig
# u-boot-imx8qxp-c0.imx
TARGET_BOOTLOADER_CONFIG += imx8qxp-c0:imx8qxp_mek_android_defconfig
# u-boot-imx8qm-md.imx
TARGET_BOOTLOADER_CONFIG += imx8qm-md:imx8qm_mek_android_hdmi_defconfig
# bootloader-imx8qxp-c0-dual.img and spl-imx8qxp-c0-dual.bin
TARGET_BOOTLOADER_CONFIG += imx8qxp-c0-
dual:imx8qxp_mek_android_dual_defconfig
# u-boot-imx8dx.imx
TARGET_BOOTLOADER_CONFIG += imx8dx:imx8dx_mek_android_defconfig
# u-boot-imx8dx-mek-uuu.imx
TARGET_BOOTLOADER_CONFIG += imx8dx-mek-uuu:imx8dx_mek_android_uuu_defconfig
# bootloader-imx8qm-trusty-dual.img and spl-imx8qm-trusty-dual.bin
TARGET_BOOTLOADER_CONFIG += imx8qm-trusty-
dual:imx8qm_mek_android_trusty_dual_defconfig
# bootloader-imx8qxp-trusty-dual.img and spl-imx8qm-trusty-dual.bin
TARGET_BOOTLOADER_CONFIG += imx8qxp-trusty-
dual:imx8qxp_mek_android_trusty_dual_defconfig
# bootloader-imx8qm-trusty-secure-unlock-dual.img and spl-imx8qm-trusty-
secure-unlock-dual.bin
TARGET_BOOTLOADER_CONFIG += imx8qm-trusty-secure-unlock-
dual:imx8qm_mek_android_trusty_secure_unlock_dual_defconfig
# bootloader-imx8qxp-trusty-c0-dual.img and spl-imx8qxp-trusty-c0-dual.bin
TARGET_BOOTLOADER_CONFIG += imx8qxp-trusty-c0-
dual:imx8qxp_mek_android_trusty_dual_defconfig
# bootloader-imx8qxp-trusty-secure-unlock-dual.img and spl-imx8qxp-trusty-
secure-unlock-dual.bin
TARGET_BOOTLOADER_CONFIG += imx8qxp-trusty-secure-unlock-
dual:imx8qxp_mek_android_trusty_secure_unlock_dual_defconfig
# u-boot-imx8qm-mek-uuu.imx
TARGET_BOOTLOADER_CONFIG += imx8qm-mek-uuu:imx8qm_mek_android_uuu_defconfig
# u-boot-imx8qxp-mek-uuu.imx
TARGET_BOOTLOADER_CONFIG += imx8qxp-mek-uuu:imx8qxp_mek_android_uuu_defconfig
```

可以使用defconfig文件名来定位容器的布局信息。在刚刚生成的`make_android.txt`中搜索defconfig文件名。一行提示该defconfig文件的构建流程已经完成。容器和容器签名块的偏移值可以在该行之前的几行中找到。

在日志文件中搜索容器布局信息。对于要签名的镜像由`mkimage_imx8`直接生成的情况，带有`imx8qm_mek_android_uuu_defconfig`的defconfig文件的`u-boot-imx8qm-mek-uuu.imx`的容器布局信息可以在日志文件中找到，如下所示。

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

不相关的行被省略并用省略号表示。

```
CST: CONTAINER 0 offset: 0x400
CST: CONTAINER 0: Signature Block: offset is at 0x590
...
==================== Finish building imx8qm-mek-
uuu:imx8qm_mek_android_uuu_defconfig ====================
```

日志文件包括带有imx8qm_mek_android_trusty_dual_defconfig的defconfig文件的bootloader-imx8qm-trusty-dual.img和spl-imx8qm-trusty-dual.bin的容器布局信息，如下所示。前两行的布局信息适用于bootloader-imx8qm-trusty-dual.img，接下来的两行适用于spl-imx8qm-trusty-dual.bin。

```
CST: CONTAINER 0 offset: 0x0
CST: CONTAINER 0: Signature Block: offset is at 0x190
...
CST: CONTAINER 0 offset: 0x400
CST: CONTAINER 0: Signature Block: offset is at 0x610
...
==================== Finish building
 imx8qm:imx8qm_mek_android_trusty_dual_defconfig ====================
```

对于要签名的镜像不是由mkimage_imx8直接生成的情况，以带有imx8qm_mek_android_defconfig的defconfig文件的u-boot-imx8qm.imx为例。以下信息应能够在日志文件中检索到。前两行用于中间文件u-boot-atf-container.img。此文件是包含Arm Trusted Firmware和U-Boot的第三个容器。接下来的两行布局信息用于另一个中间文件，该文件包含第二个容器。此中间文件被填充到1KB对齐边界，然后u-boot-atf-container.img被附加到此文件。因此，后两行可以直接使用，而前两行需要加上偏移值。为388KB，十六进制为0x61000，然后最后的u-boot-imx8qm.imx中第三个容器的偏移值为0x61000。它在文件中的签名偏移值为0x61000+0x190=0x61190。

```
CST: CONTAINER 0 offset: 0x0
CST: CONTAINER 0: Signature Block: offset is at 0x190
...
CST: CONTAINER 0 offset: 0x400
CST: CONTAINER 0: Signature Block: offset is at 0x610
...
append u-boot-atf-container.img at 388 KB
...
==================== Finish building imx8qm:imx8qm_mek_android_defconfig
 ====================
```

下表列出了容器和容器签名的偏移值汇总：

表2. 容器偏移信息

包含待签名容器的文件	文件中的容器偏移值	容器签名块的偏移值
u-boot-imx8qm-mek-uuu.imx	0x400	0x590
bootloader-imx8qm-trusty-dual.img	0x0	0x190
spl-imx8qm-trusty-dual.bin	0x400	0x610
u-boot-imx8qm.imx	0x400 0x61000	0x610 0x61190

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

同样的规则也适用于其它目标，唯一的例外是i.MX 8ULP和i.MX 95的UUU U-Boot目标启用了SPL，因此目标配置imx8ulp_evk_android_uuu_defconfig和imx95_evk_android_uuu_defconfig应遵循与u-boot-imx8qm.imx相同的规则。

6. 为镜像文件签名。
将要签名的文件复制到代码签名工具（CST）目录中的inux64/bin/目录中，使用名为cst的二进制文件对这些文件进行签名。执行此cst时，需要将一个CSF描述文件作为输入文件。CSF示例位于${MY_ANDROID}/vendor/nxp-opensource/uboot-imx/doc/imx/ahab/csf_examples/目录中。cst_uboot_atf.txt被复制到代码签名工具（CST）的linux64/bin/目录中。
根据要签名的镜像对刚刚复制的cst_uboot_atf.txt进行一些更改。对于要签名的镜像由mkimage_imx8直接生成的情况，以u-boot-imx8qm-mek-uuu.imx为例，做如下修改：

```
@@ -4,9 +4,9 @@ Version = 1.0
[Install SRK]
# SRK table generated by srktool
-File = "../crts/SRK_1_2_3_4_table.bin"
+File = "../../crts/SRK_1_2_3_4_table.bin"
# Public key certificate in PEM format on this example only using SRK key
-Source = "../crts/SRK1_sha384_secp384r1_v3_usr_crt.pem"
+Source = "../../crts/SRK1_sha384_secp384r1_v3_usr_crt.pem"
# Index of the public key certificate within the SRK table (0 .. 3)
Source index = 0
# Type of SRK set (NXP or OEM)
@@ -16,6 +16,6 @@ Revocations = 0x0
[Authenticate Data]
# Binary to be signed generated by mkimage
-File = "u-boot-atf-container.img"
+File = "u-boot-imx8qm-mek-uuu.imx"
# Offsets = Container header Signature block (printed out by mkimage)
-Offsets = 0x0 0x110
+Offsets = 0x400 0x590
```

然后执行以下命令：

```
$ ./cst -i cst_uboot_atf.txt -o signed-u-boot-imx8qm-mek-uuu.imx
```

在成功执行上述命令后，生成signed-u-boot-imx8qm-mek-uuu.imx。将其复制回输出目录，并像以前一样更改其名称，因为uuu_imx_android_flash脚本根据镜像的名称来烧写镜像。
根据对u-boot-imx8qm-mek-uuu.imx签名的描述对所有其它由mkimage_imx8直接生成的镜像进行签名，例如bootloader-imx8qm-trusty-dual.img、spl-imx8qm-trusty- dual.bin等。并非由mkimage_imx8直接生成的镜像的签名流程略有不同，它需要签名两次。以u-boot-imx8qm.imx为例，对原始cst_uboot_atf.txt进行以下更改：

```
@@ -4,9 +4,9 @@ Version = 1.0
[Install SRK]
# SRK table generated by srktool
-File = "../crts/SRK_1_2_3_4_table.bin"
+File = "../../crts/SRK_1_2_3_4_table.bin"
# Public key certificate in PEM format on this example only using SRK key
-Source = "../crts/SRK1_sha384_secp384r1_v3_usr_crt.pem"
+Source = "../../crts/SRK1_sha384_secp384r1_v3_usr_crt.pem"
# Index of the public key certificate within the SRK table (0 .. 3)
Source index = 0
# Type of SRK set (NXP or OEM)
@@ -16,6 +16,6 @@ Revocations = 0x0
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

```
[Authenticate Data]
# Binary to be signed generated by mkimage
-File = "u-boot-atf-container.img"
+File = "u-boot-imx8qm.imx"
# Offsets = Container header Signature block (printed out by mkimage)
-Offsets = 0x0 0x110
+Offsets = 0x400 0x610
```

然后执行以下命令：

```
$ ./cst -i cst_uboot_atf.txt -o first_signed_u-boot-imx8qm.imx
Apply the following change on the "cst_uboot_atf.txt" as below to do the
 second sign
@@ -16,6 +16,6 @@ Revocations = 0x0
[Authenticate Data]
# Binary to be signed generated by mkimage
-File = "u-boot-imx8qm.imx"
+File = "first_signed_u-boot-imx8qm.imx"
# Offsets = Container header  Signature block (printed out by mkimage)
-Offsets    = 0x400 0x610
+Offsets    = 0x61000 0x61190
```

并执行以下命令：

```
$ ./cst -i cst_uboot_atf.txt -o second_signed_u-boot-imx8qm.imx
```

在成功执行上述命令后，生成`second_signed_u-boot-imx8qm.imx`，将其复制回输出目录，并像以前一样更改其名称。所有并非由`mkimage_imx8`直接生成的镜像都可以使用上述方法签名，这意味着启用了SPL的单引导加载程序镜像可以使用此方法签名。

现在镜像已签名。在使用签名的镜像启动时，嵌入在镜像文件中的SRK表用于对签名进行验证。对嵌入的SRK表的验证则基于其哈希值进行。哈希值在i.MX芯片上被烧写到OTP eFuse中，因此不会被其它数据篡改。执行以下步骤来烧录SRK哈希值。

7. 转储SRK哈希值。

   在代码签名工具（CST）中将目录更改为`crts/`。对于i.MX 8Quad平台，执行以下命令以转储SRK哈希值：

```
$ od -t x4 SRK_1_2_3_4_fuse.bin
0000000 d436cc46 8ecccda9 b89e1601 5fada3db
0000020 d454114a b6cd51f4 77384870 c50ee4b2
0000040 a27e5132 eba887cf 592c1e2b bb501799
0000060 ee702e07 cf8ce73e fb55e2d5 eba6bbd2
```

   在i.MX 8ULP和i.MX 95上，SRK哈希使用SHA-256并转储8字熔丝：

```
$ od -t x4 SRK_1_2_3_4_fuse.bin
0000000 db2959f2 90dfc39c 53394566 e0b75829
0000020 85e6f3b1 af00983d e5e804fe 7a451024
```

8. 使用U-Boot fuse命令将哈希值烧录到芯片中。

   由于从Android Auto镜像的U-Boot中移除了fuse命令以缩短启动时间，需要使用UUU将UUU使用的U-Boot加载到RAM中，然后使用fuse命令。

   将电路板更改为串行下载模式，并执行以下命令将U-Boot下载到RAM中。然后进入fastboot模式。

   对于i.MX 8QuadMax，如下所示：

```
$ sudo ./uuu_imx_android_flash.sh -f imx8qm -i
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

```
$ sudo ./uuu_imx_android_flash.sh -f imx8qxp -i
```

对于i.MX 8QuadXPlus，如下所示：

```
$ sudo ./uuu_imx_android_flash.sh -f imx8ulp -i
```

对于i.MX 8ULP，如下所示：

```
$ $ sudo ./uuu_imx_android_flash.sh -f imx95 -i
```

对于i.MX 95，如下所示：

执行上述命令后，UUU在当前工作目录下使用的U-Boot被加载到板载RAM中，并进入fastboot模式。

在U-Boot控制台上，显示U-Boot处于fastboot模式。按“CTRL+C”退出fastboot模式，进入U-Boot命令模式。

对于i.MX 8QuadMax，在U-Boot控制台上执行以下命令：

```
=> fuse prog 0 722 0xd436cc46
=> fuse prog 0 723 0x8ecccda9
=> fuse prog 0 724 0xb89e1601
=> fuse prog 0 725 0x5fada3db
=> fuse prog 0 726 0xd454114a
=> fuse prog 0 727 0xb6cd51f4
=> fuse prog 0 728 0x77384870
=> fuse prog 0 729 0xc50ee4b2
=> fuse prog 0 730 0xa27e5132
=> fuse prog 0 731 0xeba887cf
=> fuse prog 0 732 0x592c1e2b
=> fuse prog 0 733 0xbb501799
=> fuse prog 0 734 0xee702e07
=> fuse prog 0 735 0xcf8ce73e
=> fuse prog 0 736 0xfb55e2d5
=> fuse prog 0 737 0xeba6bbd2
```

对于i.MX 8QuadXPlus，在U-Boot控制台上执行以下命令：

```
=> fuse prog 0 730 0xd436cc46
=> fuse prog 0 731 0x8ecccda9
=> fuse prog 0 732 0xb89e1601
=> fuse prog 0 733 0x5fada3db
=> fuse prog 0 734 0xd454114a
=> fuse prog 0 735 0xb6cd51f4
=> fuse prog 0 736 0x77384870
=> fuse prog 0 737 0xc50ee4b2
=> fuse prog 0 738 0xa27e5132
=> fuse prog 0 739 0xeba887cf
=> fuse prog 0 740 0x592c1e2b
=> fuse prog 0 741 0xbb501799
=> fuse prog 0 742 0xee702e07
=> fuse prog 0 743 0xcf8ce73e
=> fuse prog 0 744 0xfb55e2d5
=> fuse prog 0 745 0xeba6bbd2
```

对于i.MX 8ULP，在U-Boot控制台上执行以下命令：

```
=> fuse prog 15 0 0xdb2959f2
=> fuse prog 15 1 0x90dfc39c
=> fuse prog 15 2 0x53394566
=> fuse prog 15 3 0xe0b75829
=> fuse prog 15 4 0x85e6f3b1
=> fuse prog 15 5 0xaf00983d
=> fuse prog 15 6 0xe5e804fe
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

```
=> fuse prog 15 7 0x7a451024
```

对于i.MX 95，在U-Boot控制台上执行以下命令：

```
=> fuse prog 16 0 0xdb2959f2
=> fuse prog 16 1 0x90dfc39c
=> fuse prog 16 2 0x53394566
=> fuse prog 16 3 0xe0b75829
=> fuse prog 16 4 0x85e6f3b1
=> fuse prog 16 5 0xaf00983d
=> fuse prog 16 6 0xe5e804fe
=> fuse prog 16 7 0x7a451024
```

现在，镜像已签名，SRK哈希值已被烧录。可以将镜像烧写到电路板上。有关如何烧写i.MX Android镜像，请参阅《Android版本说明》（RN00201）。

芯片现在处于“开启”阶段，验证失败不会阻止启动流程。为了确保SRK哈希值被正确烧录，并且对镜像进行正确签名，请在启动期间查看SECO事件。在U-Boot的defconfig文件中启用`CONFIG_AHAB_BOOT`后，使用U-Boot命令查看SECO事件。在对镜像进行签名并对SRK哈希值进行烧录后，启动电路板进入U-Boot命令模式。在U-Boot控制台上执行以下命令：

```
=> ahab_status
```

如果上述命令输出SECO事件，则使用以下代码检查它是否与AHAB验证有关。

```
0x0087EE00 = The container image is not signed.
0x0087FA00 = The container image was signed with wrong key that is not
 matching the OTP SRK hashes.
```

例如，如果已经对SRK哈希值进行了烧写，但镜像未签名，那么在执行`ahab_status`后，控制台上会显示以下提示：

```
=> ahab_status
Lifecycle: 0x0020, NXP closed
SECO Event[0] = 0x0087EE00
CMD = AHAB_AUTH_CONTAINER_REQ (0x87)
IND = AHAB_NO_AUTHENTICATION_IND (0xEE)
SECO Event[1] = 0x0087EE00
CMD = AHAB_AUTH_CONTAINER_REQ (0x87)
IND = AHAB_NO_AUTHENTICATION_IND (0xEE)
```

在确认SRK哈希值已正确烧录且签名的镜像不会导致AHAB相关SECO事件后，在U-Boot控制台上执行以下命令，以关闭芯片：

```
=> ahab_close
```

***注：**此关闭操作对芯片是**不可逆的**，如果AHAB验证失败，关闭的芯片将无法启动。*

***注：**此外，需要上电复位才能使上述命令生效。*

3.1.2 使用HABv4进行安全启动

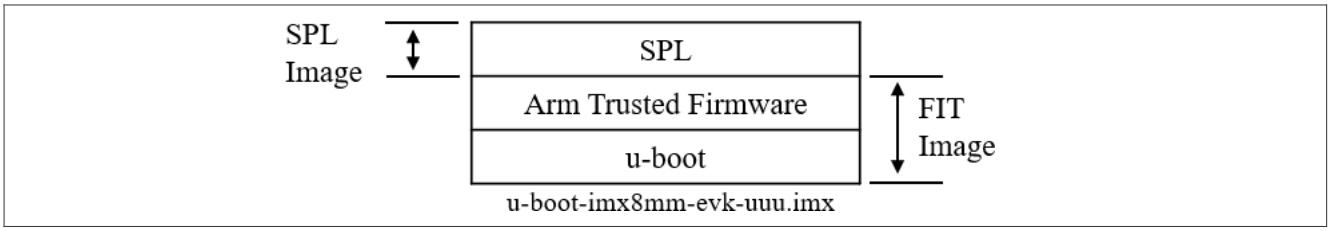
HABv4基于镜像向量表（IVT）和扁平镜像树（FIT）验证镜像。启动镜像格式的详细信息可以在相关芯片的参考手册中找到。根据参考手册，下面描述的签名流程将代码签名工具生成的命令序列文件（CSF）嵌入到最终镜像中。CSF用于在启动期间验证镜像。

有3种布局。下面以i.MX 8M Mini为例介绍镜像的不同布局。

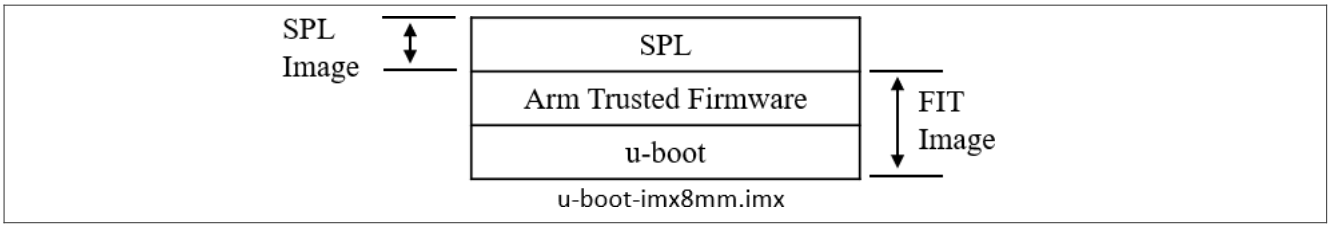
本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

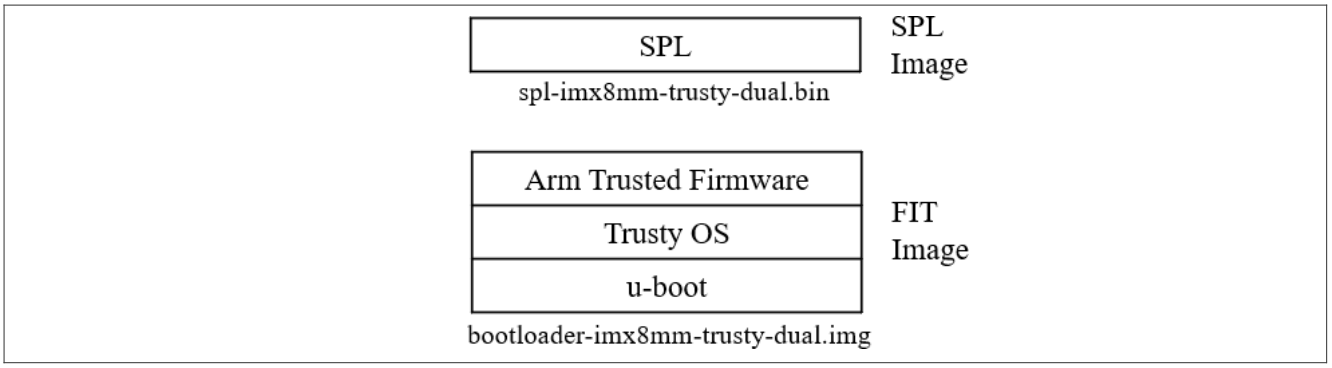
i.MX 8M平台的UUU U-Boot镜像由SPL和FIT构造。FIT结构包含Arm Trusted Firmware和U-Boot本身。它是在构建时构造的，并附加到SPL镜像的末尾。以`u-boot-imx8mm-evk-uuu.imx`为例，其高级布局结构如下图所示。



对于单引导加载程序的情况，i.MX 8M的U-Boot镜像在结构上与UUU U-Boot镜像没有差别，只是功能配置有所不同。以`u-boot-imx8mm.imx`为例，下图展示了其布局结构。



对于双引导加载程序的情况，以`spl-imx8mm-trusty-dual.bin`和`bootloader-imx8mm-trusty-dual.img`为例。这种情况下，U-Boot被拆分为SPL和FIT镜像。下图所示为其布局结构。



要为i.MX 8M芯片的镜像签名，请执行以下章节中的步骤。

3.1.2.1 获取CST工具和密钥配置

1. 从恩智浦官网下载代码签名工具（CST）。
   使用以下命令解压缩工具包：

   ```
   $ tar zxvf cst-3.1.0.tgz
   ```

2. 生成HABv4 PKI树。
   解压缩工具包后，进入`${CST}/keys/`目录，并执行以下命令：

   ```
   ./hab4_pki_tree.sh
   ```

   然后根据此脚本的输出输入一些参数。示例如下：

   ```
   Do you want to use an existing CA key (y/n)?: n
   Do you want to use Elliptic Curve Cryptography (y/n)?: n
   ```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

```
Enter key length in bits for PKI tree: 2048
Enter PKI tree duration (years): 5
How many Super Root Keys should be generated? 4
Do you want the SRK certificates to have the CA flag set? (y/n)?: y
```

3. 生成AHAB SRK表和eFuse哈希值。
   进入${CST}/crts/目录，并执行以下命令：

```
$ ../linux64/bin/srktool -h 4 -t SRK_1_2_3_4_table.bin -e \
SRK_1_2_3_4_fuse.bin -d sha256 -c \
SRK1_sha256_2048_65537_v3_ca_crt.pem,\
SRK2_sha256_2048_65537_v3_ca_crt.pem,\
SRK3_sha256_2048_65537_v3_ca_crt.pem,\
SRK4_sha256_2048_65537_v3_ca_crt.pem
```

   在成功执行上述命令后，分别生成SRK表及其SHA256值，并将其保存在${CST}/crts/目录下的两个文件中。SRK表嵌入在CSF中，因此，在启动期间，它可用于验证签名。SRK表的SHA256值被烧录到OTP eFuse硬件中，作为“安全根密钥”，用于验证CSF中的SRK表。
   在${CST}/keys/和/release/crts/中生成的文件非常重要。如果SRK HASH值被烧录到芯片中，然后芯片从“开启”状态改为“关闭”状态，那么电路板只能通过使用这些文件签名的镜像进行启动。
   如果您同时使用i.MX 8Quad和i.MX 8M芯片，建议分别为这两个系列使用此CST的两个副本，因为此流程会在同一目录中生成同名文件，同时还要防止这些文件被覆盖。
4. 转储SRK哈希值。
   在代码签名工具（CST）中将目录更改为crts/。执行以下命令以转储SRK哈希值：

```
hexdump -e '/4 "0x"' -e '/4 "%X""\n"' SRK_1_2_3_4_fuse.bin
0x20593752
0x6ACE6962
0x26E0D06C
0xFC600661
0x1240E88F
0x1209F144
0x831C8117
0x1190FD4D
```

5. 使用U-Boot fuse命令将哈希值烧录到芯片上。
   将刚刚签名的镜像烧写到电路板上，然后启动进入U-boot命令模式，并执行以下命令，以烧录SRK哈希值。这对于i.MX 8M Mini、i.MX 8M Nano、i.MX 8M Quad和i.MX 8M Plus是一样的。

```
fuse prog -y 6 0 0x20593752 0x6ACE6962 0x26E0D06C 0xFC600661
fuse prog -y 7 0 0x1240E88F 0x1209F144 0x831C8117 0x1190FD4D
```

3.1.2.2 对引导加载程序镜像进行签名

1. 构建Android镜像以生成要签名的文件。
   要支持HAB功能，请在U-Boot代码中的相应defconfig文件中启用CONFIG_IMX_HAB配置。默认情况下，这些配置不被启用。以i.MX 8M Mini EVK和i.MX 8M Nano EVK为例，文件如下：

```
imx8mm_evk_android_defconfig
imx8mm_evk_android_dual_defconfig
imx8mm_evk_android_trusty_secure_unlock_dual_defconfig
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

```
imx8mm_evk_android_trusty_dual_defconfig
imx8mm_evk_android_uuu_defconfig
```

在签名过程中需要最终U-Boot镜像的布局信息。构建Android镜像时，保存构建系统的日志信息。例如，执行以下命令：

```
$ ./imx-make.sh -j12 2>&1 | tee make_android.txt
```

在构建过程中，构建系统输出信息也保存在make_android.txt中。

2. 获取要签名的文件的布局信息。
   最终的U-Boot镜像文件是使用不同的U-Boot defconfig文件生成的。以i.MX 8M Mini为例，带有不同的U-Boot defconfig文件的各种U-Boot编译目标可以在${MY_ANDROID}/device/nxp/imx8m/evk_8mm/BoardConfig.mk中找到。如下所示，并附有一些注释。

```
TARGET_BOOTLOADER_CONFIG := imx8mm:imx8mm_evk_android_defconfig
TARGET_BOOTLOADER_CONFIG += imx8mm-dual:imx8mm_evk_android_dual_defconfig
TARGET_BOOTLOADER_CONFIG += imx8mm-trusty-secure-unlock-
dual:imx8mm_evk_android_trusty_secure_unlock_dual_defconfig
TARGET_BOOTLOADER_CONFIG += imx8mm-trusty-
dual:imx8mm_evk_android_trusty_dual_defconfig
TARGET_BOOTLOADER_CONFIG += imx8mm-evk-uuu:imx8mm_evk_android_uuu_defconfig
```

在刚刚生成的make_android.txt日志中搜索defconfig文件名，寻找提示该defconfig文件构建流程已完成的行。签名过程中需要的布局信息可以在该行前的几行中找到。布局信息需要根据是否启用了双引导加载程序进行分类。名称中带有“dual”子串的defconfig文件启用了双引导加载程序。相反，没有“dual”子串的文件则未启用双引导加载程序。
对于单引导加载程序情况，以imx8mm_evk_android_uuu_defconfig为例，其相应的输出文件为u-boot-imx8mm-evk-uuu.imx。下面列出了与此输出文件布局信息直接相关的行，不相关的行则被省略并用省略号表示。

```
Loader IMAGE:
 ...
 csf_off                0x2ea00
 spl hab block:         0x7e0fc0 0x0 0x2ea00
 ...
Second Loader IMAGE:
 sld_header_off         0x57c00
 sld_csf_off            0x58c20
 sld hab block:         0x401fadc0 0x57c00 0x1020
 fit-fdt csf_off        0x5ac20
 fit-fdt hab block:     0x401fadc0 0x57c00 0x3020
 ...
 0x40200000 0x5CC00 0x108548
 0x40308548 0x165148 0xE5E8
 0x920000 0x173730 0xA8E0
 ...
====================Finish building imx8mm-evk-
uuu:imx8mm_evk_android_uuu_defconfig ====================
```

对于双引导加载程序情况，以imx8mm_evk_android_trusty_dual_defconfig为例，其相应的输出文件为spl-imx8mm-trusty-dual.bin和bootloader-imx8mm-trusty-dual.img。下面列出了与此输出文件的布局信息直接相关的行，不相关的行则被省略并用省略号表示。

```
Loader IMAGE:
 ...
 csf_off                0x34600
 spl hab block:         0x7e0fc0 0x0 0x34600
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

```
 ...
FIT IVT IMAGE:
 fit_csf_off            0x1020
 fit hab block:         0x401fadc0 0x0 0x1020
 fit-fdt_csf_off        0x3020
 fit-fdt hab block:     0x401fadc0 0x0 0x3020
 ...
 0x40200000 0x5000 0x11CA50
 0x4031CA50 0x121A50 0xE5F0
 0x920000 0x130040 0xDB30
 0xBE000000 0x13DB70 0x191A00
 ...
==================Finish building imx8mm-trusty-
dual:imx8mm_evk_android_trusty_dual_defconfig ==================
```

检索布局信息的方法可用于其它目标。

3. 对镜像文件进行签名。

   将要签名的文件复制到代码签名工具（CST）目录中的linux64/bin/目录中，使用名为cst的二进制文件对这些文件进行签名。执行此cst时，需要将CSF描述文件作为输入文件。CSF描述文件示例位于${MY_ANDROID}/vendor/nxp-opensource/uboot-imx/doc/imx/habv4/csf_examples/mx8m/目录中。将csf_fit.txt、csf_spl.txt和csf_fit_fdt.txt复制到代码签名工具（CST）的linux64/bin/目录中，同时要签名的文件也需要复制到该目录中。单引导加载程序情况和双引导加载程序情况的签名流程不同。本节介绍了这两种情况的示例，但在实际中，不同的U-Boot目标文件应该按照以下步骤逐一签名。

   对于带有imx8mm_evk_android_uuu_defconfig defconfig文件的单引导加载程序示例，修改复制的csf_fit.txt, csf_spl.txt和csf_fit_fdt.txt，如下所示。

```
diff --git a/csf_fit.txt b/csf_fit.txt
index d9218ab..dfd0ded 100644
--- a/csf_fit.txt
+++ b/csf_fit.txt
@@ -8,12 +8,12 @@

     [Install SRK]
         # Index of the key location in the SRK table to be installed
-    File = "../crts/SRK_1_2_3_4_table.bin"
+    File = "../../crts/SRK_1_2_3_4_table.bin"
         Source index = 0

     [Install CSFK]
         # Key used to authenticate the CSF data
-    File = "../crts/CSF1_1_sha256_2048_65537_v3_usr_crt.pem"
+    File = "../../crts/CSF1_1_sha256_2048_65537_v3_usr_crt.pem"

     [Authenticate CSF]

@@ -23,14 +23,14 @@
         # Target key slot in HAB key store where key will be installed
         Target index = 2
         # Key to install
-    File = "../crts/IMG1_1_sha256_2048_65537_v3_usr_crt.pem"
+    File = "../../crts/IMG1_1_sha256_2048_65537_v3_usr_crt.pem"

     [Authenticate Data]
         # Key slot index used to authenticate the image data
         Verification index = 2
         # Authenticate Start Address, Offset, Length and file
-    Blocks = 0x401fcdc0 0x057c00 0x01020 "flash.bin", \
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

```
-              0x40200000 0x05AC00 0x9AAC8 "flash.bin", \
-              0x00910000 0x0F56C8 0x09139 "flash.bin", \
-              0xFE000000 0x0FE804 0x4D268 "flash.bin", \
-              0x4029AAC8 0x14BA6C 0x06DCF "flash.bin"
+    Blocks = 0x401fadc0 0x57c00 0x1020 "u-boot-imx8mm-evk-uuu.imx", \
+              0x40200000 0x5CC00 0x108548 "u-boot-imx8mm-evk-uuu.imx", \
+              0x40308548 0x165148 0xE5E8 "u-boot-imx8mm-evk-uuu.imx", \
+              0x920000 0x173730 0xA8E0 "u-boot-imx8mm-evk-uuu.imx"
diff --git a/csf_fit_fdt.txt b/csf_fit_fdt.txt
index dd88843dee..21498de8b9 100644
--- a/csf_fit_fdt.txt
+++ b/csf_fit_fdt.txt
@@ -8,12 +8,12 @@

     [Install SRK]
         # Index of the key location in the SRK table to be installed
-    File = "../crts/SRK_1_2_3_4_table.bin"
+    File = "../../crts/SRK_1_2_3_4_table.bin"
         Source index = 0

     [Install CSF}
         # Key used to authenticate the CSF data
-    File = "../crts/CSF1_1_sha256_2048_65537_v3_usr_crt.pem"
+    File = "../../crts/CSF1_1_sha256_2048_65537_v3_usr_crt.pem"

     [Authenticate CSF]

@@ -23,10 +23,10 @@
         # Target key slot in HAB key store where key will be installed
         Target index = 2
         # Key to install
-    File = "../crts/IMG1_1_sha256_2048_65537_v3_usr_crt.pem"
+    File = "../../crts/IMG1_1_sha256_2048_65537_v3_usr_crt.pem"

     [Authenticate Data]
         # Key slot index used to authenticate the image data
         Verification index = 2
         # Authenticate Start Address, Offset, Length and file
-    Blocks = 0x401fadc0 0x57c00 0x3020 "signed-flash.bin"
+    Blocks = 0x401fadc0 0x57c00 0x3020 "u-boot-imx8mm-evk-uuu.imx"
diff --git a/csf_spl.txt b/csf_spl.txt
index 39adf7a..80165a8 100644
--- a/csf_spl.txt
+++ b/csf_spl.txt
@@ -8,12 +8,12 @@

     [Install SRK]
         # Index of the key location in the SRK table to be installed
-    File = "../crts/SRK_1_2_3_4_table.bin"
+    File = "../../crts/SRK_1_2_3_4_table.bin"
         Source index = 0

     [Install CSFK]
         # Key used to authenticate the CSF data
-    File = "../crts/CSF1_1_sha256_2048_65537_v3_usr_crt.pem"
+    File = "../../crts/CSF1_1_sha256_2048_65537_v3_usr_crt.pem"

     [Authenticate CSF]

     [Unlock]
```


© 2024 NXP B.V. 版权所有。

```
        # Leave Job Ring and DECO master ID registers Unlocked
        Engine = CAAM
-          Features = MID
+          Features = MID, MFG

    [Install Key]
        # Key slot index used to authenticate the key to be installed
@@ -28,10 +28,10 @@
        # Target key slot in HAB key store where key will be installed
        Target index = 2
        # Key to install
-    File = "../crts/IMG1_1_sha256_2048_65537_v3_usr_crt.pem"
+    File = "../../crts/IMG1_1_sha256_2048_65537_v3_usr_crt.pem"

    [Authenticate Data]
        # Key slot index used to authenticate the image data
        Verification index = 2
        # Authenticate Start Address, Offset, Length and file
-    Blocks = 0x7e0fc0 0x1a000 0x2a600 "flash.bin"
+    Blocks = 0x7e0fc0 0x0 0x2ea00 "u-boot-imx8mm-evk-uuu.imx"
```

对于带有imx8mm_evk_android_trusty_dual_defconfig defconfig文件的双引导加载程序示例，需要修改复制的csf_fit.txt、csf_spl.txt和csf_fit_fdt.txt，如下所示。

```
diff --git a/csf_fit.txt b/csf_fit.txt
index d9218ab..dfd0ded 100644
--- a/csf_fit.txt
+++ b/csf_fit.txt
@@ -8,12 +8,12 @@

    [Install SRK]
        # Index of the key location in the SRK table to be installed
-    File = "../crts/SRK_1_2_3_4_table.bin"
+    File = "../../crts/SRK_1_2_3_4_table.bin"
        Source index = 0

    [Install CSFK]
        # Key used to authenticate the CSF data
-    File = "../crts/CSF1_1_sha256_2048_65537_v3_usr_crt.pem"
+    File = "../../crts/CSF1_1_sha256_2048_65537_v3_usr_crt.pem"

    [Authenticate CSF]

@@ -23,14 +23,14 @@
        # Target key slot in HAB key store where key will be installed
        Target index = 2
        # Key to install
-    File = "../crts/IMG1_1_sha256_2048_65537_v3_usr_crt.pem"
+    File = "../../crts/IMG1_1_sha256_2048_65537_v3_usr_crt.pem"

    [Authenticate Data]
        # Key slot index used to authenticate the image data
        Verification index = 2
        # Authenticate Start Address, Offset, Length and file
-    Blocks = 0x401fcdc0 0x057c00 0x01020 "flash.bin", \
-             0x40200000 0x05AC00 0x9AAC8 "flash.bin", \
-             0x00910000 0x0F56C8 0x09139 "flash.bin", \
-             0xFE000000 0x0FE804 0x4D268 "flash.bin", \
-             0x4029AAC8 0x14BA6C 0x06DCF "flash.bin"
+    Blocks = 0x401fadc0 0x0 0x1020 "bootloader-imx8mm-trusty-dual.img", \
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

```
+             0x40200000 0x5000 0x11CA50 "bootloader-imx8mm-trusty-dual.img",
 \
+             0x4031CA50 0x121A50 0xE5F0 "bootloader-imx8mm-trusty-dual.img",
 \
+             0x920000 0x130040 0xDB30 "bootloader-imx8mm-trusty-dual.img", \
+             0xBE000000 0x13DB70 0x191A00 "bootloader-imx8mm-trusty-
dual.img"
diff --git a/csf_fit_fdt.txt b/csf_fit_fdt.txt
index dd88843dee..21498de8b9 100644
--- a/csf_fit_fdt.txt
+++ b/csf_fit_fdt.txt
@@ -8,12 +8,12 @@

     [Install SRK]
         # Index of the key location in the SRK table to be installed
-    File = "../crts/SRK_1_2_3_4_table.bin"
+    File = "../../crts/SRK_1_2_3_4_table.bin"
         Source index = 0

     [Install CSFK]
         # Key used to authenticate the CSF data
-    File = "../crts/CSF1_1_sha256_2048_65537_v3_usr_crt.pem"
+    File = "../../crts/CSF1_1_sha256_2048_65537_v3_usr_crt.pem"

     [Authenticate CSF]

@@ -23,10 +23,10 @@
         # Target key slot in HAB key store where key will be installed
         Target index = 2
         # Key to install
-    File = "../crts/IMG1_1_sha256_2048_65537_v3_usr_crt.pem"
+    File = "../../crts/IMG1_1_sha256_2048_65537_v3_usr_crt.pem"

     [Authenticate Data]
         # Key slot index used to authenticate the image data
         Verification index = 2
         # Authenticate Start Address, Offset, Length and file
-    Blocks = 0x401fadc0 0x57c00 0x3020 "signed-flash.bin"
+    Blocks = 0x401fadc0 0x0 0x3020 "bootloader-imx8mm-trusty-dual.img"
diff --git a/csf_spl.txt b/csf_spl.txt
index 39adf7a..80165a8 100644
--- a/csf_spl.txt
+++ b/csf_spl.txt
@@ -8,12 +8,12 @@

     [Install SRK]
         # Index of the key location in the SRK table to be installed
-    File = "../crts/SRK_1_2_3_4_table.bin"
+    File = "../../crts/SRK_1_2_3_4_table.bin"
         Source index = 0

     [Install CSFK]
         # Key used to authenticate the CSF data
-    File = "../crts/CSF1_1_sha256_2048_65537_v3_usr_crt.pem"
+    File = "../../crts/CSF1_1_sha256_2048_65537_v3_usr_crt.pem"

     [Authenticate CSF]

     [Unlock]
         # Leave Job Ring and DECO master ID registers Unlocked
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

```
        Engine = CAAM
-          Features = MID
+          Features = MID, MFG

    [Install Key]
        # Key slot index used to authenticate the key to be installed
@@ -28,10 +28,10 @@
        # Target key slot in HAB key store where key will be installed
        Target index = 2
        # Key to install
-    File = "../crts/IMG1_1_sha256_2048_65537_v3_usr_crt.pem"
+    File = "../../crts/IMG1_1_sha256_2048_65537_v3_usr_crt.pem"

    [Authenticate Data]
        # Key slot index used to authenticate the image data
        Verification index = 2
        # Authenticate Start Address, Offset, Length and file
-    Blocks = 0x7e0fc0 0x1a000 0x2a600 "flash.bin"
+    Blocks = 0x7e0fc0 0x0 0x34600 "spl-imx8mm-trusty-dual.bin"
```

执行以下命令，使用CSF描述文件生成CSF。对于单引导加载程序和双引导加载程序，使用的命令是相同的。

```
$ ./cst --i=csf_spl.txt --o=csf_spl.bin
$ ./cst --i=csf_fit.txt --o=csf_fit.bin
```

执行以下命令，将CSF嵌入到U-Boot镜像中。

```
# For the single bootloader example with "imx8mm_evk_android_uuu_defconfig"
 defconfig file
$ dd if=csf_spl.bin of=u-boot-imx8mm-evk-uuu.imx seek=$((0x2ea00)) bs=1
 conv=notrunc
$ dd if=csf_fit.bin of=u-boot-imx8mm-evk-uuu.imx seek=$((0x58c20)) bs=1
 conv=notrunc

# For the dual bootloader example with
 "imx8mm_evk_android_trusty_dual_defconfig" defconfig file
$ dd if=csf_spl.bin of=spl-imx8mm-trusty-dual.bin seek=$((0x34600)) bs=1
 conv=notrunc
$ dd if=csf_fit.bin of=bootloader-imx8mm-trusty-dual.img seek=$((0x1020))
 bs=1 conv=notrunc
```

执行以下命令，使用FDT标头的CSF描述文件生成CSF。对于单引导加载程序和双引导加载程序，使用的命令是相同的。

```
./cst -i csf_fit_fdt.txt -o csf_fit_fdt.bin
```

执行以下命令，将CSF插入到U-Boot镜像中。

```
# For the single bootloader example with "imx8mm_evk_android_uuu_defconfig"
 defconfig file
$ dd if=csf_fit_fdt.bin of=u-boot-imx8mm-evk-uuu.imx seek=$((0x5ac20)) bs=1
 conv=notrunc

# For the dual bootloader example with
 "imx8mm_evk_android_trusty_dual_defconfig" defconfig file
$ dd if=csf_fit_fdt.bin of=bootloader-imx8mm-trusty-dual.img seek=$((0x3020))
 bs=1 conv=notrunc
```

在成功执行上述命令后，完成了对`u-boot-imx8mm-evk-uuu.imx`或`spl-imx8mm-trusty-dual.bin`和`bootloader-imx8mm-trusty-dual.img`的签名。然后根据签名流程的描述对其它镜像进行签名。

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

镜像现在已签名。在使用签名镜像启动时，嵌入在镜像文件中的SRK表用于验证签名。对嵌入的SRK表的验证则基于其哈希值进行。哈希值在i.MX芯片上被烧写到OTP eFuse中，因此不受其它因素的影响。

3.1.2.3 对MCU固件进行签名

i.MX 8M上的MCU固件由Cortex-A核启动。其签名将由HABv4验证。下面提供了一些脚本来对MCU固件进行自动签名。将这些脚本保存到`linux64/bin/`目录。

1. 创建CSF模板。
   保存以下CSF模板并将其重命名为`mcucsf_template`。这是MCU固件的CSF文件。

```
[Header]
    Version = 4.3
    Hash Algorithm = sha256
    Engine Configuration = 0
    Certificate Format = X509
    Signature Format = CMS
[Install SRK]
    File = "../../crts/SRK_1_2_3_4_table.bin"
    Source index = 0
[Install CSFK]
    File = "../../crts/CSF1_1_sha256_2048_65537_v3_usr_crt.pem"
[Authenticate CSF]
[Install Key]
    Verification index = 0
    Target index = 2
    File = "../../crts/IMG1_1_sha256_2048_65537_v3_usr_crt.pem"
[Authenticate Data]
    Verification index = 2
    Blocks = %ddr_addr% 0x0 %auth_len% "mcu-pad-ivt.bin"
```

2. 创建IVT生成模板。
   保存以下IVT模板并将其重命名为`genivt_template`。此脚本有助于生成IVT。

```
#! /usr/bin/perl -w
use strict;
open(my $out, '>:raw', 'ivt.bin') or die "Unable to open: $!";
print $out pack("V", 0x402000D1);    # Signature
print $out pack("V", %jump_addr%);   # Jump Location
print $out pack("V", 0x0);       # Reserved
print $out pack("V", 0x0);       # DCD pointer
print $out pack("V", 0x0);       # Boot Data
print $out pack("V", %self_ptr%);    # Self Pointer
print $out pack("V", %csf_ptr%);# CSF Pointer
print $out pack("V", 0x0);       # Reserved
close($out);
```

3. 创建脚本`habmcugen_template`。
   保存以下模板并将其重命名为`habmcugen_template`。此脚本有助于组合生成的镜像。

```
#! /bin/bash
echo "extend mcu image to %pad_len%..."
objcopy -I binary -O binary --pad-to %pad_len% --gap-fill=0x5A mcu.bin mcu-
pad.bin
echo "generate IVT"
./genIVT
echo "attach IVT..."
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

```
cat mcu-pad.bin ivt.bin > mcu-pad-ivt.bin
echo "generate csf data..."
./cst --o mcu_csf.bin --i mcu.csf
echo "merge image and csf data..."
cat mcu-pad-ivt.bin mcu_csf.bin > mcu-signed.bin
echo "extend final image to %sig_len%..."
objcopy -I binary -O binary --pad-to %sig_len% --gap-fill=0x5A mcu-signed.bin
 mcu-signed-pad.bin
```

4. 创建脚本mk_secure_mcu_imx8m。
   保存以下脚本并将其重命名为mk_secure_mcu_imx8m。此脚本调用所有上述脚本并生成最终签名的MCU镜像。

```
#! /bin/bash
let ddr_addr=$1
if [ ! -f mcu.bin ]
then
    printf "File \"mcu.bin\" does not exist.\n"
    exit 1
fi
# Calculate the size
let pad_len=$((0x20000)) # padding mcu image to 128KB
let auth_len=$((pad_len + 0x20)) # +0x20 "IVT"
let sig_len=$((auth_len + 0x2000))
let self_ptr=$((ddr_addr + pad_len))
let csf_ptr=$((ddr_addr + auth_len))
let jump_addr=$((ddr_addr))
# change value to hex string
pad_len=`printf "0x%X" ${pad_len}`
auth_len=`printf "0x%X" ${auth_len}`
sig_len=`printf "0x%X" ${sig_len}`
ddr_addr=`printf "0x%X" ${ddr_addr}`
self_ptr=`printf "0x%X" ${self_ptr}`
csf_ptr=`printf "0x%X" ${csf_ptr}`
jump_addr=`printf "0x%X" ${jump_addr}`
# Create habUimagegen
sed -e s/%pad_len%/${pad_len}/g -e s/%sig_len%/${sig_len}/g
 habmcugen_template > habmcuGen
chmod +x habmcuGen
# Create mcu.csf
sed -e s/%ddr_addr%/${ddr_addr}/g -e s/%auth_len%/${auth_len}/g
 mcucsf_template > mcu.csf
# Create genIVT
sed -e s/%jump_addr%/${jump_addr}/g -e s/%self_ptr%/${self_ptr}/g -e s/
%csf_ptr%/${csf_ptr}/g genivt_template > genIVT
chmod +x genIVT
# Generate secure boot
./habmcuGen
# OK
printf "mcu image with Signature \"mcu-signed-pad.bin\" is ready to use.\n"
```

5. 生成签名的MCU固件。
   将未签名的MCU固件复制到linux64/bin并将其重命名为mcu.bin。执行以下命令，生成最终签名的MCU固件mcu-signed-pad.bin：

```
./mk_secure_mcu_imx8m <mcu-firmware-load-address>
```

   由于Android平台上只支持TCM运行MCU固件，mcu-firmware-load-address应成为TCM中的加载地址，如0x7e0000。

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

3.1.2.4 关闭芯片

通过上述步骤，镜像被签名，并且SRK哈希值已被烧录。但由于芯片处于“开启”阶段，验证失败不会阻止启动流程。要确保SRK哈希值正确烧录并且镜像正确签名，可以在U-Boot日志中查看HAB事件。烧写已签名的镜像并重新启动到U-Boot控制台，然后执行以下命令：

```
=> hab_status
```

确保未报告HAB事件。在SPL调用HAB验证FIT部分之前，它首先验证CSF。需要确保在启动日志中没有以下与CSF相关的错误。

```
Error: CSF header command not found
```

在确认SRK哈希值被正确烧录并且签名的镜像未引发HAB事件后，在U-Boot控制台上执行以下命令，以关闭芯片：

```
=> fuse prog -y 1 3 0x2000000
```

此关闭操作对芯片是**不可逆的**，如果HABv4验证失败，关闭的芯片将无法启动。

3.1.3 使用AHAB进行加密启动

在使用AHAB解密镜像时，加密的启动镜像需要数据加密密钥（DEK）blob。AHAB能够通过调用EdgeLock安全飞地（ELE）或SECO身份验证功能来解密镜像容器。镜像必须由CST加密，并且生成的DEK必须被封装并包含在容器签名块中。要加密i.MX 8ULP、i.MX 8Quad和i.MX 95系列芯片的镜像并对其进行签名，请按照以下步骤操作。

3.1.3.1 引导加载程序的镜像布局

有两种启动镜像结构。以`imx8ulp_evk_android_trusty_defconfig`为例，其对应的输出文件为`u-boot-imx8ulp-trusty.imx`。下图所示为输出文件的结构。

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

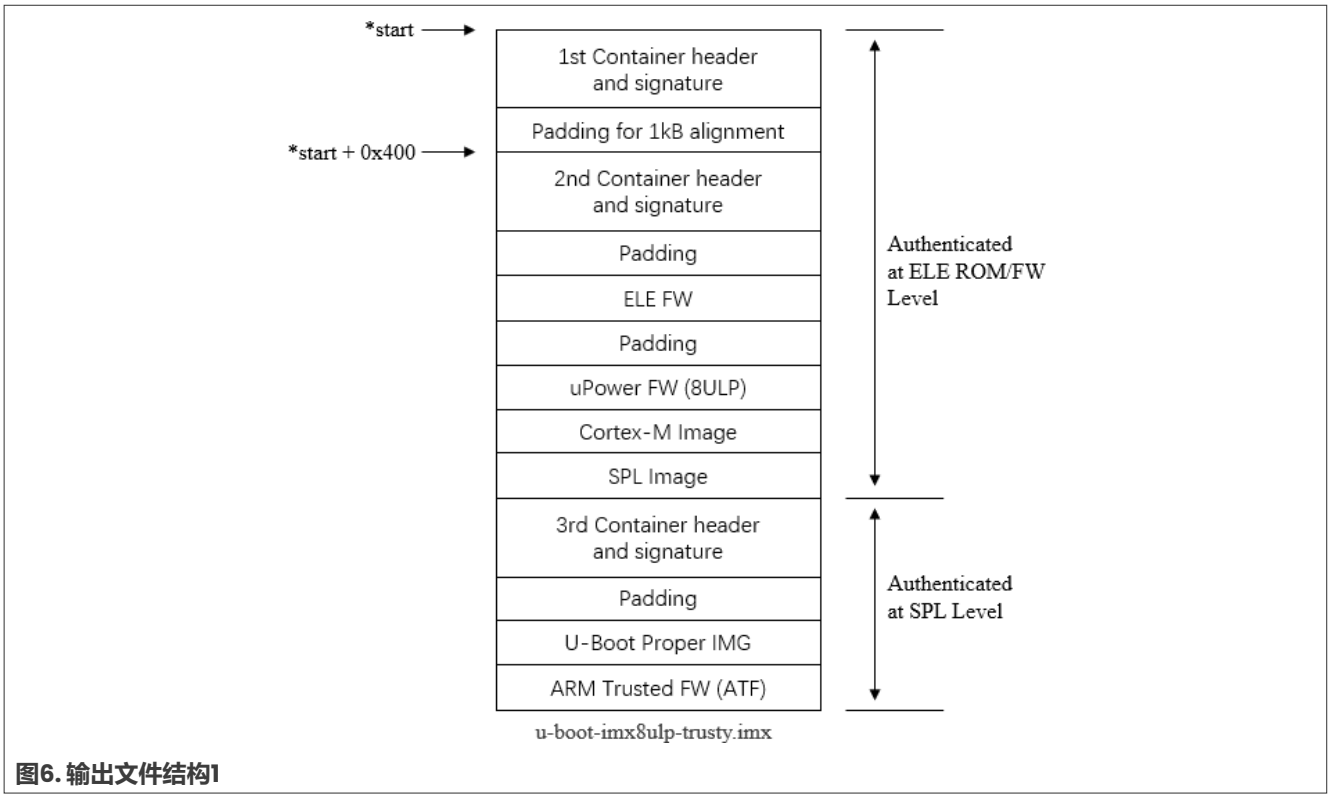


图6. 输出文件结构1

以`imx8ulp_evk_android_trusty_dual_defconfig`为例，其相应的输出文件为`spl-imx8ulp-trusty-dual.bin`和`bootloader-imx8ulp-trusty-dual.img`。下图所示为输出文件的结构。

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

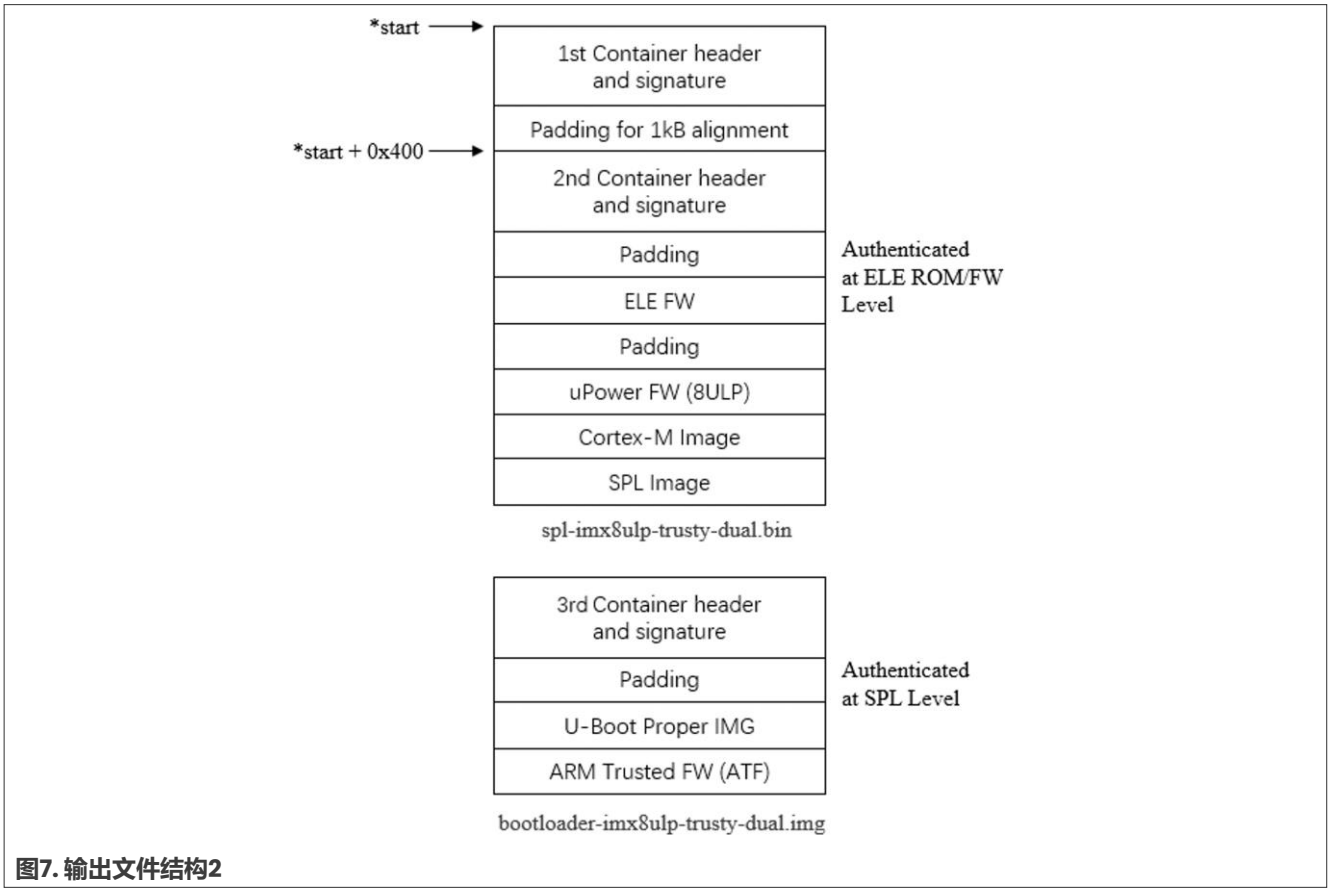


图7. 输出文件结构2

3.1.3.2 在CST中启用加密启动支持

CST 3.0.0及以后版本默认启用加密功能。如果使用更早的版本，则必须显式启用加密功能。

对于3.0.0之前的CST版本，CST后端必须重新编译。执行以下命令，启用CST中的加密支持：

```
$ sudo apt-get install libssl-dev openssl
$ cd <CST install directory>/code/back_end/src
$ gcc -o cst_encrypted -I ../hdr -L ../../../linux64/lib *.c -lfrontend -lcrypto
$ cp cst_encrypted ../../../linux64/bin/cst
```

3.1.3.3 构建Android镜像，构造容器

对于i.MX 8ULP和i.MX 95系列芯片，需要额外的U-Boot工具。启用以下功能：

- Defconfig

```
CONFIG_CMD_DEKBLOB=y
CONFIG_AHAB_BOOT=y
CONFIG_IMX_ELE_DEK_ENCAP=y
```

对于i.MX 8Quad系列芯片，启用以下功能：

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

- Defconfig

```
CONFIG_CMD_DEKBLOB=y
CONFIG_AHAB_BOOT=y
CONFIG_IMX_SECO_DEK_ENCAP=y
```

在进行这些更改后，U-Boot镜像必须重新编译。

3.1.3.4 记录构建日志，获取布局信息

在签名和加密流程中，需要最终U-Boot镜像的布局信息。构建Android镜像时，保存构建系统的日志信息。mkimage日志在加密启动过程中用于创建命令序列文件（CSF）。

- 单引导加载程序情况

  对于单引导加载程序情况，以`imx8ulp_evk_android_trusty_defconfig`为例。下面列出了与此输出文件的布局信息直接相关的行，不相关的行则被省略并用省略号表示。

```
CST: CONTAINER 0 offset: 0x0
CST: CONTAINER 0: Signature Block: offset is at 0x190
...
CST: CONTAINER 0 offset: 0x400
CST: CONTAINER 0: Signature Block: offset is at 0x590
...
append u-boot-atf-container.img at 317 KB
...
========Finish building imx8ulp-trusty:imx8ulp_evk_android_trusty_defconfig
 =========
```

- 双引导加载程序情况

  对于双引导加载程序情况，以`imx8ulp_evk_android_trusty_dual_defconfig`为例。下面列出了与此输出文件的布局信息直接相关的行，不相关的行则被省略并用省略号表示。

```
CST: CONTAINER 0 offset: 0x0
CST: CONTAINER 0: Signature Block: offset is at 0x190
...
CST: CONTAINER 0 offset: 0x400
CST: CONTAINER 0: Signature Block: offset is at 0x590
...
======Finish building imx8ulp-trusty-
dual:imx8ulp_evk_android_trusty_dual_defconfig =======
```

3.1.3.5 创建SPL镜像的CSF描述文件

将要签名的文件复制到代码签名工具（CST）目录的`linux64/bin/`目录中。使用名为CST的二进制文件对这些文件进行签名。执行此CST时，需要将CSF描述文件作为输入文件。CSF描述文件示例位于`${MY_ANDROID}/vendor/nxp-opensource/uboot-imx/doc/imx/ahab/csf_examples/`目录中。将下面列出的文件复制到CST `linux64/bin/`目录：

```
csf_enc_boot_image.txt
```

要签名的文件也需要复制到此目录中。单引导加载程序情况和双引导加载程序情况的签名流程不同。以下部分描述了这两种情况的示例，但在实际中，不同的U-Boot目标文件应该按照以下步骤逐一签名。

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

- 单引导加载程序情况

  对于带有imx8ulp_evk_android_trusty_defconfig defconfig文件的单引导加载程序示例，需按如下方式修改CSF描述文件，第一次加密U-Boot镜像并生成dek_spl.bin文件。生成加密的镜像first-u-boot-imx8ulp-trusty.imx。执行以下命令复制CSF描述文件：

```
$ cp csf_enc_boot_image.txt csf_enc_spl_image.txt
```

  然后修改刚刚复制的文件。由于第一个容器已由恩智浦签名，此CSF描述文件应用于第二个容器。偏移项应填写为第二个容器。

```
[Authenticate Data]
# Binary to be signed generated by mkimage
File = "u-boot-imx8ulp-trusty.imx"
# Offsets = Container header  Signature block (printed out by mkimage)
Offsets   = 0x400             0x590

[Install Secret Key]
Key = "dek_spl.bin"
Key Length = 128
#Key Identifier = 0x1234CAFE
Image Indexes = 0xFFFFFFFE
```

- 双引导加载程序情况

  对于带有imx8ulp_evk_android_trusty_dual_defconfig defconfig文件的双引导加载程序示例，其流程与单引导加载程序情况相同。按如下方式修改CSF描述文件，然后生成加密的SPL镜像spl-imx8ulp-trusty-dual-enc.bin。

```
[Authenticate Data]
# Binary to be signed generated by mkimage
File = "spl-imx8ulp-trusty-dual.bin"
# Offsets = Container header  Signature block (printed out by mkimage)
Offsets   = 0x400             0x590

[Install Secret Key]
Key = "dek_spl.bin"
Key Length = 128
#Key Identifier = 0x1234CAFE
Image Indexes = 0xFFFFFFFE
```

3.1.3.6 对SPL镜像进行加密和签名

使用代码签名工具对镜像进行加密。该工具生成加密镜像和一个random dek_spl.bin文件。同时生成CSF二进制文件。

- 单引导加载程序情况

```
$ ./cst -i csf_enc_spl_image.txt -o first-u-boot-imx8ulp-trusty.imx
The DEK BLOB must be inserted at offset 0x7c0 (its expected size is 72 bytes)
CSF Processed successfully and signed image available in first-u-boot-imx8ulp-trusty.imx
```

- 双引导加载程序情况

```
$ ./cst -i csf_enc_spl_image.txt -o spl-imx8ulp-trusty-dual-enc.bin
The DEK BLOB must be inserted at offset 0x7c0 (its expected size is 72 bytes)
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

```
CSF Processed successfully and signed image available in spl-imx8ulp-trusty-
dual-enc.bin
```

输出日志在后续步骤中用于将DEK blob插入到签名块中。

3.1.3.7 创建引导加载程序镜像的CSF描述文件

- 单引导加载程序情况
  与SPL镜像一样，执行以下命令复制CSF描述文件：

```
$ cp csf_enc_boot_image.txt csf_enc_bl_image.txt
```

`imx-mkimage`已打印要附加的`u-boot-atf-container.img`的偏移值，如下所示：

```
imx-mkimage output:
append u-boot-atf-container.img at 317 KB
```

附加`u-boot-atf-container.img`的偏移值为317KB，十六进制为0x4F400。第三个容器标头的偏移值已打印为0x0，签名块的偏移值打印为0x190。最终镜像中第三个容器签名块的偏移值可以按如下方式计算：

```
Container header = 0x0 + 0x4f400 = 0x4f400
Signature block = 0x190 + 0x4f400 = 0x4f590
```

然后修改刚刚复制的文件。此CSF描述应用于第三个容器，第二次加密U-Boot镜像，并生成`dek_bl.bin`文件。生成加密镜像`second-u-boot-imx8ulp-trusty.imx`。

```
[Unlock]
# Leave Job Ring and DECO master ID registers Unlocked
Features = MID, MFG

[Authenticate Data]
# Binary to be signed generated by mkimage
File = "first-u-boot-imx8ulp-trusty.imx"
# Offsets = Container header  Signature block (printed out by mkimage)
Offsets   = 0x4f400             0x4f590

[Install Secret Key]
Key = "dek_bl.bin"
Key Length = 128
#Key Identifier = 0x1234CAFE
Image Indexes = 0xFFFFFFFE
```

- 双引导加载程序情况
  在这种情况下，无需在这里计算偏移值，因为容器位于一个单独的文件中。按如下方式修改CSF描述文件。生成加密镜像`bootloader-imx8ulp-trusty-dual-enc.img`。

```
[Unlock]
# Leave Job Ring and DECO master ID registers Unlocked
Features = MID, MFG

[Authenticate Data]
# Binary to be signed generated by mkimage
File = "bootloader-imx8ulp-trusty-dual.img"
# Offsets = Container header  Signature block (printed out by mkimage)
Offsets   = 0x0             0x190

[Install Secret Key]
Key = "dek_bl.bin"
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

```
Key Length = 128
#Key Identifier = 0x1234CAFE
Image Indexes = 0xFFFFFFFE
```

3.1.3.8 对引导加载程序镜像进行加密和签名

- 单引导加载程序情况
  对于单引导加载程序情况，执行以下命令生成加密镜像和一个随机dek_blob_bl.bin文件。同时生成CSF二进制文件。

```
./cst -i csf_enc_bl_image.txt -o second-u-boot-imx8ulp-trusty.imx
    The DEK BLOB must be inserted at offset 0x4f7c0 (its expected size is 72
 bytes)
    CSF Processed successfully and signed image available in second-u-boot-
imx8ulp-trusty.imx
```

- 双引导加载程序情况
  对于双引导加载程序情况，执行以下命令：

```
./cst -i csf_enc_bl_image.txt -o bootloader-imx8ulp-trusty-dual-enc.img
    The DEK BLOB must be inserted at offset 0x3c0 (its expected size is 72
 bytes)
    CSF Processed successfully and signed image available in bootloader-
imx8ulp-trusty-dual-enc.img
```

3.1.3.9 生成DEK blob

DEK必须封装到CAAM blob中，以便包含在最终的加密二进制文件中。U-Boot提供了一个名为generate-dek-blob的fastboot命令，该命令调用ELE或SECO blob封装API。此命令仅在具有闭合配置和签名镜像的电路板上有效。参见第3.1.1节生成安全启动镜像并正确关闭电路板。以下步骤对于单引导加载程序和双引导加载程序情况是相同的。

启动电路板并进入fastboot模式。从主机PC执行以下命令：

```
fastboot stage dek_spl.bin
fastboot oem generate-dek-blob
fastboot get_staged dek_blob_spl.bin
fastboot stage dek_bl.bin
fastboot oem generate-dek-blob
fastboot get_staged dek_blob_bl.bin
```

3.1.3.10 组装加密镜像

在前面的步骤中生成的DEK blob应该插入到容器签名块中。

- 单引导加载程序情况
  以下CSF日志用于确定第二个容器中的DEK blob偏移值：

```
The DEK BLOB must be inserted at offset 0x7c0 (its expected size is 72 bytes)
CSF Processed successfully and signed image available in first-u-boot-imx8ulp-
trusty.imx
```

1. 将DEK blob插入第二个容器签名块：

```
dd if=dek_blob_spl.bin of=second-u-boot-imx8ulp-trusty.imx bs=1 seek=
$((0x7c0)) conv=notrunc
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

以下CSF日志用于确定第三个容器中的DEK blob偏移值：

```
The DEK BLOB must be inserted at offset 0x4f7c0 (its expected size is 72
 bytes)
CSF Processed successfully and signed image available in second-u-boot-
imx8ulp-trusty.imx
```

2. 将DEK blob插入第三个容器签名块中：

```
dd if=dek_blob_bl.bin of=second-u-boot-imx8ulp-trusty.imx bs=1 seek=
$((0x4f7c0)) conv=notrunc
```

- 双引导加载程序情况
  以下CSF日志用于确定SPL镜像的第二个容器中的DEK blob偏移值：

```
The DEK BLOB must be inserted at offset 0x7c0 (its expected size is 72 bytes)
CSF Processed successfully and signed image available in spl-imx8ulp-trusty-
dual-enc.bin
```

1. 将DEK blob插入第二个容器签名块：

```
dd if=dek_blob_spl.bin of=spl-imx8ulp-trusty-dual-enc.bin bs=1 seek=
$((0x7c0)) conv=notrunc
```

2. 以下CSF日志用于确定引导加载程序镜像的第一个容器中的DEK blob偏移值：

```
The DEK BLOB must be inserted at offset 0x3c0 (its expected size is 72
 bytes)
CSF Processed successfully and signed image available in bootloader-imx8ulp-
trusty-dual-enc.img
```

3. 将DEK blob插入第一个容器签名块：

```
dd if=dek_blob_bl.bin of=bootloader-imx8ulp-trusty-dual-enc.img bs=1 seek=
$((0x3c0)) conv=notrunc
```

3.1.1.2 烧写加密启动镜像

成功执行上述命令后，生成签名和加密的镜像。将这些镜像复制回输出目录，像以前一样更改它们的名称，然后将其烧写到相应的分区。如果一切操作正确，电路板将成功启动。

3.1.2 使用HABv4进行加密启动

当使用HABv4解密镜像时，加密启动镜像需要数据加密密钥（DEK）blob。DEK blob用作安全层，通过一次性可编程主密钥（OTPMK）将DEK打包和存储在芯片外，OTPMK密钥对每个芯片都是唯一的。要对i.MX 8M芯片的镜像进行加密和签名，请按照以下步骤操作。

3.1.2.1 引导加载程序的镜像布局

有两种启动镜像结构。以`imx8mm_evk_android_trusty_defconfig`为例，其对应的输出文件为`u-boot-imx8mm-trusty.im`。下图所示为输出文件的结构。

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

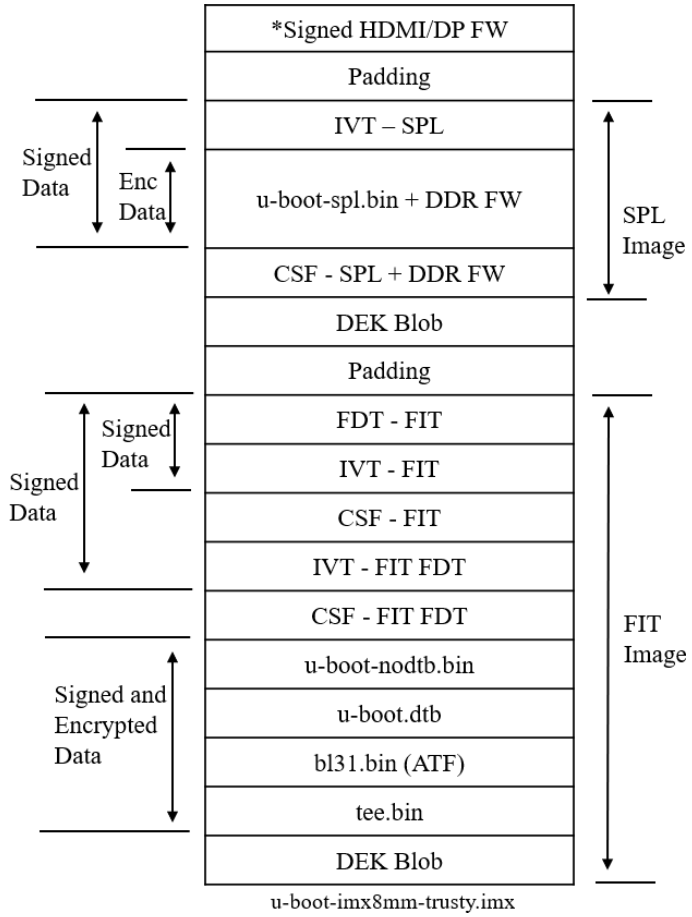


图8. 输出文件结构1

以`imx8mm_evk_android_trusty_dual_defconfig`为例，其对应的输出文件为`spl-imx8mm-trusty-dual.bin`和`bootloader-imx8mm-trusty-dual.img`。下图所示为输出文件的结构。

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

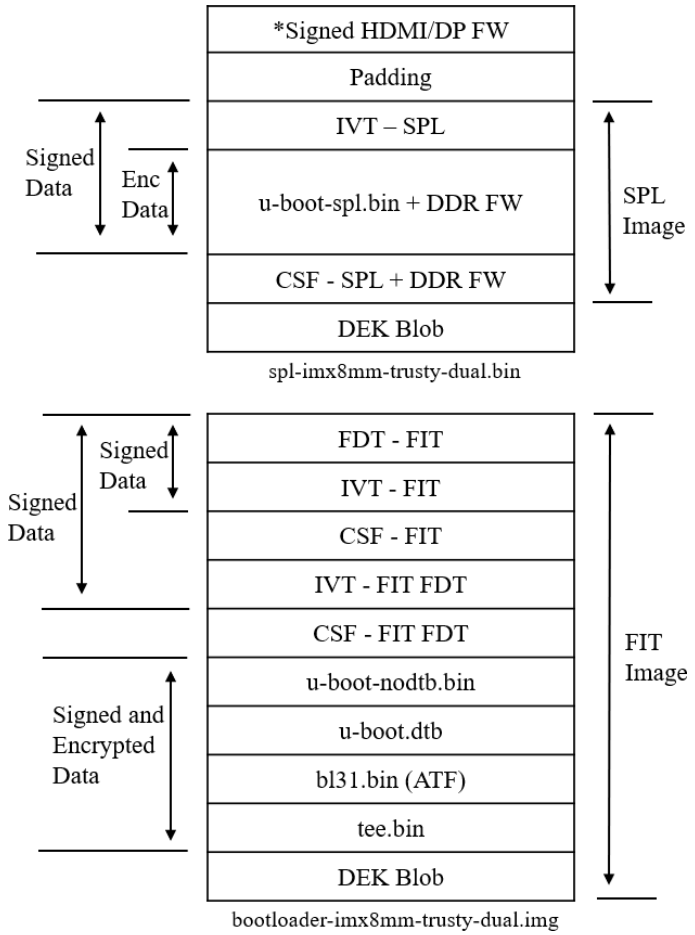


图9. 输出文件结构2

3.1.2.2 在U-Boot中启用加密启动支持

要部署加密启动镜像，需要额外的U-Boot工具。启用以下功能：

- Defconfig

```
CONFIG_IMX_HAB=y
CONFIG_CMD_PRIBLOB=y
CONFIG_CMD_DEKBLOB=y
```

3.1.2.3 在CST中启用加密启动支持

CST v3.0.0及以后的版本默认启用加密功能。如果使用更早的版本，则必须显式启用加密功能。

对于早于v3.0.0的CST版本，CST后端必须重新编译。执行以下命令，启用CST中的加密支持：

```
$ sudo apt-get install libssl-dev openssl
$ cd <CST install directory>/code/back_end/src
$ gcc -o cst_encrypted -I ../hdr -L ../../../linux64/lib *.c -lfrontend -lcrypto
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

```
$ cp cst_encrypted ../../../linux64/bin/cst
```

3.1.2.4 构建Android镜像，生成要签名的文件

如${MY_ANDROID}/vendor/nxp-opensource/uboot-imx/doc/imx/habv4/guides/mx8m_secure_boot.txt中所述，imx-mkimage工程用于打包所有镜像。

在${MY_ANDROID}/vendor/nxp-opensource/imx-mkimage/iMX8M创建一个虚拟DEK blob：

```
$ dd if=/dev/zero of=iMX8M/dek_blob_fit_dummy.bin bs=96 count=1 && sync
```

在进行这些更改后，U-Boot镜像必须重新编译。由于有了这个虚拟文件，构建脚本可以在u-boot.its中添加一个dek_blob节点，并且mkimage_imx8可以在目标文件中为dek_blob预留空间。

3.1.2.5 记录构建日志，获取布局信息

在签名和加密过程中，需要最终U-Boot镜像的布局信息。在构建Android镜像时，保存构建系统的日志信息。mkimage日志在加密启动过程中用于创建命令序列文件（CSF）。

- 单引导加载程序情况

  对于单引导加载程序情况，以imx8mm_evk_android_trusty_defconfig为例，下面列出了与此输出文件的布局信息直接相关的行，不相关的行则被省略并用省略号表示。

```
Loader IMAGE:
  header_image_off        0x0
  dcd_off                 0x0
  image_off               0x40
  csf_off                 0x33e00
  spl hab block:          0x7e0fc0 0x0 0x33e00

Second Loader IMAGE:
  sld_header_off          0x57c00
  sld_csf_off             0x58c20
  sld hab block:          0x401fadc0 0x57c00 0x1020
  fit-fdt csf_off         0x5ac20
  fit-fdt hab block:      0x401fadc0 0x57c00 0x3020
...

  0x40200000 0x5CC00 0x11C918
  0x4031C918 0x179518 0xE5E8
  0x920000 0x187B00 0xDB30
  0xBE000000 0x195630 0x191A00
...
==================Finish building imx8mm-
trusty:imx8mm_evk_android_trusty_defconfig ==================
```

- 双引导加载程序情况

  对于双引导加载程序情况，以imx8mm_evk_android_trusty_dual_defconfig为例。下面列出了与此输出文件的布局信息直接相关的行，不相关的行则被省略并用省略号表示。

```
Loader IMAGE:
    header_image_off        0x0
    dcd_off                 0x0
    image_off               0x40
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

```
    csf_off                 0x34600
    spl hab block:          0x7e0fc0 0x0 0x34600

...

FIT IVT IMAGE:
    fit_csf_off             0x1020
    fit hab block:          0x401fadc0 0x0 0x1020
    fit-fdt_csf_off         0x3020
    fit-fdt hab block:      0x401fadc0 0x0 0x3020

...

    0x40200000 0x5000 0x11CA50
    0x4031CA50 0x121A50 0xE5F0
    0x920000 0x130040 0xDB30
    0xBE000000 0x13DB70 0x191A00
...
==================Finish building imx8mm-trusty-
dual:imx8mm_evk_android_trusty_dual_defconfig ==================
```

3.1.2.6 创建SPL+DDR FW镜像的CSF描述文件

将要签名的文件复制到代码签名工具（CST）目录的`linux64/bin/`目录中。使用名为CST的二进制文件对这些文件进行签名。执行CST时需要将CSF描述文件作为输入文件。CSF描述文件示例位于`${MY_ANDROID}/vendor/nxp-opensource/uboot-imx/doc/imx/habv4/csf_examples/mx8m/`目录下。将下面列出的文件复制到CST `linux64/bin/`目录：

```
csf_spl_enc.txt
csf_spl_sign_enc.txt
csf_fit_enc.txt
csf_fit_sign_enc.txt
csf_fit_fdt.txt
```

要签名的文件也需要复制到此目录中。单引导加载程序和双引导加载程序情况的签名流程不同。以下部分介绍了这两种情况的示例，但在实际中，不同的U-Boot目标文件应该按照以下步骤逐一签名。

当前的CST实现无法同时对镜像进行加密和签名。因此，当使用HAB时，需要两个CSF文件。

- 单引导加载程序情况
  对于带有`imx8mm_evk_android_trusty_defconfig`defconfig文件的单引导加载程序示例，修改刚刚复制的`csf_spl_enc.txt`和`csf_spl_sign_enc.txt`。
  按照如下方式修改第一个文件`csf_spl_enc.txt`，对SPL和DDR FW镜像进行加密，并生成`dek_spl.bin`文件。生成加密的SPL和DDR FW镜像`u-boot-imx8mm-trusty-spl-enc.imx`。
  1. 添加MFG功能，以避免制造保护公钥（MPPUBK，由CAAM模块生成）被擦除。

     ```
     [Unlock]
       Engine = CAAM
       Features = MID, MFG
     ```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

2. 添加“Authenticate Data”（验证数据）命令，仅覆盖SPL IVT和启动数据：

```
[Authenticate Data]
  ...
  # Blocks = Authenticate_Start_Address Header_Image_Off SPL_header(fixed
 value)
  Blocks = 0x7e0fc0 0x0 0x40 "u-boot-imx8mm-trusty.imx"
```

注：*这里的所有十六进制数都可以从构建日志中获取。*
Authenticate_Start_Address是spl hab block项中的第一个十六进制数：

```
spl hab block:          0x7e0fc0                    0x0
 0x33e00
                        Authenticate_Start_Address   Header_Image_Off
 SPL_Image_length
```

3. 添加“Install Secret Key”（安装密钥）命令，以生成dek_spl.bin文件并安装blob。Blob地址取决于镜像布局，可以按如下方式计算：

```
[Install Secret Key]
  ...
  Key = "dek_spl.bin"
  # Blob Address = Authenticate_Start_Address + SPL_Image_length +
 CSF_Padding(fixed value)
  #              = 0x7e0fc0 + 0x33e00 + 0x2000 = 0x816dc0
  Blob Address = 0x816dc0
```

注：*CSF_Padding是一个固定值。这是为CSF保留的空间。*

4. 添加“Decrypt Data”（解密数据）命令，对文件进行加密。由于SPL镜像标头无法加密，请按如下方式计算该块：
起始地址=Authenticate_Start_Address+SPL_header（固定值）=0x7e0fc0 + 0x40 = 0x7e1000
偏移值=image_off = 0x40
解密大小=SPL_Image_length - SPL_header（固定值）=0x33e00 - 0x40 = 0x33dc0

```
[Decrypt Data]
  ...
   Blocks = 0x7e1000 0x40 0x33dc0 "u-boot-imx8mm-trusty-spl-enc.imx"
```

注：*SPL_header是一个固定值，由SPL标头结构的大小决定。*
按照如下方式修改第二个文件csf_spl_sign_enc.txt。此文件用于对之前生成的加密SPL镜像（u-boot-imx8mm-trusty-spl-enc.imx）进行签名。

5. 添加MFG功能，以避免MPPUBK被擦除。

```
[Unlock]
  Engine = CAAM
  Features = MID, MFG
```

6. “Authenticate Data”（验证数据）命令应覆盖整个SPL和DDR FW镜像。文件参数为加密镜像u-boot-imx8mm-trusty-spl-enc.imx：

```
[Authenticate Data]
  ...
  # Blocks = Authenticate_Start_Address Header_Image_Off SPL_Image_length
  Blocks = 0x7e0fc0 0x0 0x33e00 "u-boot-imx8mm-trusty-spl-enc.imx"
```

7. 添加“Install Secret Key”（安装密钥）命令，以生成虚拟DEK blob文件。blob地址应与csf_spl_enc.txt中使用的地址相同：

```
[Install Secret Key]
  ...
  Key = "dek_spl_dummy.bin"
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

```
  Blob Address = 0x816dc0
```

8. 添加“Decrypt Data”（解密数据）命令，对文件进行加密。由于镜像是在上述CSF中加密的，需要加密一个虚拟文件。blob地址应与csf_spl_enc.txt中使用的地址相同：

```
[Decrypt Data]
  ...
  Blocks = 0x7e1000 0x40 0x33dc0 "u-boot-imx8mm-trusty-spl-enc-dummy.imx"
```

- 双引导加载程序情况
  对于带有imx8mm_evk_android_trusty_dual_defconfig defconfig文件的双引导加载程序示例，修改刚刚复制的csf_spl_enc.txt和csf_spl_sign_enc.txt。
  按如下方式修改第一个文件csf_spl_enc.txt，以加密SPL和DDR FW镜像并生成dek_spl.bin文件。生成加密SPL和DDR FW镜像spl-imx8mm-trusty-dual-enc.bin。
  1. 添加MFG功能，以避免MPPUBK被擦除：

```
[Unlock]
  Engine = CAAM
  Features = MID, MFG
```

  2. 添加“Authenticate Data”（验证数据）命令，仅覆盖SPL IVT和启动数据：

```
[Authenticate Data]
  ...
  # Blocks = Authenticate_Start_Address header_image_off SPL_header(fixed
 value)
  Blocks = 0x7e0fc0 0x0 0x40 "spl-imx8mm-trusty-dual.bin"
```

  注：这里的所有十六进制数字都可以从构建日志获得。
  Authenticate_Start_Address是spl hab block项中的第一个十六进制数：

```
spl hab block:          0x7e0fc0                        0x0
 0x34600
                        Authenticate_Start_Address    header_image_off
 SPL_Image_length
```

  3. 添加“Install Secret Key”（安装密钥）命令，以生成dek_spl.bin文件并安装blob。blob地址取决于镜像布局，可按如下方式计算：

```
[Install Secret Key]
  ...
  Key = "dek_spl.bin"
  # Blob Address = Authenticate_Start_Address +  SPL_Image_length +
 CSF_Padding(fixed value)
  #              = 0x7e0fc0 + 0x34600 + 0x2000 = 0x8175c0
  Blob Address = 0x8175c0
```

  4. 添加“Decrypt Data”（解密数据）命令，对文件进行加密。由于SPL镜像标头无法加密，请按如下方式计算块：
     起始地址=Authenticate_Start_Address + SPL_header（固定值）= 0x7e0fc0 + 0x40 = 0x7e1000
     偏移值=image_off = 0x40
     解密大小=SPL_Image_length - SPL_header（固定值）=0x34600 - 0x40 = 0x345c0

```
[Decrypt Data]
  ...
  Blocks = 0x7e1000 0x40 0x345c0 "spl-imx8mm-trusty-dual-enc.bin"
```

     按如下方式修改第二个文件csf_spl_sign_enc.txt，对之前生成的加密SPL镜像（spl-imx8mm-trusty-dual-enc.bin）进行签名。

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

5. 添加MFG功能，以避免MPPUBK被擦除：

```
[Unlock]
  Engine = CAAM
  Features = MID, MFG
```

6. “Authenticate Data”（验证数据）命令应覆盖整个SPL和DDR FW镜像。文件参数为加密镜像`spl-imx8mm-trusty-dual-enc.bin`：

```
[Authenticate Data]
  ...
  # Blocks = Authenticate_Start_Address header_image_off SPL_Image_length
  Blocks = 0x7e0fc0 0x0 0x34600 "spl-imx8mm-trusty-dual-enc.bin"
```

7. 添加“Install Secret Key”（安装密钥）命令，以生成虚拟DEK blob文件。blob地址应与`csf_spl_enc.txt`中使用的地址相同：

```
[Install Secret Key]
  ...
  Key = "dek_spl_dummy.bin"
  Blob Address = 0x8175c0
```

8. 添加“Decrypt Data”（解密数据）命令，对文件进行加密。由于镜像是在上述CSF中加密的，需要加密一个虚拟文件。blob地址应与`csf_spl_enc.txt`中使用的地址相同：

```
[Decrypt Data]
  ...
  Blocks = 0x7e1000 0x40 0x345c0 "spl-imx8mm-trusty-dual-enc-dummy.bin"
```

3.1.2.7 对SPL+DDR FW镜像进行加密和签名

CST用于对镜像进行加密并重新生成随机DEK。在此过程中，生成两个CSF二进制文件，但最终镜像中只包含一个CSF二进制文件。

- 单引导加载程序情况
  对于单引导加载程序情况，执行以下命令，以生成带有CSF描述文件的CSF。
  – 对SPL+DDR FW镜像进行加密：

```
$ cp u-boot-imx8mm-trusty.imx u-boot-imx8mm-trusty-spl-enc.imx
$ ./cst -i csf_spl_enc.txt -o csf_spl_enc.bin
```

  – 对加密SPL+DDR FW镜像进行签名：

```
$ cp u-boot-imx8mm-trusty-spl-enc.imx u-boot-imx8mm-trusty-spl-enc-dummy.imx
$ ./cst -i csf_spl_sign_enc.txt -o csf_spl_sign_enc.bin
```

- 双引导加载程序情况
  对于双引导加载程序情况，执行以下命令，以生成带有CSF描述文件的CSF。
  – 对SPL+DDR FW镜像进行加密：

```
$ cp spl-imx8mm-trusty-dual.bin spl-imx8mm-trusty-dual-enc.bin
$ ./cst -i csf_spl_enc.txt -o csf_spl_enc.bin
```

  – 对加密SPL+DDR FW镜像进行签名：

```
$ cp spl-imx8mm-trusty-dual-enc.bin spl-imx8mm-trusty-dual-enc-dummy.bin
$ ./cst -i csf_spl_sign_enc.txt -o csf_spl_sign_enc.bin
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

3.1.2.8 创建SPL镜像的最终CSF二进制文件

由于最终镜像中只包含一个CSF二进制文件，需要将Nonce/MAC从csf_spl_enc.bin换为csf_spl_sign_enc.bin。以下步骤对于单引导加载程序和双引导加载程序情况是相同的。

1. 根据CSF中的MAC字节值计算Nonce/MAC大小：

```
Nonce/MAC size = Nonce size + MAC bytes + CSF header for Nonce/Mac = 12 + 16
 + 8 = 36 bytes
```

2. 计算CSF中的Nonce/MAC偏移值：

```
MAC offset = csf_spl_enc.bin size - Nonce/MAC size = 3980 - 36 = 3944 Bytes
```

3. 从csf_spl_enc.bin提取Nonce/MAC：

```
$ dd if=csf_spl_enc.bin of=noncemac.bin bs=1 skip=3944 count=36
```

4. 用上面提取的地址替换csf_spl_sign_enc.bin的MAC地址：

```
$ dd if=noncemac.bin of=csf_spl_sign_enc.bin bs=1 seek=3944 count=36
```

3.1.2.9 创建FIT镜像的CSF描述文件

- 单引导加载程序情况
  与SPL镜像一样，对FIT镜像进行加密和签名需要两个CSF文件。对于单引导加载程序情况，以下步骤使用前面步骤中创建的u-boot-imx8mm-trusty-spl-enc.imx镜像。修改刚刚复制的csf_fit_enc.txt和csf_fit_sign_enc.txt。
  按如下方式修改第一个文件csf_fit_enc.txt，对FIT镜像进行加密并生成dek_fit.bin文件。
  1. 修改“Authenticate Data”（验证数据）命令，使其仅覆盖FIT镜像FDT标头：

```
[Authenticate Data]
  ...
  # Blocks = Authenticate_Start_Address sld_header_off sld_csf_off -
 sld_header_off
  Blocks = 0x401fadc0 0x57c00 0x1020 "u-boot-imx8mm-trusty-spl-enc.imx"
```

  注： *这里的所有十六进制数字都可以从构建日志获得。*
  Authenticate_Start_Address是sld hab block项中的第一个十六进制数：

```
sld hab block:          0x401fadc0                  0x57c00
 0x1020
                        Authenticate_Start_Address   sld_header_off
 sld_csf_off - sld_header_off
```

  2. 添加“Install Secret Key”（安装密钥）命令，以生成dek_fit.bin文件并安装blob。blob地址是iMX8M/soc.mak文件中imx-mkimage工程中定义的固定地址：

```
DEK_BLOB_LOAD_ADDR = 0x40400000
```

  3. 修改Blob地址：

```
[Install Secret Key]
  ...
  Key = "dek_fit.bin"
  Blob Address = 0x40400000
```

  4. 添加“Decrypt Data”（解密数据）命令，对文件进行加密。

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

CST只能加密16字节对齐的镜像。由于u-boot-nodtb.bin和u-boot.dtb共同16字节对齐，我们应该将print_fit_hab中提供的前两行视为一个块。

imx-mkimage输出：

```
0x40200000 0x5CC00 0x11C918----+----Total length = 0x11c918 + 0xe5e8 =
 0x12af00
0x4031C918 0x179518 0xE5E8 ---- +
0x920000 0x187B00 0xDB30
0xBE000000 0x195630 0x191A00
```

在csf_fit_enc.txt中解密数据：

```
[Decrypt Data]
    ...
    Blocks = 0x40200000 0x5cc00 0x12af00 "u-boot-imx8mm-trusty-fit-enc.imx",
 \
             0x920000 0x187b00 0xdb30 "u-boot-imx8mm-trusty-fit-enc.imx", \
             0xbe000000 0x195630 0x191a00 "u-boot-imx8mm-trusty-fit-enc.imx"
```

按如下方式修改第二个文件csf_fit_sign_enc.txt，对之前生成的加密FIT镜像（u-boot-imx8mm-trusty-fit-enc.imx）进行签名。

5. “Authenticate Data”（验证数据）命令应覆盖整个FIT镜像：

```
[Authenticate Data]
    ...
    Blocks = 0x401fadc0 0x57c00 0x1020 "u-boot-imx8mm-trusty-fit-enc.imx"   \
             0x40200000 0x5cc00 0x12af00 "u-boot-imx8mm-trusty-fit-enc.imx",
 \
             0x920000 0x187b00 0xdb30 "u-boot-imx8mm-trusty-fit-enc.imx", \
             0xbe000000 0x195630 0x191a00 "u-boot-imx8mm-trusty-fit-enc.imx"
```

6. 添加“Install Secret Key”（安装密钥）命令，生成虚拟DEK blob文件。blob地址应与csf_fit_enc.txt中使用的地址相同：

```
[Install Secret Key]
  ...
  Key = "dek_fit_dummy.bin"
  Blob Address = 0x40400000
```

7. 添加“Decrypt Data”（解密数据）命令，对文件进行加密。由于镜像是在上述CSF中加密的，需要加密一个虚拟文件。blob地址应与csf_fit_enc.txt中使用的地址相同：

```
[Decrypt Data]
    ...
    Blocks = 0x40200000 0x5cc00 0x12af00 "u-boot-imx8mm-trusty-fit-enc-
dummy.imx", \
             0x920000 0x187b00 0xdb30 "u-boot-imx8mm-trusty-fit-enc-
dummy.imx", \
             0xbe000000 0x195630 0x191a00 "u-boot-imx8mm-trusty-fit-enc-
dummy.imx"
```

- 双引导加载程序情况

  对于双引导加载程序情况，CSF文件是基于bootloader-imx8mm-trusty-dual.img生成的。修改刚刚复制的csf_fit_enc.txt和csf_fit_sign_enc.txt。

  按如下方式修改第一个文件csf_fit_enc.txt，对FIT镜像进行加密并生成dek_fit.bin文件。

  1. 修改“Authenticate Data”（验证数据）命令，使其仅覆盖FIT镜像FDT标头：

```
[Authenticate Data]
  ...
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

```
  # Blocks = Authenticate_Start_Address fit_header_off fit_csf_off -
 fit_header_off
  Blocks = 0x401fadc0 0x0 0x1020 "bootloader-imx8mm-trusty-dual.img"
```

***注：**这里的所有十六进制数字都可以从构建日志获得。*

Authenticate_Start_Address是spl hab block项中的第一个十六进制数：

```
fit hab block:         0x401fadc0                    0x0
 0x1020
                       Authenticate_Start_Address    fit_header_off
 fit_csf_off - fit_header_off
```

2. 添加“Install Secret Key”（安装密钥）命令，以生成dek_fit.bin文件并安装blob。blob地址是iMX8M/soc.mak文件中imx-mkimage工程中定义的固定地址：

```
DEK_BLOB_LOAD_ADDR = 0x40400000
```

3. 修改Blob地址：

```
[Install Secret Key]
  ...
  Key = "dek_fit.bin"
  Blob Address = 0x40400000
```

CST只能加密16字节对齐的镜像。由于u-boot-nodtb.bin和u-boot.dtb共同16字节对齐，我们应该将print_fit_hab中提供的前两行视为一个块。

imx-mkimage输出：

```
0x40200000 0x5000 0x11ca50----+----Total length = 0x11ca50 + 0xe5f0 =
 0x12b040
0x4031ca50 0x121a50 0xe5f0 --- +
0x920000 0x130040 0xdb30
0xbe000000 0x13db70 0x191a00
```

4. 添加“Decrypt Data”（解密数据）命令，对文件进行加密。

```
[Decrypt Data]
    ...
    Blocks = 0x40200000 0x5000 0x12b040 "bootloader-imx8mm-trusty-dual-
enc.img", \
             0x920000 0x130040 0xdb30 "bootloader-imx8mm-trusty-dual-
enc.img", \
             0xbe000000 0x13db70 0x191a00 "bootloader-imx8mm-trusty-dual-
enc.img"
```

按如下方式修改第二个文件csf_fit_sign_enc.txt，以对之前生成的加密FIT镜像（bootloader-imx8mm-trusty-dual-enc.img）进行签名。

5. “Authenticate Data”（验证数据）命令应覆盖整个FIT镜像：

```
[Authenticate Data]
    ...
    Blocks = 0x401fadc0 0x0 0x1020 "bootloader-imx8mm-trusty-dual-enc.img"
 \
             0x40200000 0x5000 0x12b040 "bootloader-imx8mm-trusty-dual-
enc.img", \
             0x920000 0x130040 0xdb30 "bootloader-imx8mm-trusty-dual-
enc.img", \
             0xbe000000 0x13db70 0x191a00 "bootloader-imx8mm-trusty-dual-
enc.img"
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

6. 添加“Install Secret Key”（安装密钥）命令，以生成虚拟DEK blob文件，blob地址应与csf_fit_enc.txt中使用的地址相同：

```
[Install Secret Key]
  ...
  Key = "dek_fit_dummy.bin"
  Blob Address = 0x40400000
```

7. 添加“Decrypt Data”（解密数据）命令，对文件进行加密。由于镜像是在上述CSF中加密的，需要加密一个虚拟文件。blob地址应与csf_fit_enc.txt中使用的地址相同：

```
[Decrypt Data]
    ...
    Blocks = 0x40200000 0x5000 0x12b040 "bootloader-imx8mm-trusty-dual-enc-
dummy.img", \
             0x920000 0x130040 0xdb30 "bootloader-imx8mm-trusty-dual-enc-
dummy.img", \
             0xbe000000 0x13db70 0x191a00 "bootloader-imx8mm-trusty-dual-enc-
dummy.img"
```

3.1.2.10 对FIT镜像进行加密和签名

CST用于加密镜像并重新生成随机DEK。在此过程中，生成两个CSF二进制文件，但最终镜像中只包含一个二进制文件。

- 单引导加载程序情况
  对于单引导加载程序情况，执行以下命令，对镜像进行加密和签名。同时生成CSF描述文件。
  1. 对FIT镜像进行加密：

```
$ cp u-boot-imx8mm-trusty-spl-enc.imx u-boot-imx8mm-trusty-fit-enc.imx
$ ./cst -i csf_fit_enc.txt -o csf_fit_enc.bin
```

  2. 对加密FIT镜像进行签名：

```
$ cp u-boot-imx8mm-trusty-fit-enc.imx u-boot-imx8mm-trusty-fit-enc-dummy.imx
$ ./cst -i csf_fit_sign_enc.txt -o csf_fit_sign_enc.bin
```

- 双引导加载程序情况
  对于双引导加载程序情况，执行以下命令。
  1. 对FIT镜像进行加密：

```
$ cp bootloader-imx8mm-trusty-dual.img bootloader-imx8mm-trusty-dual-enc.img
$ ./cst -i csf_fit_enc.txt -o csf_fit_enc.bin
```

  2. 对加密FIT镜像进行签名：

```
$ cp bootloader-imx8mm-trusty-dual-enc.img bootloader-imx8mm-trusty-dual-
enc-dummy.img
$ ./cst -i csf_fit_sign_enc.txt -o csf_fit_sign_enc.bin
```

3.1.2.11 创建FIT镜像的最终CSF二进制文件

由于最终镜像中只包含一个CSF二进制文件，需要将Nonce/MAC从csf_fit_enc.bin换为csf_fit_sign_enc.bin。以下步骤对于单引导加载程序和双引导加载程序情况是相同的。

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

1. 根据CSF中的MAC字节值计算Nonce/MAC大小：

```
Nonce/MAC size = Nonce size + MAC bytes + CSF header for Nonce/Mac = 12 + 16
 + 8 = 36 bytes
```

2. 计算csf_fit_enc.bin中的Nonce/MAC偏移值：

```
MAC offset = csf_fit_enc.bin size - Nonce/MAC size = 3996 - 36 = 3960 Bytes
```

3. 从csf_fit_enc.bin提取Nonce/MAC：

```
$ dd if=csf_fit_enc.bin of=noncemac.bin bs=1 skip=3960 count=36
```

4. 计算csf_fit_sign_enc.bin中的Nonce/MAC偏移值：

```
MAC offset = csf_fit_enc.bin size - Nonce/MAC size = 4020 - 36 = 3984 Bytes
```

5. 用上面提取的地址替换csf_fit_sign_enc.bin的MAC地址：

```
$ dd if=noncemac.bin of=csf_fit_sign_enc.bin bs=1 seek=3984 count=36
```

3.1.2.12 生成DEK Blob

DEK必须封装到CAAM blob中，以便包含在最终的加密二进制文件中。U-Boot提供了一个名为generate-dek-blob的fastboot命令，该命令调用Trusty OS中包含的CAAM实现。此命令仅在具有闭合配置和签名镜像的电路板上有效。参见第3.1.2节生成安全启动镜像并正确关闭电路板。以下步骤对于单引导加载程序和双引导加载程序情况是相同的。

启动电路板并进入fastboot模式。从主机PC执行以下命令：

```
$ fastboot stage dek_spl.bin
$ fastboot oem generate-dek-blob
$ fastboot get_staged dek_spl_blob.bin
$ fastboot stage dek_fit.bin
$ fastboot oem generate-dek-blob
$ fastboot get_staged dek_fit_blob.bin
```

然后，主机PC生成dek_spl_blob.bin和dek_fit_blob.bin。

3.1.2.13 插入CSF文件和DEK blob

- 单引导加载程序情况

1. 对于单引导加载程序情况，从主机PC执行以下命令：

```
$ cp u-boot-imx8mm-trusty-fit-enc.imx encrypted-u-boot-imx8mm-trusty.imx
```

2. imx-mkimage在目标文件中为CSF二进制文件预留了空间。查看构建日志。SPL的CSF二进制文件偏移值打印在csf_off标签后。将csf_spl_sign_enc.bin插入encrypted-u-boot-imx8mm-trusty.imx中，偏移值为0x33e00：

```
$ dd if=csf_spl_sign_enc.bin of=encrypted-u-boot-imx8mm-trusty.imx seek=
$((0x33e00)) bs=1 conv=notrunc
```

3. CSF预留空间的大小为0x2000字节。dek_blob应添加在其旁边。将dek_spl_blob.bin插入encrypted-u-boot-imx8mm-trusty.imx，偏移值为0x33e00 + 0x2000：

```
$ dd if=dek_spl_blob.bin of=encrypted-u-boot-imx8mm-trusty.imx seek=
$((0x35e00)) bs=1 conv=notrunc
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

4. FIT的CSF二进制文件偏移值打印在sld_csf_off标签之后。将csf_fit_sign_enc.bin插入到encrypted-u-boot-imx8mm-trusty.imx中，偏移值为0x58c20：

```
$ dd if=csf_fit_sign_enc.bin of=encrypted-u-boot-imx8mm-trusty.imx seek=
$((0x58c20)) bs=1 conv=notrunc
```

5. DEK blob必须插入到FIT镜像的最后一个镜像项中。可以使用print_fit_hab目标日志提供的最后一行：

```
0x40200000 0x5CC00 0x11C918
0x4031C918 0x179518 0xE5E8
0x920000 0x187B00 0xDB30
0xBE000000 0x195630 0x191A00 -> Last line in print_fit_hab log
```

6. 将dek_fit_blob.bin插入encrypted-u-boot-imx8mm-trusty.imx中，偏移值为0x195630 + 0x191a00：

```
$ dd if=dek_fit_blob.bin of=encrypted-u-boot-imx8mm-trusty.imx seek=
$((0x327030)) bs=1 conv=notrunc
```

- 双引导加载程序情况

1. 对于双引导加载程序情况，从主机PC执行以下命令：

```
$ cp spl-imx8mm-trusty-dual-enc.bin encrypted-spl-imx8mm-trusty-dual.bin
$ cp bootloader-imx8mm-trusty-dual-enc.img encrypted-bootloader-imx8mm-
trusty-dual.img
```

2. 与单引导加载程序情况一样，SPL的CSF二进制文件偏移值打印在csf_off标签后。将csf_spl_sign_enc.bin插入encrypted-spl.bin中，偏移值为0x35200：

```
$ dd if=csf_spl_sign_enc.bin of=encrypted-spl-imx8mm-trusty-dual.bin seek=
$((0x34600)) bs=1 conv=notrunc
```

3. CSF预留空间的大小为0x2000字节。dek_blob应添加在其旁边。将dek_spl_blob.bin插入到encrypted-spl.bin中，偏移值为0x35200 + 0x2000：

```
$ dd if=dek_spl_blob.bin of=encrypted-spl-imx8mm-trusty-dual.bin seek=
$((0x36600)) bs=1 conv=notrunc
```

4. 与单引导加载程序情况不同的是，FIT的CSF二进制文件偏移值打印在fit_csf_off标签之后。将csf_fit_sign_enc.bin插入到encrypted-fit.bin中，偏移值为0x1020：

```
$ dd if=csf_fit_sign_enc.bin of=encrypted-bootloader-imx8mm-trusty-dual.img
 seek=$((0x1020)) bs=1 conv=notrunc
```

5. DEK blob必须插入到FIT镜像的最后一个镜像项中。可以使用print_fit_hab目标日志提供的最后一行：

```
0x40200000 0x5000 0x11CA50
0x4031CA50 0x121A50 0xE5F0
0x920000 0x130040 0xDB30
0xBE000000 0x13DB70 0x191A00 -> Last line in print_fit_hab log
```

6. 将dek_fit_blob.bin插入到encrypted-fit.bin中，偏移值为0x13db70 + 0x191a00：

```
$ dd if=dek_fit_blob.bin of=encrypted-bootloader-imx8mm-trusty-dual.img
 seek=$((0x2cf570)) bs=1 conv=notrunc
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

3.1.2.14 烧写加密启动镜像

成功执行上述命令后，生成已签名和加密的镜像。将这些镜像复制回输出目录，像以前一样更改它们的名称，并将它们烧写到相应的分区。如果一切操作正确，电路板将成功启动。

3.2 TEE上的配置

3.2.1 ATF中的存储区域配置

TEE二进制文件由SPL加载到DRAM的$BL32_BASE地址。默认情况下，加载地址$BL32_BASE被定义为0xFE000000。它是在使用imx-mkimage生成引导加载程序镜像的过程中指定的。例如，您可以在${MY_ANDROID}/vendor/nxp-opensource/imx-mkimage中为i.MX 8QuadMax和i.MX 8QuadXPlus指定加载地址为0xFF000000，如下所示：

```
diff --git a/iMX8QM/soc.mak b/iMX8QM/soc.mak
index 355851e..fe70191 100644
--- a/iMX8QM/soc.mak
+++ b/iMX8QM/soc.mak
@@ -82,7 +82,7 @@ u-boot-atf-container.img: bl31.bin u-boot-hash.bin
fi
if [ -f "tee.bin" ]; then \
    if [ $(shell echo $(ROLLBACK_INDEX_IN_CONTAINER)) ]; then \
       - ./$(MKIMG) -soc QM -sw_version $(ROLLBACK_INDEX_IN_CONTAINER) -rev B0 -
c -ap bl31.bin a53 0x80000000 -ap u-boot-hash.bin a53 0x80020000 -ap tee.bin a53
 0xFE000000 -out u-boot-atf-container.img; \
       + ./$(MKIMG) -soc QM -sw_version $(ROLLBACK_INDEX_IN_CONTAINER) -rev B0 -
c -ap bl31.bin a53 0x80000000 -ap u-boot-hash.bin a53 0x80020000 -ap tee.bin a53
 0xFF000000 -out u-boot-atf-container.img; \
   else \
       ./$(MKIMG) -soc QM -rev B0 -c -ap bl31.bin a53 0x80000000 -ap u-
boot-hash.bin a53 0x80020000 -ap tee.bin a53 0xFE000000 -out u-boot-atf-
container.img; \
fi; \
diff --git a/iMX8QX/soc.mak b/iMX8QX/soc.mak
index 56422e0..d917dc3 100644
--- a/iMX8QX/soc.mak
+++ b/iMX8QX/soc.mak
@@ -73,7 +73,7 @@ u-boot-atf.itb: u-boot-hash.bin bl31.bin
u-boot-atf-container.img: bl31.bin u-boot-hash.bin
if [ -f tee.bin ]; then \
   if [ $(shell echo $(ROLLBACK_INDEX_IN_CONTAINER)) ]; then \
       - ./$(MKIMG) -soc QX -sw_version $(ROLLBACK_INDEX_IN_CONTAINER) -rev B0 -
c -ap bl31.bin a35 0x80000000 -ap u-boot-hash.bin a35 0x80020000 -ap tee.bin a35
 0xFE000000 -out u-boot-atf-container.img; \
       + ./$(MKIMG) -soc QX -sw_version $(ROLLBACK_INDEX_IN_CONTAINER) -rev B0 -
c -ap bl31.bin a35 0x80000000 -ap u-boot-hash.bin a35 0x80020000 -ap tee.bin a35
 0xFF000000 -out u-boot-atf-container.img; \
   else \
       ./$(MKIMG) -soc QX -rev B0 -c -ap bl31.bin a35 0x80000000 -ap u-
boot-hash.bin a35 0x80020000 -ap tee.bin a35 0xFE000000 -out u-boot-atf-
container.img; \
   fi; \
```

将TEE二进制文件加载到DRAM后，ATF尝试在$BL32_BASE地址（大小为$BL32_SIZE）启动它，这些在${MY_ANDROID}/vendor/nxp-opensource/arm-trusted-firmware/plat/imx/$(PLAT)/include/platform_def.h中定义。默认情况下，$BL32_BASE被定义为0xFE000000，$BL32_SIZE被定义为0x02000000，但您可以根据需要进行配置。

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

例如，对于i.MX 8QuadMax和i.MX 8QuadXPlus，$BL32_BASE可以配置为0xFF000000，$BASE_SIZE可以配置为0x03000000，如下所示：

```
diff --git a/plat/imx/imx8qm/include/platform_def.h b/plat/imx/imx8qm/include/
platform_def.h
index b305bfc..6f9f7d4 100644
--- a/plat/imx/imx8qm/include/platform_def.h
+++ b/plat/imx/imx8qm/include/platform_def.h
@@ -37,8 +37,8 @@
#define BL31_LIMIT 0x80020000
#ifdef TEE_IMX8
-#define BL32_BASE 0xfe000000
-#define BL32_SIZE 0x02000000
+#define BL32_BASE 0xff000000
+#define BL32_SIZE 0x03000000
#define BL32_SHM_SIZE 0x00400000
#define BL32_LIMIT 0x100000000
#endif
diff --git a/plat/imx/imx8qxp/include/platform_def.h b/plat/imx/imx8qxp/include/
platform_def.h
index 24eacc2..cfc0717 100644
--- a/plat/imx/imx8qxp/include/platform_def.h
+++ b/plat/imx/imx8qxp/include/platform_def.h
@@ -33,8 +33,8 @@
#define BL31_LIMIT 0x80020000
#ifdef TEE_IMX8
-#define BL32_BASE 0xfe000000
-#define BL32_SIZE 0x02000000
+#define BL32_BASE 0xff000000
+#define BL32_SIZE 0x03000000
#define BL32_SHM_SIZE 0x00400000
#define BL32_LIMIT 0x100000000
#define PLAT_TEE_IMAGE_OFFSET 0x84000000
```

下表列出了i.MX 8Quad平台上不同尺寸DRAM的推荐$BL32_BASE和$BL32_SIZE：

表3. DRAM的推荐$BL32_BASE和$BL32_SIZE

DRAM的大小 (GB)	$BL32_BASE	$BL32_SIZE
6	0xFE000000	0x02000000
4	0xFE000000	0x02000000
3	0xFE000000	0x02000000
2	0xFE000000	0x02000000
1	0xBE000000	0x02000000

3.2.2 Trusty OS的基本文件和文件夹构造

i.MX Trusty OS为Android平台和Android Automotive平台提供了安全解决方案。它还提供了一套开发API，供客户开发自己的TA。

Trusty OS基于LittleKernel。i.MX Trusty OS具有以下基本文件结构。

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

表4. i.MX Trusty OS的基本文件结构

文件夹名称	文件夹说明
trusty/device/nxp/imx8	此文件夹包含脚本文件。构建对象的大多数配置在此文件夹中定义，包括工程配置文件。 Makefile配置、电路板配置和需要构建的模块。
trusty/hardware/nxp/app	恩智浦专用的TA源代码文件夹。目前位于此文件夹中、提供安全功能的hwcrypto TA依赖于i.MX SoC硬件。
trusty/hardware/nxp/target	恩智浦参考电路板对象文件夹。此文件中仅定义了此文件夹中构建对象的rules.mk、平台名称和UART信息。
trusty/hardware/nxp/platform/imx	恩智浦面向Trusty OS的SoC专用源代码。所有i.MX SoC都共享这些代码。包括平台初始化代码、UART驱动程序和寄存器映射定义。
trusty/kernel/lib	Trusty OS核心代码，包括安全监测器调用管理、TIPC/QL-TIPC协议栈。
external/lk	LittleKernel代码，包括所有LittleKernel模块，如架构代码、中断管理、任务管理和SMP支持。
trusty/user/app	Trusty OS的TA放置在此处，包括AVB、Gatekeeper和Keymaster用户空间源代码。

有关TA实现，参见Google Trusty OS参考网页：https://source.android.com/security/trusty/trusty-ref。

3.2.3 在Trusty OS中应用新的构建对象

默认情况下，恩智浦已在i.MX Trusty OS中提供了i.MX 8QuadMax/8QuadXPlus和 i.MX 8M Mini/8M Quad系列模板。要基于i.MX 8QuadMax/8QuadXPlus或i.MX 8M Mini/8M Quad/8M Plus添加新平台，需要添加或修改以下文件或模块。

在${MY_TRUSTY}/trusty/device/nxp/imx8/project中，imx8-inc.mk包含所有常见配置，例如CPU核和需要构建的模块。imx8-inc.mk可以被构建对象的mk文件（如imx8qm.mk）覆盖。

例如，要添加一个基于i.MX 8QuadMax SoC、名为imx8qm-abc的新构建对象，该对象有6个CPU和1024个RPMB块，请在${MY_TRUSTY}trusty/device/nxp/imx8/project中编写一个名为imx8qm-abc.mk的新.mk文件。内容如下：

```
TARGET := imx8q
# imx8q/x use lpuart for UART IP
IMX_USE_LPUART := true
SMP_MAX_CPUS := 6
STORAGE_RPMB_BLOCK_COUNT := 1024
include project/imx8-inc.mk
```

在Trusty OS代码的根目录中，执行$make list。然后显示imx8qm-abc。

3.2.4 在Trusty OS中添加单元测试并添加CAAM自检

Trusty OS支持两个单元测试，以测试Trusty IPC（TIPC）和CAAM的功能。这些测试仅用于调试目的，不应与开源单元测试一起发布。对于i.MX 8QuadMax和i.MX 8QuadXPlus，要包含这些单元测试，需要在${MY_TRUSTY}/trusty/device/nxp/imx8/project中进行以下更改：

```
diff --git a/project/imx8-inc.mk b/project/imx8-inc.mk
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

```
index 681a223..e7dcfdb 100644
--- a/project/imx8-inc.mk
+++ b/project/imx8-inc.mk
@@ -70,6 +70,7 @@ GLOBAL_DEFINES += APP_STORAGE_RPMB_BLOCK_COUNT=$(STORAGE_RPMB_BLOCK_COUNT)
GLOBAL_DEFINES += \
WITH_LIB_VERSION=1 \
+   WITH_CAAM_SELF_TEST=1 \
# ARM suggest to use system registers to access GICv3/v4 registers
GLOBAL_DEFINES += ARM_GIC_USE_SYSTEM_REG=1
@@ -98,6 +99,8 @@ TRUSTY_ALL_USER_TASKS := \
trusty/user/app/keymaster \
trusty/user/app/gatekeeper \
trusty/user/app/storage \
+   trusty/user/base/lib/tipc/test/main \
+   trusty/user/base/lib/tipc/test/srv \
# This project requires trusty IPC
WITH_TRUSTY_IPC := true
```

重新构建Trusty OS，并将输出二进制文件复制到`${MY_ANDROID}/vendor/nxp/fsl-proprietary/uboot-firmware/imx8q_car`。进行以下更改，构建`trusty-ut-ctrl`二进制文件：

```
diff --git a/imx8q/mek_8q/mek_8q.mk b/imx8q/mek_8q/mek_8q.mk
index f34f38e1..ea64ed94 100644
--- a/imx8q/mek_8q/mek_8q.mk
+++ b/imx8q/mek_8q/mek_8q.mk
@@ -277,7 +277,8 @@ PRODUCT_PACKAGES += \
        android.hardware.gatekeeper-service.trusty \
        storageproxyd \
        imx_dek_extractor \
-    imx_dek_inserter
+    imx_dek_inserter \
+    trusty-ut-ctrl
    endif

    ifeq ($(PRODUCT_IMX_TRUSTY),true)
```

重新构建Android工程，`trusty-ut-ctrl`二进制文件位于`${MY_ANDROID}/out/target/product/mek_8q/vendor/bin/trusty-ut-ctrl`。

将镜像烧写到电路板上，Trusty OS在初始化CAAM时运行CAAM单元测试，如果CAAM正确初始化，以下日志会在U-Boot中显示：

```
hwsecure: 22: hwsecure init.
caam_drv: 2425: caam hwrng test PASS!!!
caam_drv: 2458: caam blob test PASS!!!
caam_drv: 969: dek blob decap succeed!
caam_drv: 2482: caam dek blob test PASS!!!
caam_drv: 2565: caam gen kdf root key test PASS!!!
caam_drv: 2515: caam AES enc test PASS!!!
caam_drv: 2524: caam AES enc test PASS!!!
caam_drv: 2552: caam hash test PASS!!!
caam_drv: 2645: AES CBC encrytion test passed!
caam_drv: 2653: AES CBC decrytion test passed!
caam_drv: 2596: AES ECB encrytion test passed!
caam_drv: 2604: AES ECB decrytion test passed!
caam_drv: 2703: AES CTR encrytion test passed!
caam_drv: 2711: AES CTR decrytion test passed!
caam_drv: 2775: AES GCM encryption test passed!
caam_drv: 2784: AES GCM decryption test passed!
caam_drv: 2820: DES EDE ECB encrytion test passed!
caam_drv: 2828: DES EDE ECB decrytion test passed!
caam_drv: 2899: DES EDE CBC encrytion test passed!
caam_drv: 2907: DES EDE CBC decrytion test passed!
```

运行以下命令，测试Trusty进程间通讯（IPC）：

```
mek_8q:/ # trusty-ut-ctrl com.android.ipc-unittest.ctrl
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

正确的结果如下所示：

```
ipc-unittest-main: 2778: first_free_handle_index: 3
ipc-unittest-main: 2762: retry ret 0, event handle 1000, event 0x1
ipc-unittest-main: 2765: nested ret -13, event handle 1000, event 0x1

[ RUN      ] ipc.wait_negative
[       OK ] ipc.wait_negative
[ RUN      ] ipc.close handle negative
[       OK ] ipc.close_handle_negative
[ RUN      ] ipc.set_cookie_negative
[       OK ] ipc.set cookie negative
...
[ RUN      ] ipc.recv handle
[       OK ] ipc.recv handle
[ RUN      ] ipc.recv handle negative
[       OK ] ipc.recv handle negative
[ RUN      ] ipc.send handle bulk
[       OK ] ipc.send handle bulk
[ RUN      ] ipc.echo handle bulk
[       OK ] ipc.echo handle bulk
[ RUN      ] ipc.tipc connect
[       OK ] ipc.tipc connect
[ RUN      ] ipc.tipc send recv 1
[       OK ] ipc.tipc send recv 1
[ RUN      ] ipc.tipc send recv hdr payload
[       OK ] ipc.tipc send recv hdr payload
[ RUN      ] ipc.dup is different
[       OK ] ipc.dup_is_different
[==========] 42 tests ran.
[ PASSED  ]  42 tests.
[ DISABLED ] 2 tests.
```

3.2.5 修改Trusty OS的控制台端口

由于硬件板设计不同，调试UART可能会不同。i.MX Trusty OS支持通过修改源代码来配置不同的UART端口。

调试UART地址在`trusty/hardware/nxp/platform/imx/soc/$SOC_NAME/include/imx-regs.h`中定义。

例如，如果对于i.MX 8QuadMax电路板使用LPUART1而非LPUART0，则需要在`imx-regs.h`中进行以下修改：

```
diff --git a/platform/imx/soc/imx8q/include/imx-regs.h b/platform/imx/soc/imx8q/
include/imx-regs.h
index 3cbcce2..0235e69 100644
--- a/platform/imx/soc/imx8q/include/imx-regs.h
+++ b/platform/imx/soc/imx8q/include/imx-regs.h
@@ -40,7 +40,7 @@
    #define CONFIG_LPUART3_BASE 0x5A090000
    #define CONFIG_LPUART4_BASE 0x5A0A0000

-#define CONFIG_CONSOLE_TTY_BASE  CONFIG_LPUART0_BASE
+#define CONFIG_CONSOLE_TTY_BASE  CONFIG_LPUART1_BASE

    #define CONFIG_CONSOLE_TTY_VIRT (0xFFFFFFFF00000000 +
 CONFIG_CONSOLE_TTY_BASE)
```

3.2.6 配置相关的TA服务

可信应用程序（TA）是指在安全环境中运行的软件。在Trusty OS中运行着多个TA。下图展示了它们之间的关系。

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

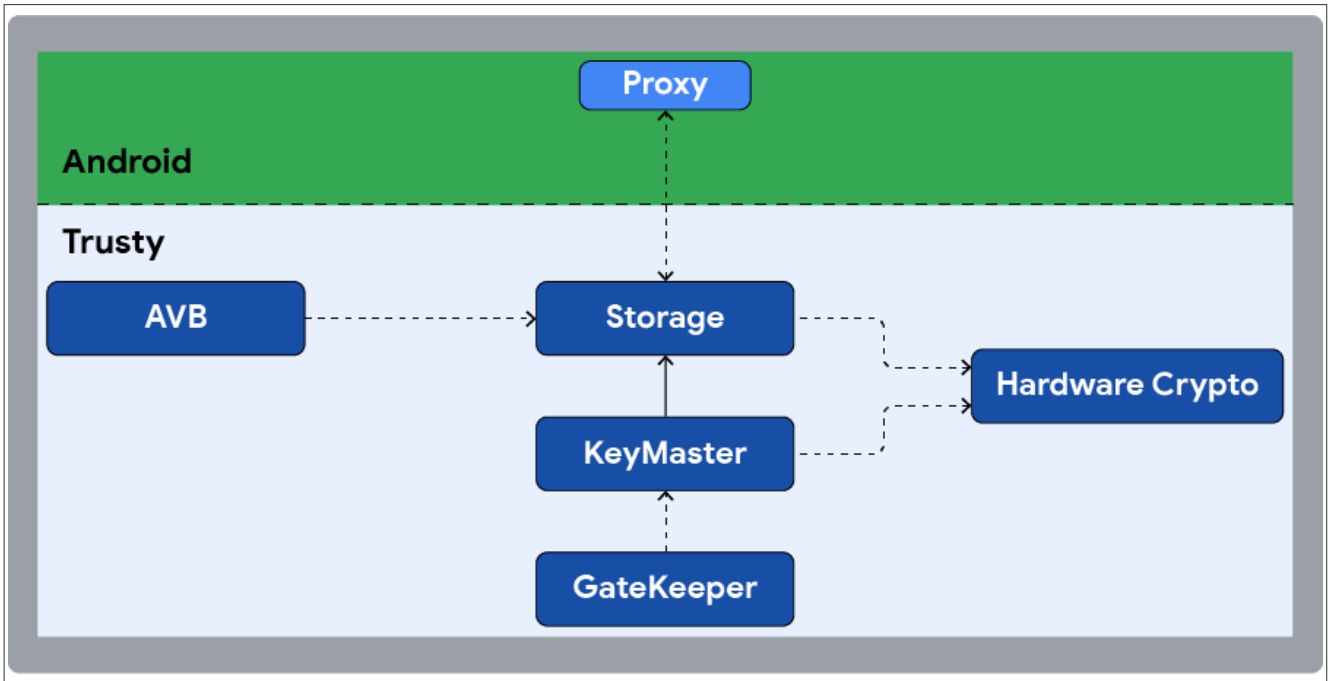


图10. TA之间的关系

- AVB TA：为Android验证启动（AVB）期间使用的数据提供防篡改操作，如回滚索引、锁定/解锁状态和vbmeta公钥。
- 存储TA：提供加密和防篡改存储，以保护应用程序，如AVB TA。所有修改安全存储区的操作都是事务性的。
- 硬件加密TA：提供硬件加密，并加速基于CAAM的操作，如RNG生成和SHA1/SHA256哈希计算。
- Keymaster TA：提供所有安全密钥库操作，访问原始密钥材料，验证密钥上的所有访问控制条件。
- Gatekeeper TA：对用户密码进行身份验证，并生成身份验证令牌，用于向Keymaster TA证明在特定时间点对特定用户进行了身份验证。

3.2.7 指定应用加载程序加密和签名密钥

Trusty OS中的可信应用程序（TA）可以内置到Trusty二进制文件中，也可以构建为可加载模块。在将TA构建为可加载模块的情况下，构建系统会对模块进行加密和签名，以确保其完整性和有效性。

1. 指定应用加载程序加密密钥。
   如果指定了加密密钥文件，Trusty OS构建系统将对可加载模块进行加密（使用AES-GCM模式）。加密密钥可以是任何有效的自定义密钥。示例加密密钥位于：`${MY_TRUSTY}/trusty/device/nxp/imx8/project/keys/aeskey.bin`。
   生成方式：

```
openssl rand 16 -out aeskey.bin
```

   注： *此示例加密密钥必须在装运产品前用自定义密钥替换。*
   此外，支持两种加密密钥，用于不同的用途。生成加密密钥后，在`${MY_TRUSTY}/trusty/device/nxp/imx8/project/imx8-inc.mk`中指定加密密钥文件，如下所示：

```
APPLOADER_ENCRYPT_KEY_0_FILE := <path-to-encryption-key-0>
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

```
APPLOADER_ENCRYPT_KEY_1_FILE := <path-to-encryption-key-1>
```

要对应用程序进行加密，需要在应用程序的rules.mk中设置相应的变量，如下所示：

```
# choose encryption key 0
APPLOADER_ENCRYPT_KEY_ID_FOR_$(MODULE) := 0
```

或

```
# choose encryption key 1
APPLOADER_ENCRYPT_KEY_ID_FOR_$(MODULE) := 1
```

2. 指定应用加载程序签名密钥。
   Trusty OS构建系统会对可加载模块进行签名并为其生成ECDSA签名。ECDSA签名密钥对的示例位于：${MY_TRUSTY}/trusty/device/nxp/imx8/project/keys。它们由{MY_TRUSTY}/trusty/user/base/app/apploader/generate_ecdsa_keys.sh中的脚本生成：

```
./generate_ecdsa_keys.sh <private key file> <public key file>
```

***注：**这些示例签名密钥必须在装运产品前用自定义密钥替换。*
与加密密钥一样，可以提供两对签名密钥文件，用于不同目的。生成签名密钥对后，在${MY_TRUSTY}/trusty/device/nxp/imx8/project/imx8-inc.mk中指定密钥文件，如下所示：

```
134 PROJECT_KEYS_DIR := trusty/device/nxp/imx8/project/keys
135
136 APPLOADER_SIGN_PRIVATE_KEY_0_FILE := \
137         $(PROJECT_KEYS_DIR)/privateKey0.der
138
139 APPLOADER_SIGN_PUBLIC_KEY_0_FILE := \
140         $(PROJECT_KEYS_DIR)/publicKey0.der
141
142 APPLOADER_SIGN_PRIVATE_KEY_1_FILE := \
143         $(PROJECT_KEYS_DIR)/privateKey1.der
144
145 APPLOADER_SIGN_PUBLIC_KEY_1_FILE := \
145         $(PROJECT_KEYS_DIR)/publicKey1.der
147
150
151 APPLOADER_SIGN_KEY_ID ?= 0
```

如果未明确指定应用加载程序签名密钥，Trusty OS将使用APPLOADER_SIGN_PRIVATE_KEY_0_FILE对可加载模块进行签名（因为“APPLOADER_SIGN_KEY_ID”为0）。要使用不同的密钥对应用程序进行签名，需要在应用程序的rules.mk中设置相应的变量，如下所示：

```
# sign with key 0
APPLOADER_SIGN_KEY_ID_FOR_$(MODULE) := 0
```

或

```
# sign with key 1
APPLOADER_SIGN_KEY_ID_FOR_$(MODULE) := 1
```

3.2.8 配置可加载可信应用程序的回滚版本

在Trusty中，应用加载程序可以防止可加载TA回滚攻击。

1. 应用加载程序从TA的清单中获取当前版本，并从安全存储区中获取TA的最后一个stored_version。比较这两个版本。如果TA的当前版本高于或等于stored_version，则加载TA。

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

2. 从TA的清单中获取min_version。如果min_version高于stored_version，则stored_version将在RPMB安全存储区中更新为min_version。

例如，可加载TA的回滚版本可以通过以下diff添加。

```
diff --git a/manifest.json b/manifest.json
index 653201c..2be563c 100644
--- a/manifest.json
+++ b/manifest.json
@@ -1,5 +1,10 @@
     {
         "uuid": "7dee2364-c036-425b-b086-df0f6c233c1b",
         "min_heap": 524288,
-     "min_stack": 65536
+     "min_stack": 65536,
+     "version": 1,
+     "min_version": 0
     }
```

***注：** version不能低于min_version，否则构建将失败。*

3.3 U-Boot中的安全配置

U-Boot由SPL加载，并通过HAB进行验证。ATF启动U-Boot。U-Boot的主要目的是加载和验证Android镜像。

3.3.1 U-Boot中的安全功能概述

i.MX Android镜像中启用了Android验证启动（AVB）。AVB中使用了一个额外的vbmeta镜像。这个vbmeta镜像未包含芯片将执行的任何代码。U-Boot使用该镜像对自身和其它Android镜像进行身份验证。其它要通过vbmeta镜像进行身份验证的镜像包括用于“启动”、“dtbo”、“系统”和“供应商”分区的镜像。如果还有其它包含执行代码的镜像，如产品，它们也基于vbmeta镜像中的数据进行身份验证。计算这些镜像的哈希值，并将元数据存储在vbmeta镜像中。下图展示了这些镜像之间的关系。

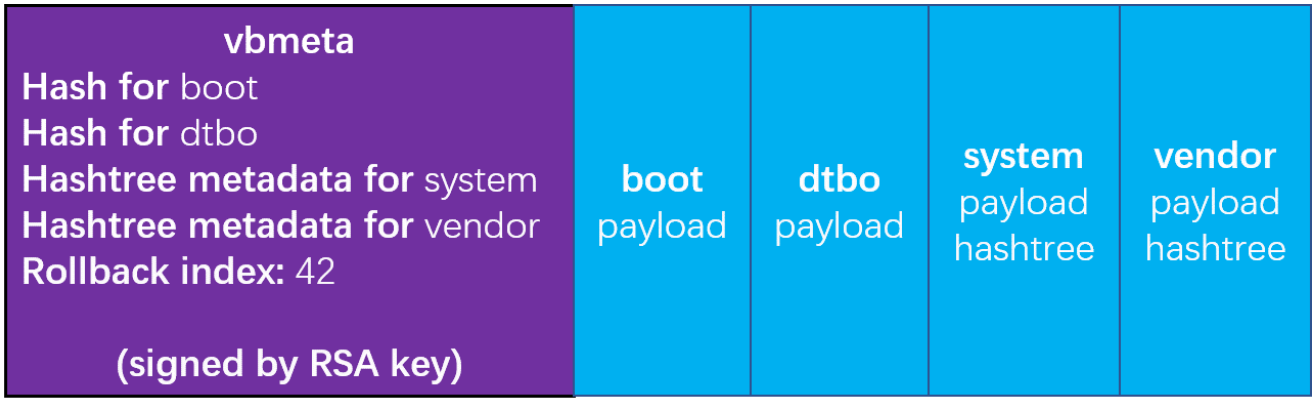


图11. vbmeta镜像与相关镜像之间的关系

为了确保vbmeta镜像是可信的，使用RSA密钥对其进行签名，在使用该镜像中的数据验证其它镜像之前，在启动时验证vbmeta镜像的签名。

为了防止回滚攻击，vbmeta镜像中存储了一个回滚索引值。当新的镜像发布时，这个回滚索引值会相应地增加。在所有镜像被验证为可启动后，该vbmeta镜像中的回滚索引值也会保存在eMMC的RPMB分区中。

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

如果vbmeta镜像中的回滚索引值小于在eMMC的RPMB分区中存储的值，U-Boot不会使用相关镜像启动。在启用双引导加载程序的情况下，SPL和U-Boot本身不在同一个文件中，因此U-Boot本身有另一个回滚索引值。

为了防止芯片在OTA期间被卡住，提供了a/b插槽功能。一些用于存储镜像的分区在启动芯片中有两个副本，名为“插槽a”和“插槽b”。镜像更新流程仅烧写一个插槽。更新失败不会影响另一个插槽。

3.3.2 生成和烧录eMMC RPMB密钥

eMMC的RPMB分区可以使用256位安全密钥烧录。安全密钥只能烧写一次。此安全密钥只能由安全环境（TEE或SPL）访问。它用于对eMMC RPMB和TEE之间的数据传输进行签名和验证。

有两种RPMB密钥生成方式：CAAM/ELE硬件绑定密钥和供应商指定密钥。这两种方式都与CAAM或ELE相关，使用efuse硬件中的值。如果需要将SRK哈希值烧写到efuse硬件并关闭芯片，请先执行此操作，然后才能对RPMB密钥进行烧写。

- CAAM/ELE硬件绑定密钥
  RPMB密钥可以从CAAM/ELE派生出来。该密钥与硬件绑定，且每个芯片唯一。TEE在每次启动时都会从CAAM/ELE派生出这个硬件绑定密钥，并且由于该密钥与CAAM/ELE硬件绑定，不需要存储此密钥的一个副本。这种方式优先，因为它更简单、更安全。
  提供了fastboot命令，从CAAM/ELE硬件绑定密钥设置RPMB密钥：

```
$ fastboot oem set-rpmb-hardware-key
```

  还提供了另一种方法，对从CAAM/ELE派生的RPMB密钥进行烧写，这种方法仅在HAB关闭的电路板上启用。以i.MX 8MP开发板为例，修改defconfig文件，如下所示：

```
diff --git a/configs/imx8mp_evk_android_trusty_dual_defconfig b/configs/
imx8mp_evk_android_trusty_dual_defconfig
index 4e2238d789..b05627c255 100644
--- a/configs/imx8mp_evk_android_trusty_dual_defconfig
+++ b/configs/imx8mp_evk_android_trusty_dual_defconfig
@@ -229,3 +229,4 @@ CONFIG_ATTESTATION_ID_DEVICE="evk_8mp"
 CONFIG_ATTESTATION_ID_PRODUCT="evk_8mp"
 CONFIG_ATTESTATION_ID_MANUFACTURER="nxp"
 CONFIG_ATTESTATION_ID_MODEL="EVK_8MP"
+CONFIG_AUTO_SET_RPMB_KEY=y
```

  更新引导加载程序并启动，RPMB密钥在HAB关闭的电路板上自动设置。

- 供应商指定密钥
  供应商可以在带有CAAM硬件的平台上指定任意自定义的RPMB密钥，但这种方式不适用于仅支持ELE的平台（例如i.MX 95）。密钥的一个副本用CAAM封装，生成的密钥blob被保存到eMMC BOOT1分区的最后一个块中。在i.MX 8QuadMax MEK和i.MX 8QuadXPlus MEK上，eMMC BOOT1分区大小为8MB。在i.MX 8M Mini EVK、i.MX 8M Plus EVK和i.MX8M Nano EVK上则为4MB。为了防止系统运行时密钥blob被篡改，BOOT1分区设置了电路板启动时上电写保护。
  根据电路板的设计，密钥blob的存储位置可能需要更改。有两个宏用于控制密钥blob的位置。这两个宏对于i.MX 8QuadMax MEK和i.MX 8QuadXPlus MEK是相同的。其定义如下：

```
#define KEYSLOT_HWPARTITION_ID 2
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

```
#define KEYSLOT_BLKS 0x3FFF
```

而对于i.MX 8M Mini EVK和i.MX 8M Nano EVK，定义如下：

```
#define KEYSLOT_HWPARTITION_ID 2
#define KEYSLOT_BLKS 0x1FFF
```

KEYSLOT_HWPARTITION_ID表示eMMC分区。**0**表示USERDATA分区，**1**表示BOOT0分区，**2**表示BOOT1分区。KEYSLOT_BLKS表示存储密钥blob的块。

对于i.MX 8QuadMax MEK，它们位于以下文件中。对于其它平台，则位于同一文件夹中的其它电路板相关头文件中。

```
/* Android Automotive */
${MY_ANDROID}/vendor/nxp-opensource/uboot-imx/include/configs/
imx8qm_mek_android_auto.h
/* Standard Android */
${MY_ANDROID}/vendor/nxp-opensource/uboot-imx/include/configs/
imx8qm_mek_android.h
```

自定义RPMB密钥文件应以魔术字“RPMB”开头，随后是原始密钥。以下步骤展示了如何生成默认密钥文件rpmb_key_test.bin：

```
$ touch rpmb_key_test.bin
$ echo -n "RPMB" > rpmb_key_test.bin
$ echo -n -e
 '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'>> rpmb_key_test.bin
```

\xHH表示8位字符，其值为十六进制值“HH”。可以用您想要设置的密钥替换上述“00”。

生成自定义RPMB密钥文件后，可以使用以下fastboot命令设置RPMB密钥：

```
$ fastboot stage rpmb_key_test.bin
$ fastboot oem set-rpmb-staged-key
```

使用两种方式之一对RPMB密钥进行烧写后，重启电路板。Trusty OS中的RPMB服务随后成功初始化。

3.3.3 生成用于镜像签名和验证的密钥

在构建Android平台时对镜像进行签名，以确保其完整性。一对非对称密钥（AVB密钥）用于对vbmeta镜像中的vbmeta结构进行签名。在标准的Android平台上，还有一对非对称密钥（AVB启动密钥），用于对启动镜像中的vbmeta结构进行签名，该结构将被构建为“链式分区”。有关链式分区的更多信息，请参见 https://android.googlesource.com/platform/external/avb/+/master/README.md。

默认情况下，用于对vbmeta镜像中的vbmeta结构进行签名的AVB私钥位于：

```
${MY_ANDROID}/device/nxp/common/security/testkey_rsa4096.pem
```

其对应的公钥为：

```
${MY_ANDROID}/device/nxp/common/security/testkey_public_rsa4096.bin.
```

用于对启动镜像签名的默认AVB启动私钥位于：

```
${MY_ANDROID}/external/avb/test/data/testkey_rsa2048.pem
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

该私钥可以使用OpenSSL生成。例如，以下命令可以生成RSA-4096私钥test_rsa4096_private.pem：

```
openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:4096 -outform PEM -out
 custom_rsa4096_private.pem
```

相应的公钥可以通过avbtool从私钥中提取。avbtool可以在${MY_ANDROID}/external/avb中找到。执行以下命令，从私钥中提取公钥：

```
avbtool extract_public_key --key custom_rsa4096_private.pem --output
 custom_rsa4096_public.bin
```

推荐使用SHA256_RSA4094和SHA256_RSA2048算法对i.MX 8Quad和i.MX 8M芯片的镜像进行签名，其加密加速和保证模块（CAAM）可以帮助加速计算哈希值。我们可以将其作为默认设置。

应使用自定义密钥对生产镜像进行签名，设置自定义AVB密钥对vbmeta镜像进行签名的示例如下：

```
diff --git a/imx8q/mek_8q/BoardConfig.mk b/imx8q/mek_8q/BoardConfig.mk
index 8e367bb..e1385f9 100644
--- a/imx8q/mek_8q/BoardConfig.mk
+++ b/imx8q/mek_8q/BoardConfig.mk
@@ -207,7 +207,7 @@ BOARD_AVB_ENABLE := true
ifeq ($(PRODUCT_IMX_CAR),true)
BOARD_AVB_ALGORITHM := SHA256_RSA4096
# The testkey_rsa4096.pem is copied from external/avb/test/data/
testkey_rsa4096.pem
-BOARD_AVB_KEY_PATH := device/nxp/common/security/testkey_rsa4096.pem
+BOARD_AVB_KEY_PATH := ${your-key-directory}/custom_rsa4096_private.pem
endif
TARGET_USES_MKE2FS := true
```

设置自定义AVB启动密钥对启动镜像进行签名的示例如下：

```
diff --git a/imx8q/mek_8q/BoardConfig.mk b/imx8q/mek_8q/BoardConfig.mk
index dbfb2821..744c2086 100644
--- a/imx8q/mek_8q/BoardConfig.mk
+++ b/imx8q/mek_8q/BoardConfig.mk
@@ -253,7 +253,7 @@ BOARD_AVB_ALGORITHM := SHA256_RSA4096
BOARD_AVB_KEY_PATH := device/nxp/common/security/testkey_rsa4096.pem
ifneq ($(PRODUCT_IMX_CAR),true)
-BOARD_AVB_BOOT_KEY_PATH := external/avb/test/data/testkey_rsa2048.pem
+BOARD_AVB_BOOT_KEY_PATH := ${your-key-directory}/custom_rsa2048_private.pem
BOARD_AVB_BOOT_ALGORITHM := SHA256_RSA2048
BOARD_AVB_BOOT_ROLLBACK_INDEX := 0
BOARD_AVB_BOOT_ROLLBACK_INDEX_LOCATION := 2
```

AVB在验证vbmeta镜像时检查签名和AVB公钥，因此启用Trusty时，AVB公钥必须存储在TEE支持的RPMB中。使电路板进入fastboot模式，并执行以下命令：

```
$ fastboot stage custom_rsa4096_public.bin
$ fastboot oem set-public-key
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

custom_rsa4096_public.bin是刚刚生成的AVB公钥。如果使用默认的AVB密钥进行调试目的，需要使用以下命令烧写默认公钥：

```
$ fastboot stage testkey_public_rsa4096.bin
$ fastboot oem set-public-key
```

3.3.4 绕过vbmeta/lock检查以进行开发

绕过vbmeta/lock检查非常方便开发工作。要在烧写完所有镜像后解锁芯片，需要启动电路板进入Android UI，在“设置”应用程序中启用“开发人员选项”，在“开发人员选项”下打开“OEM解锁”，然后重启电路板进入fastboot模式。执行以下命令：

```
$ sudo fastboot oem unlock
```

电路板解锁后，镜像可以使用fastboot命令烧写。要绕过vbmeta检查，需要使用fastboot通过--disable-verity选项烧写vbmeta镜像。以i.MX 8QuadMax MEK为例，执行以下命令：

```
$ sudo fastboot flash vbmeta_a vbmeta-imx8qm.img --disable-verity
$ sudo fastboot flash vbmeta_b vbmeta-imx8qm.img --disable-verity
```

3.3.5 更改镜像中的回滚索引值

此版本涉及3个回滚索引值：引导加载程序回滚索引、启动回滚索引和 vbmeta 回滚索引。引导加载程序回滚索引用于防止引导加载程序回滚攻击，仅在启用双引导加载程序功能时可用。启动回滚索引在启动镜像构建为链式分区时可用，仅标准Android平台支持该索引。vbmeta回滚索引在所有平台上都支持。默认情况下，各种回滚索引的初始值都为零。

当要发布一个新版本的镜像来修复上一版本中的bug时，先前的镜像可能会受到攻击，建议将回滚索引值比上一版本增加一。有关回滚索引使用的更多信息，请参见https://android.googlesource.com/platform/external/avb/+/master/README.md。

在此版本的i.MX Android平台上，对Android构建系统进行了一些修改。提供了一个名为imx-make.sh的shell脚本，以独立于Android镜像构建过程来构建U-Boot和内核代码。为了简化构建流程，由于所有镜像都可以只用一个命令构建，imx-make.sh在U-Boot和内核构建后开始构建Android镜像。在执行imx-make.sh脚本时，可以指定一个名为BOOTLOADER_RBINDEX的shell变量，以更改引导加载程序回滚索引，指定一个名为AVB_BOOT_RBINDEX的shell变量，用于更改启动回滚索引，还可以指定一个名为AVB_RBINDEX的shell变量，以更改vbmeta回滚索引。

例如，可以执行以下命令来更改回滚索引值。将${bootloader_rbindex}、${avb_rbindex}和${avb_boot_rbindex}更改为要设置的值：

```
BOOTLOADER_RBINDEX=${bootloader_rbindex} AVB_RBINDEX=${avb_rbindex}
 AVB_BOOT_RBINDEX=${avb_boot_rbindex} ./imx-make.sh -j4
```

如果U-Boot和内核已经构建且无需进行更新，可以使用make构建vbmeta和其它需要更新的相关镜像。按以下示例更改回滚索引值。将${avb_rbindex}和${avb_boot_rbindex}更改为要设置的值。

```
make -j4 AVB_RBINDEX=$(avb_rbindex) AVB_BOOT_RBINDEX=${avb_boot_rbindex}
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

3.3.6 烧写认证密钥

认证密钥在U-Boot中烧写。密钥库密钥认证旨在确定非对称密钥对是否由硬件支持，密钥有什么属性，以及有什么使用约束。

谷歌提供认证“密钥箱”，其中包含私钥（RSA和ECDSA）和相应的证书链，供Android合作伙伴前端（APFE）的合作伙伴使用。从谷歌检索到“密钥箱”后，对其进行解析，并将密钥和证书预处理到安全存储区。密钥和证书都应该使用可区分编码规则（DER）进行编码。

提供两种认证密钥和证书预处理方式：

- 直接以明文格式对密钥和证书进行预处理。
- 以AES-ECB加密格式对密钥和证书进行预处理，然后将其解密为明文，再写入安全存储区。

在预处理之前，请确保Trusty OS的安全存储区已正确初始化。

1. 以明文格式对密钥和证书进行预处理。
   提供fastboot命令，以明文格式将认证密钥和证书烧写到芯片中。这种方式更简单，但泄露密钥的风险也更大。启动电路板，进入fastboot模式，并使用以下命令。
   - 设置RSA私钥：

     ```
     $ fastboot stage ${path-to-rsa-private-key}
     $ fastboot oem set-rsa-atte-key
     ```

   - 设置ECDSA私钥：

     ```
     $ fastboot stage ${path-to-ecdsa-private-key}
     $ fastboot oem set-ec-atte-key
     ```

   - 附加RSA证书链：

     ```
     $ fastboot stage ${path-to-rsa-atte-cert}
     $ fastboot oem append-rsa-atte-cert
     ```

     第二个命令可能需要多次执行，以附加整个证书链。
   - 附加ECDSA证书链：

     ```
     $ fastboot stage < path-to-ecdsa-cert >
     $ fastboot oem append-ec-atte-cert
     ```

     第二个命令可能需要多次执行，以附加整个证书链。
2. 以AES-ECB加密格式对密钥和证书进行预处理。
   提供Fastboot命令，将AES-ECB编码密钥和证书烧写到芯片中。这样可以防止明文材料泄露。使用MPPUBK对密钥和证书进行加密，在将其写入安全存储区之前，在Trusty OS中将其解密为明文。这种方式仅能在HAB关闭的电路板上使用。工作流程如下。

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

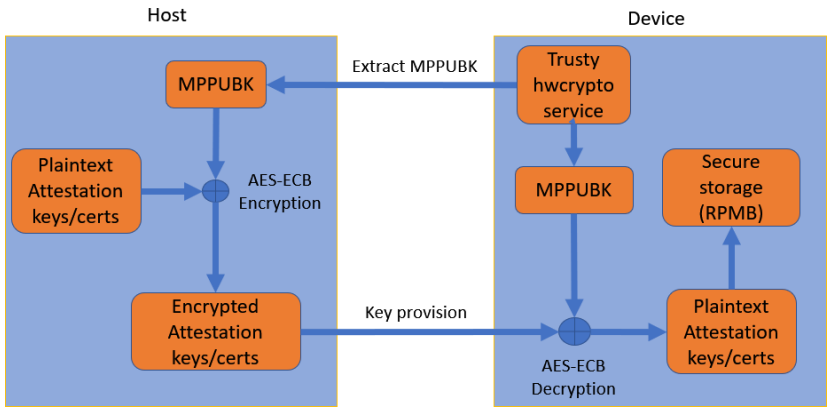


图12. 将AES-ECB编码密钥和证书烧写到芯片上

启动电路板，进入fastboot模式，并执行以下步骤：

a. 获取MPPUBK：

```
$ fastboot oem get-mppubk
$ fastboot get_staged mppubk.bin
```

b. 使用MPPUBK对明文认证密钥和证书进行加密。

认证密钥和证书应使用生成的MPPUBK以AES-ECB格式加密。以下是一个简单的加密python脚本gen_secure_atte.py：

```
from Crypto.Cipher import AES
import struct
import argparse
parser = argparse.ArgumentParser(description='Secure Provision encrypt
 tool.')
parser.add_argument('key', type=file)
parser.add_argument('plaintext', type=file)
parser.add_argument('blob', type=argparse.FileType('w'))
args = parser.parse_args()
data = args.plaintext.read()
data_len = len(data)
written_len  = struct.pack('I', data_len)
#AES need 16 bytes align, so pad the data it
data += '\0' * (((len(data)+15)/16 * 16) - len(data))
key2 = args.key.read()
key = key2[0:16]
magic = "!AT"
pad0 = struct.pack('B', 0)
cipher = AES.new(key, AES.MODE_ECB)
blob = cipher.encrypt(data)
args.blob.write(magic)
args.blob.write(pad0)
args.blob.write(written_len)
args.blob.write(blob)
#blob structure describe as below:
#{
#  char magic[4] = "!AT";
#  uint32_t len = plaintext_length
#  uint8 *encrypted_data
#}
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

在主机上对密钥和证书进行加密：

```
$ python gen_secure_atte.py mppubk.bin < path-to-plaintext-keys-or-
certificates > < encrypted-keys-or-certificates >
```

c. 设置加密RSA私钥：

```
$ fastboot stage ${path-to-encrypted-rsa-private-key}
$ fastboot oem set-rsa-atte-key-enc
```

d. 设置加密ECDSA私钥：

```
$ fastboot stage ${path-to-encrypted-ecdsa-private-key}
$ fastboot oem set-ec-atte-key-enc
```

e. 附加加密RSA证书链：

```
$ fastboot stage ${path-to-encrypted-rsa-atte-cert}
$ fastboot oem append-rsa-atte-cert-enc
```

第二个命令可能需要多次执行，以附加整个证书链。

f. 附加加密ECDSA证书链：

```
$ fastboot stage < path-to-encrypted-ecdsa-cert >
$ fastboot oem append-ec-atte-cert-enc
```

第二个命令可能需要多次执行，以附加整个证书链。

3.3.7 烧写硬件标识符

ID认证允许芯片提供其硬件标识符的证明。有关ID认证的更多信息，请参见 https://source.android.com/security/keystore/attestation。所有预处理的硬件标识符应与相关系统属性相匹配。下表列出了所有支持的标识符和系统属性的对应关系。

标识符	构建属性
ATTESTATION_ID_BRAND	ro.product.brand
ATTESTATION_ID_DEVICE	ro.product.device
ATTESTATION_ID_MANUFACTURER	ro.product.manufacturer
ATTESTATION_ID_MODEL	ro.product.model
ATTESTATION_ID_PRODUCT	ro.product.product.name
ATTESTATION_ID_SERIAL	ro.serialno

硬件标识符在引导加载程序中进行预处理，并且可以通过配置进行设置。以i.MX 8QM为例，其硬件标识符在{UBOOT_PATH}/configs/imx8qm_mek_android_trusty_defconfig中进行设置：

```
CONFIG_ATTESTATION_ID_BRAND="Android"
CONFIG_ATTESTATION_ID_DEVICE="mek_8q"
CONFIG_ATTESTATION_ID_PRODUCT="mek_8q"
CONFIG_ATTESTATION_ID_MANUFACTURER="nxp"
CONFIG_ATTESTATION_ID_MODEL="MEK-MX8Q"
```

请注意，每个芯片的序列号都是唯一的，因此其值在引导加载程序中自动检测。

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

根据需要更改标识符，然后烧写更新的引导加载程序镜像，并启动电路板进入fastboot模式，运行以下命令，对标识符进行预处理：

```
$ fastboot oem set-device-id
```

3.3.8 预处理Widevine L1密钥箱

Widevine是一种广泛使用的数字版权管理（DRM）技术，帮助避免媒体流被复制和重新分发。现在i.MX 8M Plus EVK评估板支持L1级Widevine。在使L1级Widevine运行之前，用户需要先对Widevine密钥箱进行预处理。

与认证密钥和证书一样，提供两种Widevine密钥箱预处理方式。一种是以明文格式对密钥箱二进制文件进行预处理，另一种是以AES-ECB加密格式对密钥箱二进制文件进行预处理，然后在写入安全存储区之前将其解密为明文。

- 以明文格式对密钥箱二进制文件进行预处理
  提供fastboot命令，以明文格式将密钥箱烧写到芯片中。这种方式更简单，但泄露密钥的风险也更大。
  启动电路板进入fastboot模式，并运行以下命令：

  ```
  $ fastboot stage ${path-to-keybox-binary}
  $ fastboot oem provision-wv-keybox
  ```

- 以AES-ECB加密格式对密钥箱二进制文件进行预处理
  提供fastboot命令，将AES-ECB编码密钥箱烧写到芯片中。这样可以防止明文材料被泄露。这种方式使用MPPUBK（制造保护公钥，由CAAM模块生成）对密钥箱进行加密，然后在将其写入安全存储区之前，在Trusty OS中将加密的密钥箱解密为明文。这种方式仅能在HAB关闭的电路板上使用（制造保护功能在i.MX 8ULP上尚未提供，因此此功能在i.MX 8UL上被禁用）。
  执行以下步骤，对密钥箱进行加密，并将其预处理到芯片中：

  1. 获取MPPUBK。

     ```
     $ fastboot oem get-mppubk
     $ fastboot get_staged mppubk.bin
     ```

  2. 使用MPPUBK对明文密钥箱进行加密。
     明文密钥箱二进制文件应使用生成的MPPUBK以AES-ECB格式加密。使用python脚本`gen_secure_atte.py`进行加密，如第3.3.6节所述。

     ```
     $ python gen_secure_atte.py mppubk.bin ${path-to-plaintext-keybox}
      ${encrypted-keybox}
     ```

  3. 对Widevine密钥箱进行预处理。

     ```
     $ fastboot stage ${path-to-encrypted-keybox}
     $ fastboot oem provision-wv-keybox-enc
     ```

3.3.9 更改存储锁定状态和/或回滚索引的方式

对于启用了TEE的镜像，锁定状态和回滚索引值存储在RPMB中。AVB的回滚索引值由TEE写入RPMB或从RPMB读取，但写入/读取流程由U-Boot发起。对于具有双引导加载程序功能的i.MX Android，有一个用于引导加载程序的回滚索引，该引导加载程序的回滚索引值由SPL写入RPMB或从RPMB读取。

回滚索引值和锁定状态可以供开发人员用于多种目的，不仅限于在i.MX Android代码中使用。此时，有必要了解锁定状态和回滚索引值是如何存储在电路板上的。

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

对于具有双引导加载程序功能的i.MX Android，引导加载程序的回滚索引值从RPMB读取，以便与引导加载程序镜像中的值进行比较。如果该回滚索引值大于存储在RPMB中的值且镜像被验证为可启动，则将引导加载程序镜像中的回滚索引值写入RPMB。此逻辑在以下函数中完成：

```
static int spl_verify_rbidx(struct mmc *mmc, AvbABSlotData *slot,
struct spl_image_info *spl_image)
```

在以下文件中：

```
${MY_ANDROID}/vendor/nxp-opensource/uboot-imx/lib/avb/fsl/fsl_avb_ab_flow.c
```

对于刚刚烧写了镜像的新电路板，在第一次启动时，将在以下函数中将默认的回滚索引值写入RPMB：

```
int rpmb_init(void)
```

在以下文件中：

```
${MY_ANDROID}/vendor/nxp-opensource/uboot-imx/lib/avb/fsl/fsl_avb_ab_flow.c
```

从上面列出的函数中可以看出，引导加载程序的回滚索引值通过一个`kblb_hdr_t type`结构变量定位。该结构有一个magic值。一个类型为`kblb_tag_t`的变量用于指定回滚索引值。

现在，在i.MX Android Auto中，引导加载程序的回滚索引值的偏移值由一个名为`BOOTLOADER_RBIDX_START`的宏控制，如分别用于i.MX 8QuadMax MEK和i.MX 8QuadXPlus MEK的两个文件中所定义。

```
${MY_ANDROID}/vendor/nxp-opensource/uboot-imx/include/configs/
imx8qm_mek_android_auto.h
${MY_ANDROID}/vendor/nxp-opensource/uboot-imx/include/configs/
imx8qxp_mek_android_auto.h
```

`BOOTLOADER_RBIDX_START`的值为0x3FF000，距离RPMB分区末端有4KB的偏移。

AVB的回滚索引值的读取流程由U-Boot在以下函数中发起：

```
FbLockState fastboot_get_lock_stat(void)
```

在以下文件中：

```
${MY_ANDROID}/vendor/nxp-opensource/uboot-imx/drivers/usb/gadget/
fastboot_lock_unlock.c
```

对于启用了TEE的镜像，此函数调用以下函数。它使用TIPC与TEE进行通讯，以获取该值。

```
int trusty_read_lock_state(uint8_t *lock_state)
```

AVB的回滚索引值的写入流程由U-Boot在以下函数中发起：

```
int fastboot_set_lock_stat(FbLockState lock)
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

在以下文件中：

```
${MY_ANDROID}/vendor/nxp-opensource/uboot-imx/drivers/usb/gadget/
fastboot_lock_unlock.c
```

对于启用了TEE的镜像，此函数调用以下函数。它使用TIPC与TEE进行通讯，以保存该值。

```
int trusty_write_lock_state(uint8_t lock_state)
```

读取AVB的回滚索引值，与vbmeta镜像中的值进行比较，如有必要，将vbmeta镜像中的值保存到RPMB中。此逻辑在以下函数中完成：

```
AvbABFlowResult avb_flow_dual_uboot(AvbABOps* ab_ops,
const char* const* requested_partitions,
AvbSlotVerifyFlags flags,
AvbHashtreeErrorMode hashtree_error_mode,
AvbSlotVerifyData** out_data)
```

在以下文件中：

```
${MY_ANDROID}/vendor/nxp-opensource/uboot-imx/lib/avb/fsl/fsl_avb_ab_flow.c
```

以下两个函数被调用，以读取和存储vbmeta的回滚索引：

```
AvbIOResult fsl_read_rollback_index_rpmb(AvbOps* ops, size_t rollback_index_slot,
uint64_t* out_rollback_index)
AvbIOResult fsl_write_rollback_index_rpmb(AvbOps* ops, size_t rollback_index_slot,
uint64_t rollback_index)
```

它们最终与TEE进行通讯，完成工作。

3.3.10 选择启动特定的插槽

在两个插槽都烧写了镜像的情况下，可以选择一个特定的插槽，以手动启动进行开发。启动电路板进入fastboot模式，并执行以下命令，从“插槽a”或“插槽b”启动：

```
$ sudo fastboot set_active a
$ sudo fastboot set_active b
```

3.3.11 禁用U-Boot中的开发选项

为了便于开发，在U-Boot中设置了一些开发选项，这可能会带来潜在的安全漏洞。在装运最终产品之前，这些选项必须被关闭。

- 启动延迟
  默认情况下，U-Boot保留2秒倒计时，以帮助开发人员在U-Boot停止并运行一些U-Boot命令。此功能可以通过将CONFIG_BOOTDELAY设置为**-2**来禁用。对于i.MX 8M Plus EVK，需要进行以下更改。在您正在使用的其它平台上也需要进行类似的更改。

```
diff --git a/configs/imx8mp_evk_android_trusty_defconfig b/configs/
imx8mp_evk_android_trusty_defconfig
index 80a4d45a5e..0ea9e3b9fc 100644
--- a/configs/imx8mp_evk_android_trusty_defconfig
+++ b/configs/imx8mp_evk_android_trusty_defconfig
@@ -185,3 +185,4 @@ CONFIG_ATTESTATION_ID_DEVICE="evk_8mp"
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

```
 CONFIG_ATTESTATION_ID_PRODUCT="evk_8mp"
 CONFIG_ATTESTATION_ID_MANUFACTURER="nxp"
 CONFIG_ATTESTATION_ID_MODEL="EVK_8MP"
+CONFIG_BOOTDELAY=-2
```

- Bootargs附加功能
  bootargs在开发过程中可能需要频繁更改。恩智浦U-Boot支持将U-Boot变量append_bootargs附加到默认bootargs，这将传递给内核。然而，此功能可能被黑客利用来攻击芯片，因此在任何正式发布中都应被禁用。要禁用bootargs附加功能，需要禁用CONFIG_APPEND_BOOTARGS。对于i.MX 8M Plus EVK，需要进行以下更改。在您正在使用的其它平台上也需要进行类似的更改。

```
diff --git a/configs/imx8mp_evk_android_trusty_defconfig b/configs/
imx8mp_evk_android_trusty_defconfig
index 80a4d45a5e..62ab4afaa0 100644
--- a/configs/imx8mp_evk_android_trusty_defconfig
+++ b/configs/imx8mp_evk_android_trusty_defconfig
@@ -171,7 +171,6 @@ CONFIG_CMD_BMP=y
 CONFIG_LZ4=y
 CONFIG_FLASH_MCUFIRMWARE_SUPPORT=y
-CONFIG_APPEND_BOOTARGS=y
 CONFIG_SPL_MMC_SUPPORT=y
 CONFIG_AVB_WARNING_LOGO=y
 CONFIG_AVB_WARNING_LOGO_COLS=0x320
```

- 禁用制造生产公钥提取
  制造生产公钥在某些情况下可以用作加密/解密密钥，泄露密钥可能会泄露其它重要材料。 “制造生产公钥提取”功能必须被禁用，以便传输镜像。对于i.MX 8M Plus EVK，可以进行以下更改。在您正在使用的其它平台上也需要进行类似的更改。

```
diff --git a/configs/imx8mp_evk_android_trusty_defconfig b/configs/
imx8mp_evk_android_trusty_defconfig
index 80a4d45a5e..61be3f5ebe 100644
--- a/configs/imx8mp_evk_android_trusty_defconfig
+++ b/configs/imx8mp_evk_android_trusty_defconfig
@@ -185,3 +185,4 @@ CONFIG_ATTESTATION_ID_DEVICE="evk_8mp"
 CONFIG_ATTESTATION_ID_PRODUCT="evk_8mp"
 CONFIG_ATTESTATION_ID_MANUFACTURER="nxp"
 CONFIG_ATTESTATION_ID_MODEL="EVK_8MP"
+CONFIG_GENERATE_MPPUBK=n
```

3.3.12安全解锁

安全解锁旨在防止未经授权的解锁。它需要解锁凭证。解锁操作只有在提供有效的解锁凭证后才能进行。

提供了一个示例，该示例生成带有序列号和MPPUBK（制造保护公钥，由CAAM模块生成）的解锁凭证。要启用安全解锁功能，需要启用CONFIG_SECURE_UNLOCK。对于i.MX 8M Plus EVK，需进行以下更改。在您正在使用的其它平台上也需要进行类似的更改：

```
diff --git a/configs/imx8mp_evk_android_trusty_defconfig b/configs/
imx8mp_evk_android_trusty_defconfig
index 80a4d45a5e..b3da5c6ea3 100644
--- a/configs/imx8mp_evk_android_trusty_defconfig
+++ b/configs/imx8mp_evk_android_trusty_defconfig
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

```
@@ -185,3 +185,4 @@ CONFIG_ATTESTATION_ID_DEVICE="evk_8mp"
 CONFIG_ATTESTATION_ID_PRODUCT="evk_8mp"
 CONFIG_ATTESTATION_ID_MANUFACTURER="nxp"
 CONFIG_ATTESTATION_ID_MODEL="EVK_8MP"
+CONFIG_SECURE_UNLOCK=y
```

执行以下步骤，以验证安全解锁功能。这些操作只能在HAB关闭的电路板上执行（制造保护功能在i.MX 8ULP上尚未提供，安全解锁功能需要MPPUBK支持，因此此功能在i.MX 8UL上被禁用）：

1. 获取MPPUBK。

```
$ fastboot oem get-mppubk
$ fastboot get_staged mppubk.bin
```

2. 获取序列号。

```
$ fastboot oem get-serial-number
$ fastboot get_staged serial.bin
```

3. 生成解锁凭证。在主机上使用`mppubk.bin`对`serial.bin`进行加密。有关加密脚本，请参阅第3.3.6节。

```
$ python gen_secure_atte.py mppubk.bin serial.bin serial-enc.bin
```

4. 验证安全解锁功能。

```
$ fastboot stage serial-enc.bin
$ fastboot oem unlock
```

如果芯片已处于解锁状态，可能需要先锁定芯片。

3.3.13安全固件加载器

在i.MX 8QuadMax和i.MX 95上，Widevine L1已启用。因此，为了保护VPU固件，安全固件加载器被启用。要保护的固件使用AES对称密钥加密，然后使用OPENSSL生成的非对称密钥进行签名。当需要加载该固件时，将签名固件发送到Trusty OS，由固件加载器可信应用程序进行验证和解密。

默认情况下，用于对固件进行加密的密钥位于：

```
${MY_ANDROID}/device/nxp/common/security/firmware_encrypt_key.bin
```

用于对固件进行签名的私钥位于：

```
 ${MY_ANDROID}/device/nxp/common/security/firmware_private_key.der
```

RPMB分区中的公钥用于验证固件，它位于：

```
 ${MY_ANDROID}/device/nxp/common/security/firmware_public_key.der
```

这些密钥也已复制到`$OUT/firmware_test_keys/`。用户可以根据需要替换它们。

上述3个密钥可以通过位于以下位置的示例脚本直接生成：

```
${MY_ANDROID}/device/nxp/common/tools/generate_key.sh
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

这些密钥可以参考以下命令生成：

```
./${MY_ANDROID}/device/nxp/common/tools/generate_key.sh firmware_private_key.der
 firmware_public_key.der firmware_encrypt_key.bin
```

为了保护解密和签名验证所需的密钥，需要将公钥和加密密钥烧写到RPMB分区，然后Trusty OS通过安全存储区从RPMB分区获得这两个密钥。

使电路板进入fastboot模式，然后执行以下命令，以烧写解密密钥：

```
$ fastboot stage firmware_encrypt_key.bin
$ fastboot oem provision-firmware-encrypt-key
```

使电路板进入fastboot模式，然后执行以下命令，以烧写验证密钥：

```
$ fastboot stage firmware_public_key.der
$ fastboot oem provision-firmware-sign-key
```

当`imx-make.sh`用于构建内核模块时，`package_tool`用于对指定的VPU固件进行加密和签名。当构建完成时，`vpu_fw_imx8_dec.bin.signed`在`${MY_ANDROID}/vendor/nxp/linux-firmware-imx/firmware/vpu/`目录下被创建。

3.3.14 设置加密启动OTA

加密启动OTA允许在HAB关闭的芯片上提供启用加密启动来升级镜像。

3.3.14.1 预处理DEK blob

在加密过程中获得的DEK blob文件在升级过程中是必不可少的。将它们另存为`dek_blob_spl.bin`和`dek_blob_bl.bin`。以下命令用于将DEK blobs预处理到安全存储区中。

```
$ fastboot stage dek_blob_spl.bin
$ fastboot oem provision-spl-dek-blob
$ fastboot stage dek_blob_bl.bin
$ fastboot oem provision-bootloader-dek-blob
```

3.3.14.2 设置bootargs，启用加密启动OTA

加密启动OTA默认情况下不启用。可以添加bootargs来启用它。按如下方式设置芯片bootargs：

```
=> setenv append_bootargs androidboot.encrypted_boot_ota=true
=> saveenv
```

如果设置了此`bootargs`，后安装脚本会加载加密引导加载程序并自动获得dek blob，然后将其插入到正确的位置。

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

3.4 Linux/Android平台中的安全功能配置

3.4.1 DM-Verity与vbmeta的关系

Device Mapper verity（DM-verity）内核功能支持对块的透明完整性检查。此功能可帮助Android用户确保在启动设备时，设备处于与被烧写时相同的状态。`vbmeta`镜像包含一个用于为`system.img`设置DM-verity的内核命令行描述符，以及用于`system.img`和`vendor.img`的哈希树描述符。`vbmeta`镜像中的哈希树描述符包含根哈希、盐值和哈希树的偏移值，这些对于为“系统”和“供应商”分区进行DM-verity检查至关重要。

当为“系统”和“供应商”分区启用DM-verity时，任何破坏`system.img`、`vendor.img`和`vbmeta.img`一致性的操作都会导致DM-verity检查失败，从而导致系统启动失败。

3.4.2 Trusty OS Linux驱动程序配置

Trusty OS支持将日志输出到UART或TIPC日志通道。Trusty OS Linux驱动程序支持通过TIPC通道传输来自Trusty OS的日志。默认情况下，此功能在参考镜像中被启用。

在Trusty OS Linux驱动程序trusty-log中，当此功能被启用时，Trusty OS会关闭UART输出日志端口。Trusty OS中的UART驱动程序同步输出字符，这会花费大量的IO时间。

trusty-log驱动程序在设备树中配置，如下所示：

```
trusty-log {
compatible = "android,trusty-log-v1";
};
```

3.4.3 基于Trusty的keymaster、gatekeeper和安全存储代理

基于Trusty的keymaster HAL是一个动态可加载的库，被密钥库服务用来提供硬件支持的加密服务。它不在用户空间甚至内核空间提供任何敏感操作。所有敏感操作都委托给Trusty OS（安全环境）中的keymaster TA。关系如下图所示。

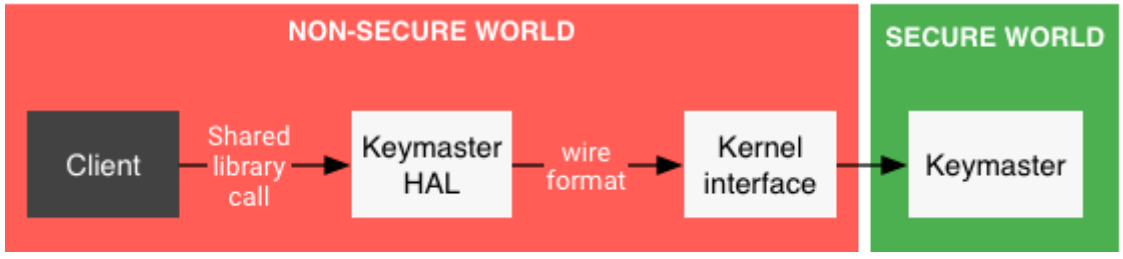


图13. keymaster HAL和keymaster TA之间的关系

Trusty支持的keymaster HAL 3.0是为Android Pie 9或以后版本设计的，不能用于Android Oreo 8.1。Android Oreo 8.1运行Trusty支持的keymaster HAL 2.0。

Gatekeeper子系统执行设备模式/密码身份验证。它通过带有密钥的HMAC注册和验证密码。此外，Gatekeeper限制连续验证尝试失败次数，并根据给定的超时和连续尝试失败次数拒绝服务请求。Trusty支持的gatekeeper将所有关键操作发送给Trusty中的gatekeeper TA。

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

安全存储代理在Linux端运行，与Trusty中的存储TA进行通讯，以执行安全存储区读/写操作，例如，从eMMC器件的RPMB分区读取数据或向其写入数据。

Trusty支持的keymaster、gatekeeper和安全存储代理都依赖于安全存储区，而安全存储区只有Trusty可以访问，但用户可能不想正确设置安全存储区（如RPMB的密钥），因为在某些情况下，安全性并不那么重要，甚至可以忽略。在这种情况下，keymaster和gatekeeper都会退回到软件支持版本，并且由内核命令行中的`androidboot.keystore`变量选择。

当Trusty及其相关的可信应用程序（如keymaster TA和storage TA）正确初始化后，U-Boot将`androidboot.keystore`设置为`trusty`，否则设置为`software`，然后通过内核命令行将其传递给内核。`androidboot.keystore`被转换为`ro.boot.keystore` Android属性，然后初始化程序选择keymaster和gatekeeper版本（基于Trusty或软件），并根据此属性启动安全存储代理。下图所示为工作流程。

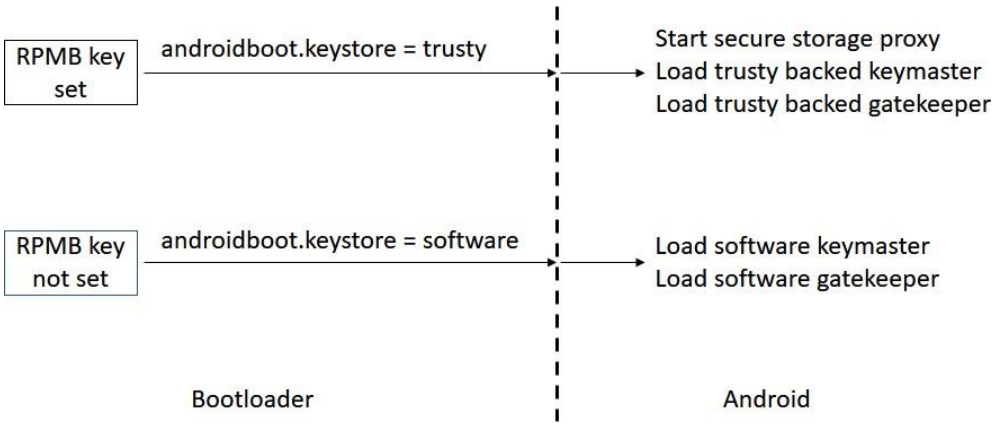


图14. TEE和TA初始化与工作流程

3.4.4 禁用GPU平面映射

当GPU平面映射被启用，黑客可能会窥探特权存储区。GPU平面映射在默认情况下是启用的，但用户可以通过添加bootargs来禁用它，如下所示：

```
galcore.baseAddress=<dram-base-address> galcore.physSize=0
 galcore.mmuException=0
```

不同平台上的`dram-base-address`各不相同。该值应根据每个平台确定。以i.MX 8M Plus为例。用户可以通过在`{MY_ANDROID}/imx8m/evk_8mp/BoardConfig.mk`中添加以下bootargs来禁用GPU平面映射：

```
diff --git a/imx8m/evk_8mp/BoardConfig.mk b/imx8m/evk_8mp/BoardConfig.mk
index 4732374b..83089631 100644
--- a/imx8m/evk_8mp/BoardConfig.mk
+++ b/imx8m/evk_8mp/BoardConfig.mk
@@ -119,6 +119,9 @@ BOARD_KERNEL_CMDLINE := init=/init
 androidboot.console=ttymxc1 androidboot.hardw
BOARD_KERNEL_CMDLINE += transparent_hugepage=never
BOARD_KERNEL_CMDLINE += swiotlb=65536
+# disable GPU flat mapping
+BOARD_KERNEL_CMDLINE += galcore.baseAddress=0x40000000 galcore.physSize=0
 galcore.mmuException=0
```


本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

```
+
# display config
BOARD_KERNEL_CMDLINE += androidboot.lcd_density=240
 androidboot.primary_display=imx-drm
```

4 关于本文中源代码的说明

本文中所示的示例代码具有以下版权和BSD-3-Clause许可：

2024年恩智浦版权所有；在满足以下条件的情况下，可以源代码和二进制文件的形式重新分发和使用本源代码（无论是否经过修改）：

1. 重新分发源代码必须保留上述版权声明、这些条件和以下免责声明。
2. 以二进制文件形式重新分发时，必须在文档和/或随分发提供的其他材料中复制上述版权声明、这些条件和以下免责声明。
3. 未经事先书面许可，不得使用版权所有者的姓名或参与者的姓名为本软件的衍生产品进行背书或推广。

本软件由版权所有者和参与者“按原样”提供，不承担任何明示或暗示的担保责任，包括但不限于对适销性和特定用途适用性的暗示保证。在任何情况下，无论因何种原因或根据何种法律条例，版权所有者或参与者均不对因使用本软件而导致的任何直接、间接、偶然、特殊、惩戒性或后果性损害（包括但不限于采购替代商品或服务；使用损失、数据损失或利润损失或业务中断）承担责任，无论是因合同、严格责任还是侵权行为（包括疏忽或其他原因）造成的，即使事先被告知有此类损害的可能性也不例外。

5 修订历史

修订历史

文档ID	发布日期	说明
UG10158 v.android-14.0.0_2.2.0	2024年10月18日	i.MX 8M Mini、i.MX 8M Nano、i.MX 8M Plus、i.MX 8M Quad、i.MX 8ULP、i.MX 8QuadMax、i.MX 8QuadXPlus GA 版本、i.MX 95 (A1 15x15) Alpha版本和i.MX 95 (A1 19x19) Beta版本。
UG10158 v.android-14.0.0_2.0.0	2024年8月9日	i.MX 8M Mini、i.MX 8M Nano、i.MX 8M Plus、i.MX 8M Quad、i.MX 8ULP、i.MX 8QuadMax、i.MX 8QuadXPlus GA版本和i.MX 95 Alpha版本。 更新了文档ID。
ASUG v.android-14.0.0_1.2.0	2024年4月19日	i.MX 8ULP EVK、i.MX 8M Mini、i.MX 8M Nano、i.MX 8M Plus、i.MX 8M Quad、i.MX 8QuadMax和i.MX 8Quad XPlus GA版本。
ASUG v.android-14.0.0_1.0.0	2024年2月6日	i.MX 8ULP EVK、i.MX 8M Mini、i.MX 8M Nano、i.MX 8M Plus、i.MX 8M Quad、i.MX 8QuadMax和i.MX 8Quad XPlus GA版本。
ASUG v.android-13.0.0_2.2.0	2023年10月24日	i.MX 8ULP EVK、i.MX 8M Mini、i.MX 8M Nano、i.MX 8M Plus、i.MX 8M Quad、i.MX 8QuadMax和i.MX 8Quad XPlus GA版本。

© 2024 NXP B.V. 版权所有。

修订历史（续）

文档ID	发布日期	说明
ASUG v.android-13.0.0_2.0.0	2023年7月	i.MX 8ULP EVK Beta版本、i.MX 8M Mini、i.MX 8M Nano、i.MX 8M Plus、i.MX 8M Quad、i.MX 8QuadMax和i.MX 8 QuadXPlus GA版本。
ASUG v.android-13.0.0_1.2.0	2023年3月	i.MX 8ULP EVK Beta版本、i.MX 8M Mini、i.MX 8M Nano、i.MX 8M Plus、i.MX 8M Quad、i.MX 8QuadMax和i.MX 8 QuadXPlus GA版本。
ASUG v.android-13.0.0_1.0.0	2023年1月	i.MX 8ULP EVK Beta版本、i.MX 8M Mini、i.MX 8M Nano、i.MX 8M Plus、i.MX 8M Quad、i.MX 8QuadMax和i.MX 8 QuadXPlus GA版本。
ASUG v.android-12.1.0_1.0.0	2022年10月	i.MX 8ULP EVK Beta版本、i.MX 8M Mini、i.MX 8M Nano、i.MX 8M Plus、i.MX 8M Quad、i.MX 8QuadMax和 i.MX 8 QuadXPlus GA版本。
ASUG v.android-12.0.0_2.0.0	2022年7月	i.MX 8ULP EVK Beta版本、i.MX 8M Mini、i.MX 8M Nano、i.MX 8M Plus和i.MX 8M Quad GA版本。
ASUG v.android-12.0.0_1.0.0	2022年3月	i.MX 8ULP EVK Beta版本、i.MX 8M Mini、i.MX 8M Nano、i.MX 8M Plus和i.MX 8M Quad GA版本。
ASUG v.android-11.0.0_2.6.0	2022年1月	更正了第3.3.2节中的一个拼写错误。
ASUG v.android-11.0.0_2.6.0	2022年1月	i.MX 8ULP EVK Beta版本、i.MX 8M Mini、i.MX 8M Nano、i.MX 8M Plus和i.MX 8M Quad GA版本。
ASUG v.android-11.0.0_2.4.0	2021年10月	i.MX 8ULP EVK Alpha版本、i.MX 8M Mini、i.MX 8M Nano、i.MX 8M Plus和i.MX 8M Quad GA版本。
ASUG v.android-11.0.0_2.2.0	2021年7月	i.MX 8M Mini、i.MX 8M Nano、i.MX 8M Plus和i.MX 8M Quad GA版本。
ASUG v.android-11.0.0_2.0.0	2021年4月	i.MX 8M Mini、i.MX 8M Nano、i.MX 8M Plus和i.MX 8M Quad GA版本。
ASUG v.android-11.0.0_1.0.0	2020年12月	i.MX 8M Plus EVK Beta版本，以及所有其它i.MX 8 GA版本。
ASUG v.android-10.0.0_2.3.0	2020年7月	i.MX 8M Plus EVK Beta1版本，以及所有其它i.MX 8 GA版本。
ASUG v.android-10.0.0_2.0.0	2020年5月	i.MX 8M Mini、i.MX 8M Nano、i.MX 8M Quad、i.MX 8Quad Max和i.MX 8QuadXPlus GA版本。
ASUG v.android-10.0.0_2.1.0	2020年4月	i.MX 8M Plus Alpha和i.MX 8QuadXPlus Beta版本。
ASUG v.android-10.0.0_1.0.0	2020年3月	删除了Android 10镜像。
ASUG v.android-10.0.0_1.0.0	2020年2月	i.MX 8M Mini、i.MX 8M Quad、i.MX 8QuadMax和i.MX 8 QuadXPlus GA版本。
ASUG v.P9.0.0_2.0.0-ga	2019年8月	更新了SCFW移植工具包的位置。
ASUG v.P9.0.0_2.0.0-ga	2019年4月	i.MX 8M、i.MX 8QuadMax、i.MX 8QuadXPlus GA版本。
ASUG v.P9.0.0_1.0.0-ga	2019年1月	i.MX 8M、i.MX 8QuadMax、i.MX 8QuadXPlus GA版本。
ASUG v.P9.0.0_1.0.0-beta	2018年11月	首次发布

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com.cn/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

目录

本文件中提供的所有信息均受法律免责声明的约束。

© 2024 NXP B.V. 版权所有。

恩智浦半导体

UG10158

i.MX Android安全用户指南

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.

© 2024 NXP B.V. All rights reserved.

For more information, please visit: https://www.nxp.com.cn Document feedback

Date of release: 18 October 2024

Document identifier: UG10158