D1-H Tina Linux 安全启动
开发指南

版本号: 1.0
发布日期: 2021.04.08

版本历史

版本号	日期	制/修订人	内容描述
1.0	2021.04.08	AWA0916	初始版本

目　录

插　图

1 概述

1.1 编写目的

介绍 D1-H TinaLinux 下防刷机功能的使用。

1.2 适用范围

适用于基于硬件平台：全志 D1-H 芯片。

软件平台：Tina v3.5 及其后续版本。

1.3 相关人员

适用于 TinaLinux 平台的客户及相关技术人员。

2 安全基础

2.1 密码学基础介绍

2.1.1 数据加密模型

（1）明文 P。准备加密的文本，称为明文。

（2）密文 Y。加密后的文本，称为密文。

（3）加解密算法 E(D)。用于实现从明文到密文或从密文到明文的一种转换关系。

（4）密钥 K。密钥是加密和解密算法中的关键参数。

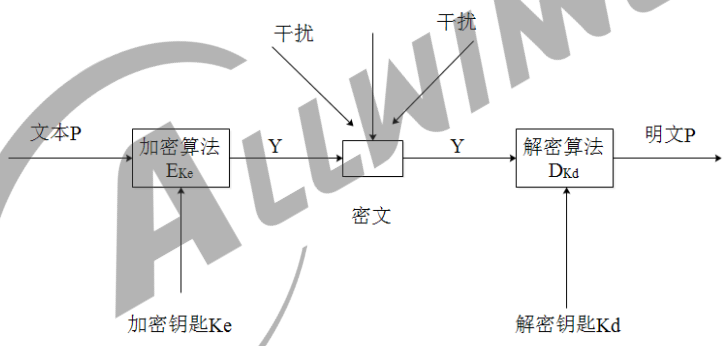

图 2-1：数据加密模型

2.1.2 加密算法

对称加密算法：加密、解密用的是同一个密钥。比如 AES 算法。

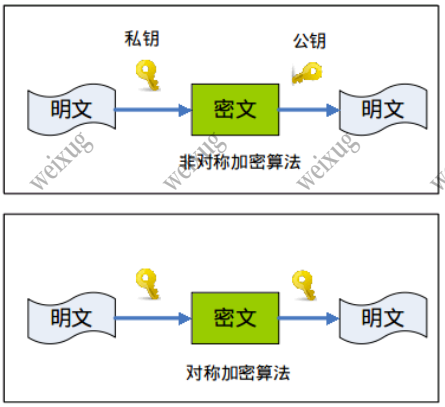

图 2-2: 对称/非对称加密算法

非对称加密算法：加密、解密用的是不同的密钥，一个密钥公开，即公钥，另一个密钥持有，即私钥。其中一把用于加密，另一把用于解密。比如 RSA 算法。

散列（hash）算法：一种摘要算法，把一笔任意长度的数据通过计算得到固定长度的输出，但不能通过这个输出得到原始计算的数据。

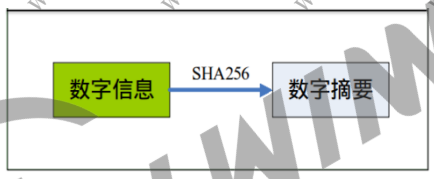

图 2-3: SHA256 算法

2.1.3 签名与证书

数字签名：数字签名是非对称密钥加密技术与数字摘要技术的应用。数字签名保证信息是由签名者自己签名发送的，签名者不能否认或难以否认；可保证信息自签发后到收到为止未曾作过任何修改，签发的文件是真实文件。

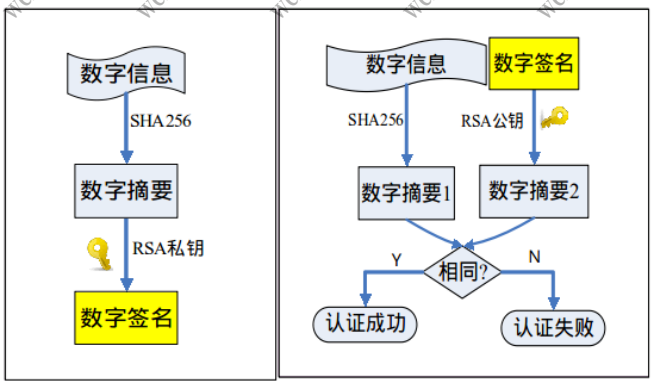

图 2-4: 数字签名与认证

数字证书：是一个经证书授权中心数字签名的包含公开密钥拥有者信息以及公开密钥的文件，是一种权威性的电子文档。

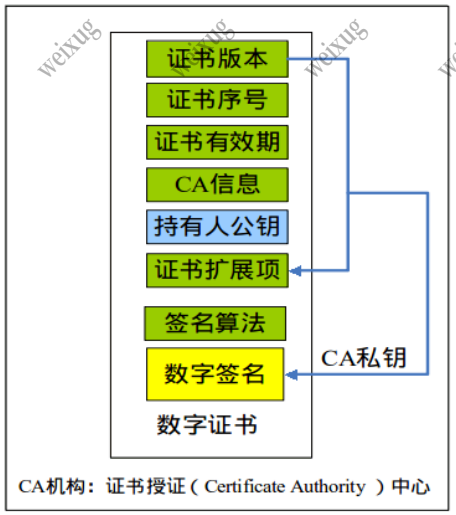

图 2-5: 数字证书

3 Secure Boot

Secure Boot，即安全启动，这里指的是防刷机功能。通常来说，Secure Boot 从 brom 执行开始，到 Linux 启动结束。Secure Boot 主要设计目的：

- 建立完整的安全信任链，确保启动阶段加载的各种镜像是可信的。
- 相关 key 的烧写。
- 安全固件版本管理。

3.1 安全启动原理

Tina D1-H 安全启动基于私钥签名-公钥验签的业界公认非对称算法实现完整的安全启动方案，具体来说，选择的是 RSA2048-SHA256。

先使用私钥给固件进行签名生成安全固件，再将根密钥公钥的 SHA256 值（即 rotpk.bin）烧写至芯片中 efuse 特定区域。启动时，固化在芯片的 brom 程序首先会读取 efuse 中的 rotpk 值，将该值与保存在 flash 上的根证书中公钥进行 SHA256 运算后的值进行比对，验证根证书中公钥的可信任性。然后会使用 flash 上存储的证书链中的一系列公钥来对各个子镜像进行逐级安全校验。验证启动顺序为 brom->sboot->opensbi->uboot->kernel。efuse 的不可更改性确保了证书链的可信任，整个流程的设计确保了整个 Linux 方案的安全启动。

3.2 生成安全固件

Tina SDK 已经将安全固件制作流程中密钥的生成和必要的签名过程集成在打包脚本内部，所以安全固件的编译及打包流程与非安全固件的几乎一致，只是在最后的打包的时候有差异。非安全固件的打包可参考用户《TinaLinux SDK 开发指南》文档，安全固件的打包步骤如下：

```
$ source build/envsetup.sh
==> 设置环境变量。
$ lunch
==> 选择方案。
$ make [-jN]
==> 编译，-jN 参数选择并行编译进程数量。
$ ./scripts/createkeys
==> 生成一组用于签名的密钥，不需要每次执行，详见3.2.2小节。生成的密钥路径位于out/{BOARD}/keys/。
$ pack -s [-d]
==> 打包固件。-s 表示制作安全固件；-d 表示生成的固件包串口信息转到tf卡座输出（可选）。
```

后续几个小节将对安全固件生成过程中一些注意事项进行说明。

3.2.1 安全固件配置

在执行 make 进行编译前，请确保包含如下配置。

3.2.1.1 内核镜像格式配置

执行 make menuconfig，确保如下选项配置正确。

```
Tina Configuration
└─> Target Images
    └─> [ ] Build filesystem for Boot (SD Card) partition
    └─> Boot (SD Card) Kernel format (boot.img)
    └─> [ ] Build filesystem for Boot-Recovery initramfs partition
    └─> Boot-Recovery initramfs Kernel format (boot.img)
```

3.2.2 签名密钥

⚠ 警告

客户在首次进行安全固件打包之前，必须运行一次./scripts/createkeys 创建自己的签名密钥，并将创建的秘钥妥善保存。每次执行 createkeys 后都会生成新的密钥，因此不用每次都执行，除非需要更换密钥。

```
[key_rsa]
key=Trustkey
key=NOTWORLD_KEY
key=PRIMARY_DEBUG_KEY
key=SCPFirmwareContentCertPK
key=SecondaryDebugCertPK
key=SoCFirmwareContentCert_KEY
key=TrustedFirmwareContentCertPK
key=TWORLD_KEY
key=NonTrustedFirmwareContentCertPK

[toc0]
;item=Item_TOC_name,            Item_filename,                    Key_Name
item=toc0,                                    sboot.bin,                Trustkey

[toc1]
;item=Item_TOC_name,     Item_filename,     Key_Name
rootkey=rootkey,         rootkey.der,              Trustkey
item=opensbi,              opensbi.fex,                SCPFirmwareContentCertPK
onlykey=boot,            boot.fex,                 SCPFirmwareContentCertPK
item=u-boot,             u-boot.fex,               NonTrustedFirmwareContentCertPK
onlydata=dtb,                sunxi.fex             NULL
```

createkeys根据[key_rsa]生成各密钥对

TOC0的内容，对于item，使用Key_Name对Item_filename进行签名，比如这里表示用Trustkey对sboot.bin进行签名。

TOC1的内容，其中onlykey表示仅包含证书，不含镜像，onlydata表示不签名，仅含镜像。

图 3-1: dragon-toc 配置文件说明

createkeys 脚本会根据device/config/chips/{IC}/对应目录下的dragon_toc*.cfg生成一组用于签名的

密钥，生成的密钥保存在out/{BOARD}/keys/目录下。执行 pack -s 时，会使用这些密钥分别对相应的镜像进行签名并生成证书。

典型的drangon_toc*.cfg文件内容如上图所示。createkeys 依据 [key_rsa] 下的 key-value 生成密钥对。打包过程中会将 sboot.bin 封装成 toc0.fex，将 opensbi/uboot/dtb 等封装成 toc1.fex。

请将生成的密钥保存到自己的私密目录，其中 Trustkey.bin，Trustkey.pem 与 rotpk.bin 三个文件为根密钥相关文件，要重点保护。

Trustkey.bin 与 Trustkey.pem 是根密钥私钥，不能泄漏和丢失。丢失与泄露会导致一系列问题，比如：生成的安全固件无法在芯片上启动、失去防刷机功能等。

3.2.3 安全固件版本管理

注：pack -s 打包完成后，生成的安全固件位于out/{BOARD}/keys/目录下，文件名为tina_{BOARD}_<uart0/card0>_secure_v[NUM].img。相对于非安全固件，文件名字多了 secure 关键字。

文件名字的 NUM 为固件版本号，由device/config/chips/{IC}/configs/default/version_base.mk或者target/allwinner/generic/version/version_base.mk文件决定，前者优先级更高。版本号的作用是防止固件回退，具体实现是：在设备启动过程中会比较当前 flash 上固件版本与 efuse 中版本信息，如果 efuse 中版本信息更高，启动失败；如果 flash 上固件的版本更高，将此版本信息写入 efuse 中，继续启动；如果版本信息一致，正常启动。

说明

最多支持更新 *32* 个版本。

3.3 开启安全启动

完全开启安全启动**共需三个前提**：

1. 烧写 efuse 中的 anti-brush bit。
2. 烧写 rotpk.bin 到 efuse 中 rotpk 区域。
3. 烧写安全固件到 flash 中。

⚠ 警告

- **不同的 IC，efuse 大小不同。efuse 的硬件特性决定了 efuse 中每个 bit 仅能烧写一次。此外，efuse 中会划分出很多区域，大部分区域也只能烧写一次。详细请参考芯片 SID 规范。**
- **烧写 anti-brush bit 操作是不可逆的，一旦烧写，后续将只能启动安全固件，启动不了非安全固件。**
- **默认情况下，通过 LiveSuit/PhoenixSuit 烧写安全固件完成时会自动烧写 anti-brush bit。**
- **anti-brush bit 与 rotpk.bin 都烧写后，设备就只能启动与 rotpk.bin 对应密钥签名的安全固件；如果只烧写 anti-brush bit，没有烧写 rotpk.bin，此设备上烧写的任何安全固件都可以启动。调试时可只烧写 anti-brush bit，但是设备出厂前必须要烧写 rotpk.bin。**

如何判断 anti-brush bit 是否烧写?

- 因为只有 anti-brush bit 烧写后才能启动安全固件，所以如果是安全启动，anti-brush bit 就一定烧写了。安全启动过程中有一些特有的打印，如 “sboot commit...” 、 “OLD version:... ” 、 “NEW version: ...” 、 “secure enable bit: 1” 等等，可用来进行判断。

如何判断 rotpk.bin 是否烧写?

- 反证法。烧录使用其他 key 签名的安全固件（安全版本号一致），如果不能启动，则表明已经烧写 rotpk。
- sboot 启动 log 中如果有 “have rotpk, do check”，表明 rotpk.bin 已经烧写。

各镜像在哪个阶段进行校验?

- 默认配置情况下，会对 sboot、opensbi、u-boot、kernel 进行校验。
- brom 运行阶段对 sboot 进行校验。如果 sboot 能启动，表明已经经过了校验。
- sboot 运行阶段对 opensbi 与 u-boot 进行校验。
- uboot 运行阶段会对 kernel 进行校验。

3.4 烧写 rotpk.bin 与 anti-brush bit

主要包含两个步骤：

1. 使用 LiveSuit/PhoenixSuit 烧写安全固件，安全固件烧写完毕时自动烧写 efuse 中的 anti-brush bit 位。
2. 使用 DragonSN 工具将 rotpk.bin 烧写到设备的 efuse 中。

说明

***DragonSN** 是 **AW** 开发的 **PC** 端烧 **key**（**SN** 号、**MAC** 地址、**rotpk** 等）工具，可以将 **key** 烧录到 **private** 分区或 **efuse** 等存储空间中，当前仅支持在 **windows** 上运行。**DragonSN** 与设备之间通过 **USB** 通信，控制设备烧录配置好的 **key** 信息。更详细的信息可参考 **DragonSN** 工具包中的使用说明。*

3.4.1 DragonSN 烧写 rotpk.bin 步骤

DragonSN 烧 rotpk.bin 具体步骤如下：

- 设置 burn_key 属性为 1。只有 burn_key 的值为 1，设备才会接收 DragonSN 通过 usb 传过来的信息，进行烧录动作。该属性位于`device/config/chips/{IC}/configs/{BOARD}/sys_config.fex`文件中 [target] 项下，如下图所示。如果未显式配置，按照 burn_key=0 来处理。

```
[target]
boot_clock      = 1008
storage_type    = 5
burn_key        = 1
```

图 3-2: burnkey 配置

- 打包安全固件，烧写到 flash 中。
- 在 PC 端对 DragonSN 工具进行配置。打开 DragonSNConfig.exe，如下图所示，点击 “添加”，在 “类型” 一栏下拉菜单中选择 rotpk，点击 “保存”、“确定”。点击 “全局配置”, 设置 “烧写模式” 为 “安全 key”。配置完成后，关闭配置工具。

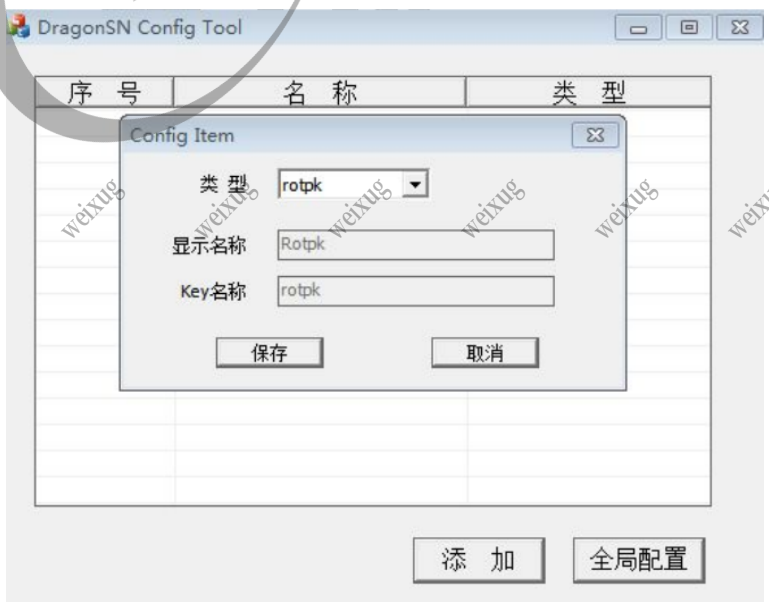

图 3-3: rotpk 烧写配置

- 运行 DragonSN.exe 工具，配置 rotpk.bin 所在的路径。然后将设备通过 usb 与 PC 连接，重启设备。当 DragonSN 提示框显示设备已连接后，开始烧录。为了保证不会烧录错误的 rotpk.bin，在烧录过程中，设备端会将 PC 端下发的 rotpk.bin 与当前 flash 上安全固件中根证书公钥的 SHA256 值进行对比，匹配成功后才烧录该 rotpk.bin。

⚠ 警告

开启 burn_key 之后，每次启动时 uboot 或尝试与 DrangonSN 工具握手，会导致启动时间变长。解决办法有两个，一是烧完 rotpk 之后，重新烧录一份不含 burn_key 的固件；二是开启 private 分区，将 DrangonSN 工具全局配置中 ``设置写标志'' 置为 1，这样烧 rotpk 完成后，将该标志写入 private 分区，同时设备将关闭烧号通道，后续启动将不会尝试与 DragonSN 工具握手。

3.5 校验 rootfs

3.1 节中提到，Secure Boot 从 brom 执行开始，到 Linux 启动结束。但是 rootfs 没有进行校验，为了校验 rootfs 的完整性，将 Secure Boot 延展至 rootfs，Tina 引入一种方法：uboot 校验 rootfs。

⚠ 警告

注 1：rootfs 必须为只读才能进行校验。
注 2：rootfs 类型必须是 squashfs。

3.5.1 uboot 校验 rootfs

由于 rootfs 通常来说较大，从 flash 中读取以及校验时间都比较长。Tina 上提供了一种在 uboot 阶段校验 squashfs rootfs 的方法，可以提取部分 rootfs 的数据来进行校验，有效减少校验时间。

3.5.1.1 uboot 校验 squashfs rootfs 功能实现

主要思路是：

- 使用 extract_suqashfs 工具对 squashfs rootfs 进行采样，具体为每 1M 取前面 rootfs_per_MB 字节的数据，最后不足 1M 的不采样。rootfs_per_MB 在`device/config/chips/{IC}/configs/{BOARD}/env*.cfg`中设置，必须为 4096 的倍数或者 full，其中 full 表示对整个 rootfs 进行校验；如未设置，默认取 4096 字节。
- 将所有采集的数据组合成新的文件，对该文件进行签名，生成证书。
- 使用 update_squashfs 工具将证书附着在 squashfs rootfs 的结尾处。

具体来说，使用 extract_squashfs 将 out/{BOARD}/image/rootfs.fex 进行采样，获取文件 out/{BOARD}/image/rootfs-extract.fex。使用秘钥 SCPFirmwareContentCertPK 对该 rootfs-extract.fex 进行签名，生成证书 out/{BOARD}/image/toc1/cert/rootfs.der。然后使用工具 update_squashfs 将该 rootfs.der 证书附着在 out/{BOARD}/image/rootfs.fex 的结尾处。启动过程，在 uboot 中按照相反的步骤对 rootfs 进行校验。

以上操作都是在打包脚本 scripts/pack_img.sh 中实现。

3.5.1.2 uboot 校验 squashfs rootfs 开启

首先，执行 make menuconfig，打开 CONFIG_USE_UBOOT_VERIFY_SQUASHFS 选项。

```
Tina Configuration
└─> Global build settings  --->
    └─> [*] Verify squashfs rootfs in uboot
```

其次，确保 uboot 文件`lichee/brandy-2.0/u-boot-2018/configs/sun20iw1p1_defconfig`中包含 CONFIG_SUNXI_PART_VERIFY=y。

使能该功能后，在启动过程中，uboot 会出现类似如下的 log。

```
[06.420]find rootfs key stored in root certif
partition rootfs verify pass
```

3.6 安全启动代价

3.6.1 启动时间增加

安全启动过程中会逐级对下一阶段运行的镜像进行校验，会增加启动时间。相对于非安全启动，整体会增长 500ms 左右（不包括 rootfs 的校验）。实际增加时间会因存储介质、硬件 CE 版本、cpu/dram 频率等因素的影响而不同。

3.6.2 ota 升级的变化

由 3.2 小节可知，安全固件封包与非安全固件有一定的差异，因此在 ota 升级时，请确保使用正确的文件。

- 升级 opensbi/uboot/dts/sys_config，需要使用 tina/out/{BOARD}/image/toc1.fex 文件；
- 升级 sboot，需要使用 tina/out/{BOARD}/image/toc0.fex 文件；

- 升级 linux kernel，需要使用 tina/out/{BOARD}/image/boot.fex 文件；
- 升级 rootfs，需要使用 tina/out/{BOARD}/image/rootfs.fex 文件。

4 量产工具

从整个安全启动的角度看，需要一整套工具来配合完成对应的工作。

4.1 RSA 密钥对生成工具

目前，有公开的密钥对生成工具 openssl，可以生成足够长度的密钥对。

Tina 开发平台 scripts 下提供了一个生成密钥对的脚本 createkeys，该脚本调用 dragonsecboot 工具，解析 dragon_toc*.cfg 中 [key_rsa] 字段，并基于字段的内容生成对应名字的密钥对。

4.2 安全固件版本管理

安全固件打包时会解析`device/config/chips/{IC}/configs/default/version_base.mk`文件或者`target/allwinner/generic/version/version_base.mk`文件决定，前者优先级更高，并基于其中的内容生成对应版本的固件。

在 efuse 中会有一块区域用来记录固件版本。

当设备启动时，会将 efuse 中记录的版本号同固件中的版本号比较，如果固件中的版本较低，则不能继续启动；如果固件中的版本比较高，将固件中的版本写入 efuse，继续启动；如果版本相同，正常启动。可防止固件版本回退。

4.3 数据封包工具

Tina 开发平台中提供固件封包工具 dragonsecboot，在安全固件打包过程中会对相关的镜像文件（sboot、uboot、kernel 等）进行签名，并生成证书以及相关信息，以便启动时对这些镜像文件进行校验，验证完整性。

4.4 烧 key 工具

烧 key 工具用来将 rotpk.bin 烧写到设备的 efuse 中，efuse 位于 IC 内部，由于 efuse 中内容一旦写入便不可更改，所以从根源上保证了根证书公钥 hash 的安全性。

可用的烧 key 工具包含 DragonKey 或者 DragonSN，工具的使用说明位于工具包中。

4.5 关闭 jtag

将 sys_config.fex 中 jtag_para 节下的 jtag_enable 设置为 0 即可关闭 jtag 调试功能。

4.6 密钥说明

4.6.1 固件签名密钥

密钥	安全固件签名私钥
功能	RSA2048 类型私钥。对 sboot、opensbi、uboot、boot、rootfs 等分区进行签名
SDK 路径	`tina/out/${BOARD}/keys/*.pem`与`tina/out/${BOARD}/keys/*.bin`，除开 rotpk.bin
设备位置	设备上不保存
烧写方式	不烧写
保密	是

密钥	安全固件签名公钥
功能	RSA2048 类型公钥。对 sboot、opensbi、uboot、boot、rootfs 等分区进行验签
SDK 路径	位于`tina/out/${BOARD}/image/toc0`以及`tina/out/${BOARD}/image/toc1`目录下的证书中
设备位置	flash 上 TOC0、TOC1、boot、rootfs 等分区中
烧写方式	随固件一起烧写
保密	否

4.6.2 efuse 中密钥

密钥	rotpk
功能	签名根密钥公钥的 sha256 值，用于安全启动中根证书的校验。长度 256bit。
SDK 路径	`tina/out/${BOARD}/keys/rotpk.bin`
设备位置	IC 中 efuse 中的 rotpk 区域
烧写方式	DragonSN 等，参考 3.4 小节
保密	否

著作权声明

商标声明

免责声明