

君正®

## SecurityBoot 工具说明文档

---

Date: Dec. 2023



北京君正集成电路股份有限公司  
Ingenic Semiconductor Co., Ltd.

**Copyright © 2005-2023 Ingenic Semiconductor Co. Ltd. All rights reserved.**

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Ingenic Semiconductor Co. Ltd.

## Trademarks and Permissions



、Ingenic and Ingenic icons are trademarks of Ingenic Semiconductor Co.Ltd. All other trademarks and trade names mentioned in this document are the property of their respective holders.

## Disclaimer

All the deliverables and data in this folder serve only as a reference for customer development. Please read through this disclaimer carefully before you use the deliverables and data in this folder. You may use the deliverables in this folder or not. However, by using the deliverables and data in this folder, you agree to accept all the content in this disclaimer unconditional and irrevocable. If you do not find the content in this disclaimer reasonable, you shall not use the deliverables and data in this folder.

The deliverables and data in this folder are provided "AS IS" without representations, guarantees or warranties of any kind (either express or implied). To the maximum extent permitted by law, Ingenic Semiconductor Co., Ltd (Ingenic) provides the deliverables and data in this folder without implied representations, guarantees or warranties, including but not limited to implied representations, guarantees and warranties of merchantability, non-infringement, or fitness for a particular purpose. Deviation of the data provided in this folder may exist under different test environments.

Ingenic takes no liability or legal responsibility for any design and development error, incident, negligence, infringement, and loss (including but not limited to any direct, indirect, consequential, or incidental loss) caused by the use of data in this folder. Users shall be responsible for all risks and consequences caused by the use of data in this folder.

Ingenic Semiconductor Co., Ltd.

Add: Ingenic Headquarters, East Bldg. 14, Courtyard #10, Xibeiwang East Road, Haidian Dist., Beijing, China

Tel: (86-10)56345000

Fax: (86-10)56345001

<http://www.ingenic.cn>

## 目录

1. Security Boot 概述.....	4
1.1 算法介绍.....	4
1.2 工具介绍.....	4
2. 功能简介.....	5
2.1 Security Boot 功能.....	5
2.2 Chip-Key 重加密功能.....	5
3. 功能配置使用及 Security Boot 流程.....	6
3.1 Security 功能配置.....	6
3.2 Security Boot 使用流程.....	7
4. 工具介绍.....	7
4.1 Key 生成工具.....	7
4.2 签名工具.....	8
4.2.1 burner bin.....	9
4.2.2 spl bin.....	10
4.2.3 uboot_with_spl bin.....	11
4.2.4 kernel.....	12
4.2.5 other bin.....	13
4.3 烧录工具.....	14
4.3.1 安全烧录流程.....	14
4.3.2 安全烧录配置.....	14
5. 安全建议.....	17

## 版本历史

日期	版本	描述
Feb, 2023	2.0	烧录工具界面更新，发布第二版
Dec, 2020	1.0	发布第一版

## 1. Security Boot 概述

Security 主要为了保护客户厂商的软件程序不被其对手复制，包括整体抄板或者抄袭某个应用程序。与此同时，还要允许终端客户进行软件升级或重新烧录软件程序。

防止代码复制是通过对代码进行加密的方式实现，可以使用 AES、2Key-3DES 加密。密码可以是用户提供的 User-Key0、User-Key1 或芯片内部的 Chip-Key（随机数密码）。

防止加密的代码被解密，我们将密码存放在芯片的 OTP 中，只有芯片中的 SC-ROM 代码可以访问。可以通过调用 SC-ROM 接口使用密码进行加解密，而软件不直接访问密码本身，防止密码泄露。

### 1.1 算法介绍

RSA 算法：

该算法为非对称算法，有一对密钥，分别是公钥和私钥。使用其中一种密钥进行加密，必须使用另外一种密钥进行解密。该算法可以使用在对数据的签名和加密，对数据进行加密时，加密的数据长度不能大于密钥的长度。

AES 算法：

该算法为对称算法，对代码数据段和公钥数据段进行加密，保证数据不可见。

HASH 算法：

该算法计算后的数据不能恢复原数据。使用该算法对代码段数据和公钥段数据进行哈希运算，用于检验数据的一致性和完整性。

### 1.2 工具介绍

在使用 Security Boot 功能时，需要用到以下的工具：

秘钥工具：

用于生成 RSA-Key、User-Key0 和 User-Key1，提供给签名工具、烧录工具和 SC-ROM 使用。

签名工具：

使用生成的 RSA 私钥、User-Key0、User-Key1 对数据进行加密和签名，并将加密信息填充到文件起始的 2048 字节中。

烧录工具：

烧录 RSA 公钥的 HASH 值、User-Key0、User-Key1 和加密后的软件程序。

## 2. 功能简介

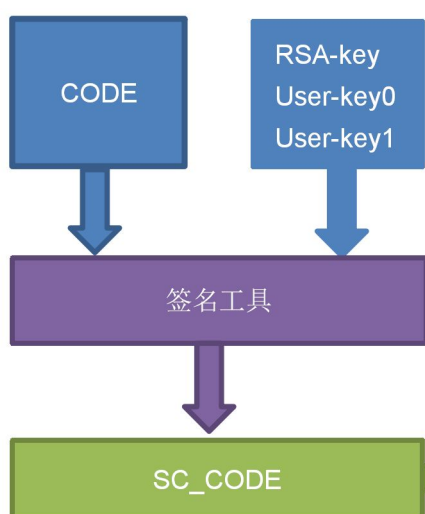
### 2.1 Security Boot 功能

激活安全启动功能须配置 EFUSE 状态寄存器中 SCBOOT\_EN 位。安全启动功能使用在 BOOT-ROM 启动 SPL、SPL 启动 BootLoader、BootLoader 启动 kernel 过程中。

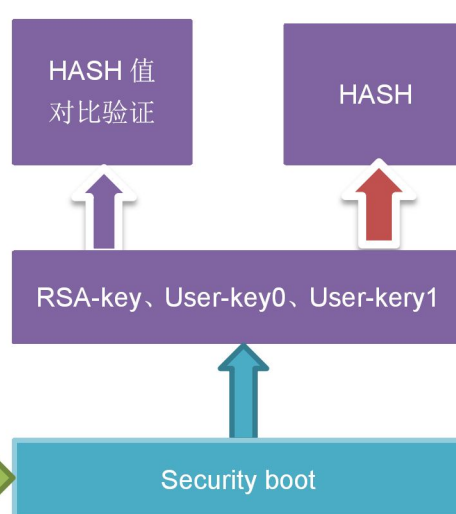


BOOT-ROM 的代码是集成在芯片中的是不可更改的被认为是最安全的我们的开发板内部存放了 AES-Key 和解密数字签名需要的 RSA-Key。

签名流程:



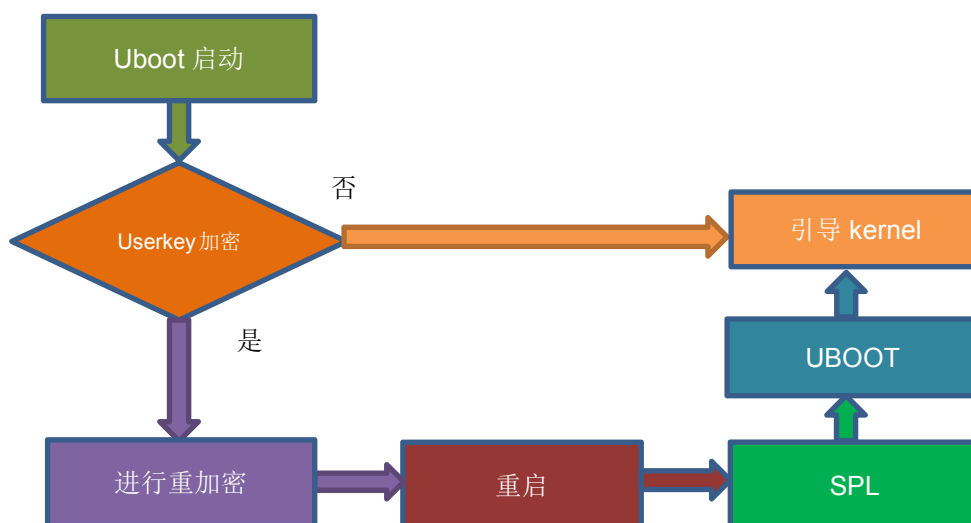
解密流程:



### 2.2 Chip-Key 重加密功能

该模块目前 X2000 只添加了 MMC 的 Chip-Key 重加密。

该模块依赖于 Security Boot 模块。使用 Chip-Key 重加密 SPL 和 UBOOT 的后比 User-Key 加密被破解的机率低，更大提升系统的安全性。



### 3. 功能配置使用及 Security Boot 流程

#### 3.1 Security 功能配置

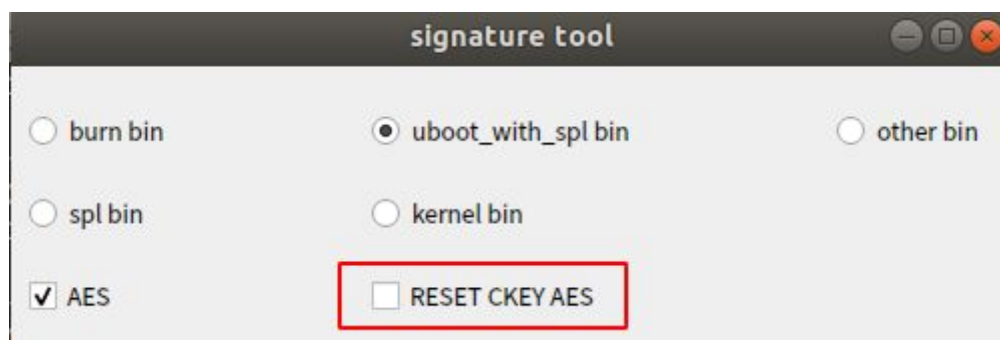
启用安全启动功能需要修改 UBOOT 板级配置文件，路径如下：

```
u-boot/include/configs/{board}.h
```

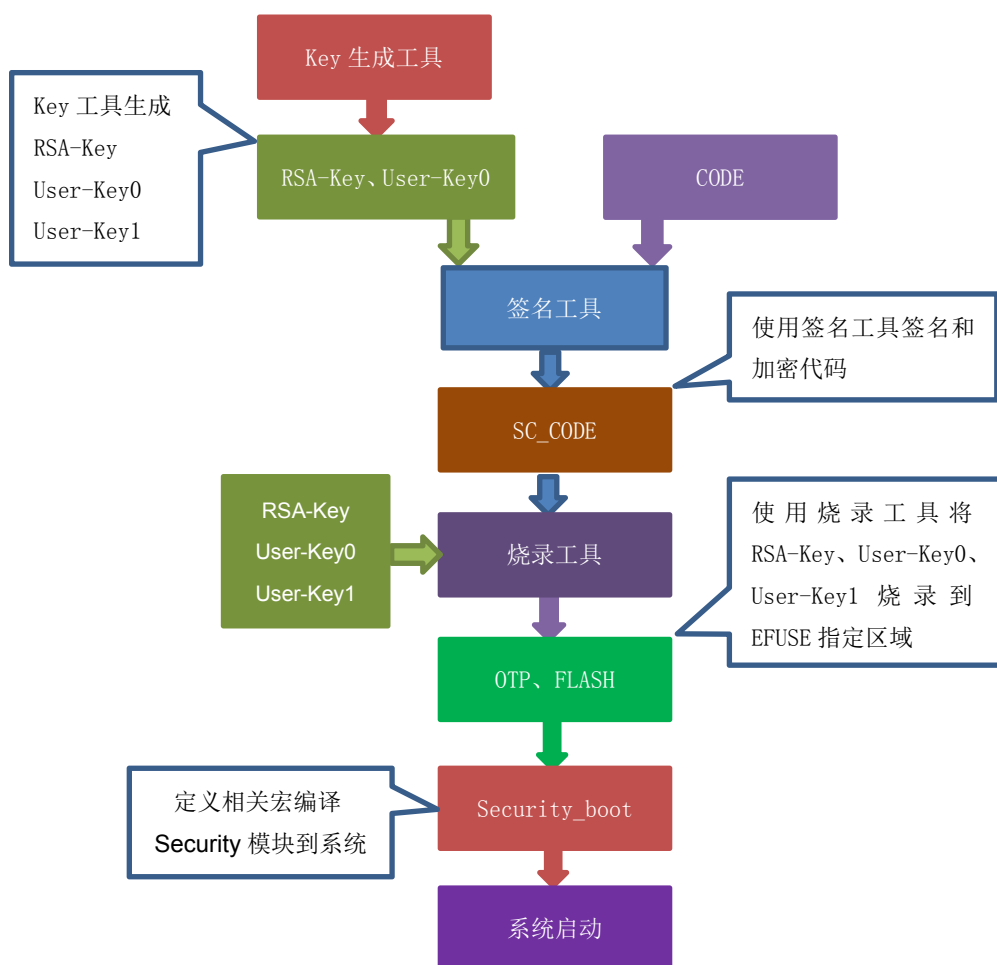
添加如下配置：

```
#define CONFIG_JZ_SCBOOT
#define CONFIG_JZ_SECURE_SUPPORT
#define CONFIG_JZ_CKEYAES /* optional */
```

如果启用 Chip-Key 重加密模块需要在签名工具签名时，勾选重加密单选框。



## 3.2 Security Boot 使用流程



## 4. 工具介绍

Security Boot 需要使用到 Key 生成工具、签名工具和烧录工具。

烧录工具路径：

```

Manhattan 工程目录/prebuilts/burnertools/
├── cloner-x.x.x-ubuntu_alpha.tar.gz
├── cloner-x.x.x-windows_alpha.zip

```

注意：

1. 由于迭代更新，版本号可能有区别。
2. Key 生成工具和签名工具可在烧录工具目录 securitytool 下找到。

### 4.1 Key 生成工具

用于生成 RAS-Key、User-Key0 和 User-Key1，提供给签名工具和烧录工具使用。

Key 生成工具路径：

```

cloner-x.x.x-ubuntu_alpha/securitytool/x2000/keytool/
├── genkey_32
├── genkey_64

```

注意：genkey\_64 代表 64 位，genkey\_32 代表 32 位。

运行 Key 生成工具，界面显示如下：



生成 Key 操作步骤：

- 1) 点击“Path”按钮，选择密钥生成目录
- 2) 点击“Generate Key”按钮，生成 RSA-Key、User-Key0 和 User-Key 1

生成如下三个文件：

- key.bin: 包含 RSA-Key、User-Key0 和 User-Key1 提供烧录工具使用。
- pri\_key.pem: RSA 一对密钥，长度为 64 个 Word。
- n: User Key0、User Key1，长度共 8 个 Word。

生成 Key 后需要把 key.bin 复制到烧录工具的 security 目录下：

```
cloner-x.x.x-ubuntu_alpha/security/x2000/key.bin
```

## 4.2 签名工具

用于签名、加密客户的固件、应用程序、配置文件，以便保护用户核心数据。

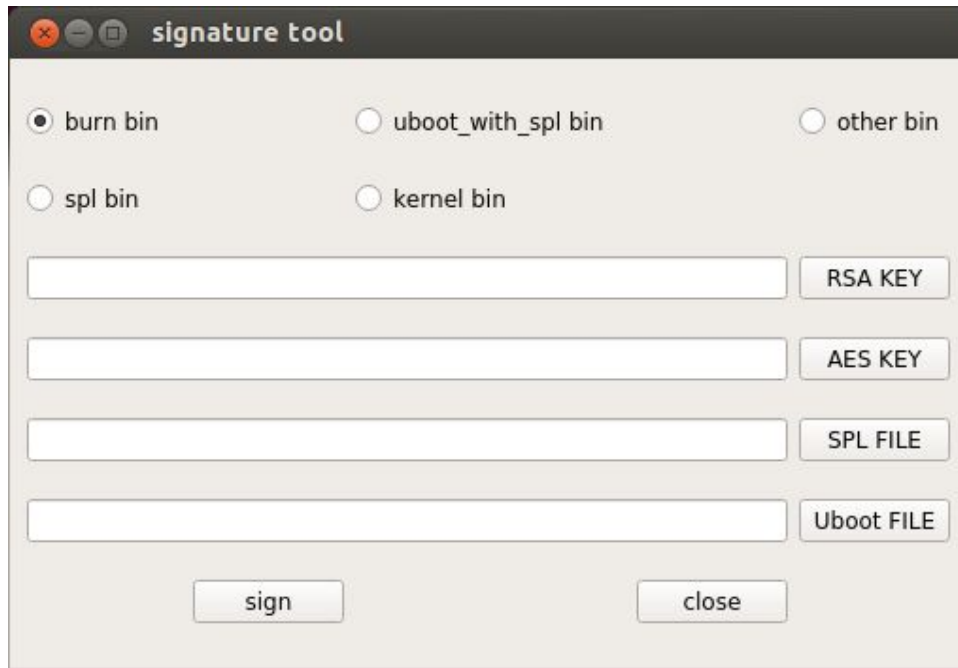
签名工具路径：

```
cloner-x.x.x-ubuntu_alpha/securitytool/x2000/sigtool/  
├── sigtool_32  
├── sigtool_64  
└── sigtool-bin.exe
```

注：sigtool\_64 代表 64 位，sigtool\_32 代表 32 位。

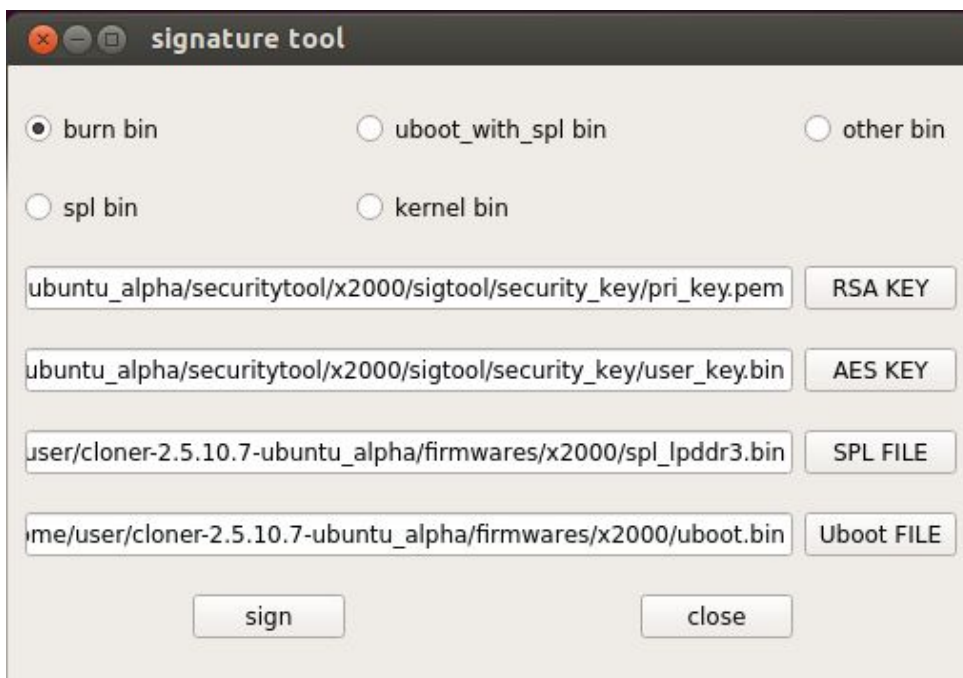
运行签名工具，界面显示如下：





#### 4.2.1 burner bin

用于烧录工具第一阶段及第二阶段的固件加密。



签名烧录固件操作步骤:

- 1) 点击“RSA KEY”按钮，选择 RSA-私钥(pri\_key.pem 文件，签名使用)。
- 2) 点击“AES KEY”按钮，选择 User-Key(user\_key.bin 文件，加密使用)。
- 3) 点击“SPL FILE”按钮，选择烧录第一阶段固件，路径如下：

```
cloner-x.x.x-ubuntu_alpha/firewares/x2000/spl_lpddr{num}.bin
```

根据芯片 ddr 类型选择 spl\_lpddr3.bin 或 spl\_lpddr2.bin 文件，签名成功后会在源文件的路径下生成 spl\_sec.bin 加密后固件。

4) 点击“UBOOT FILE”按钮，选择烧录第二阶段固件，路径如下：

```
cloner-x.x.x-ubuntu_alpha/firewares/x2000/uboot.bin
```

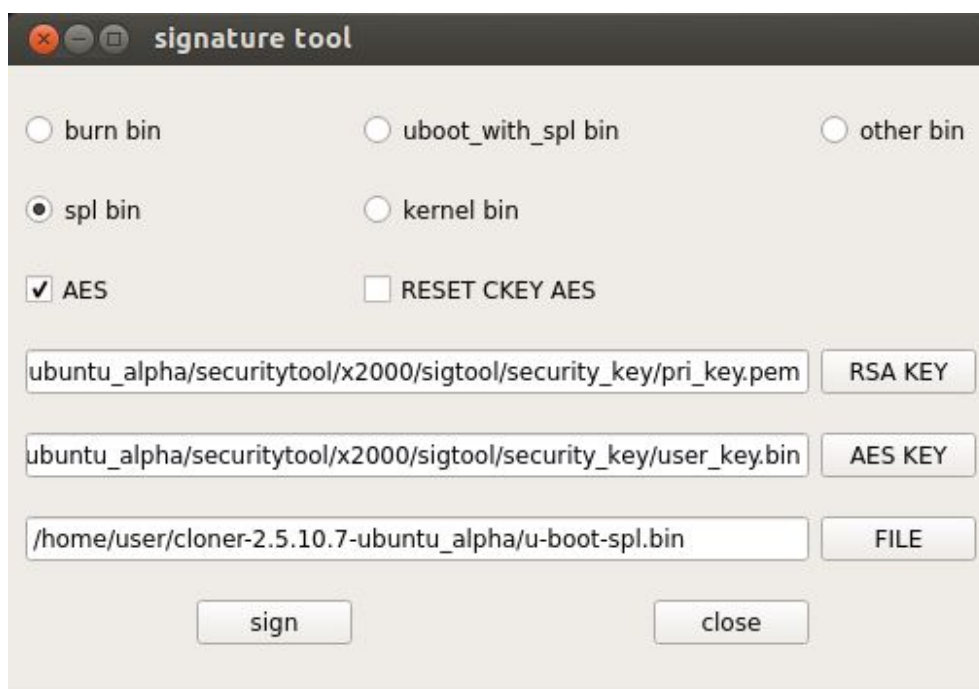
签名成功后会在源文件的路径下生成 uboot\_sec.bin 加密后固件。

烧录固件签名完成后在烧录工具的固件路径下文件结构：

```
cloner-x.x.x-ubuntu_alpha/firmwares/x2000/
├── config.cfg
├── spl.bin
├── spl_sec.bin
├── uboot.bin
└── uboot_sec.bin
```

#### 4.2.2 spl bin

签名和加密单个 SPL 文件

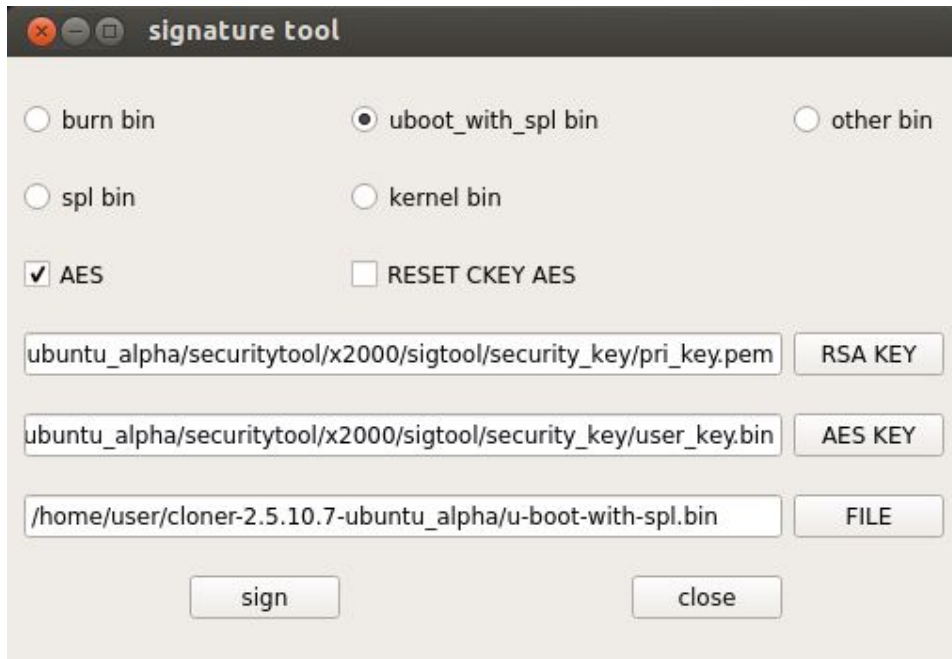


签名单个 SPL 文件操作步骤：

- 1) “AES” 单选框选中则加密，否则不加密。
- 2) 如果 AES 被选中，点击“AES KEY”按钮，选择 User-Key (user\_key.bin 文件)。
- 3) “RESET CKEY AES” 单选框选中则 Chip-Key 重加密，否则不重加密。
- 4) 点击“RSA KEY”按钮，选择 RAS-私钥 (pri\_key.pem 文件)。
- 5) 点击“FILE”按钮，选择要签名的 SPL 的文件。

### 4.2.3 uboot\_with\_spl bin

签名和加密 SPL 和 UBOOT 合并固件。



签名 SPL 和 UBOOT 合并固件操作步骤：

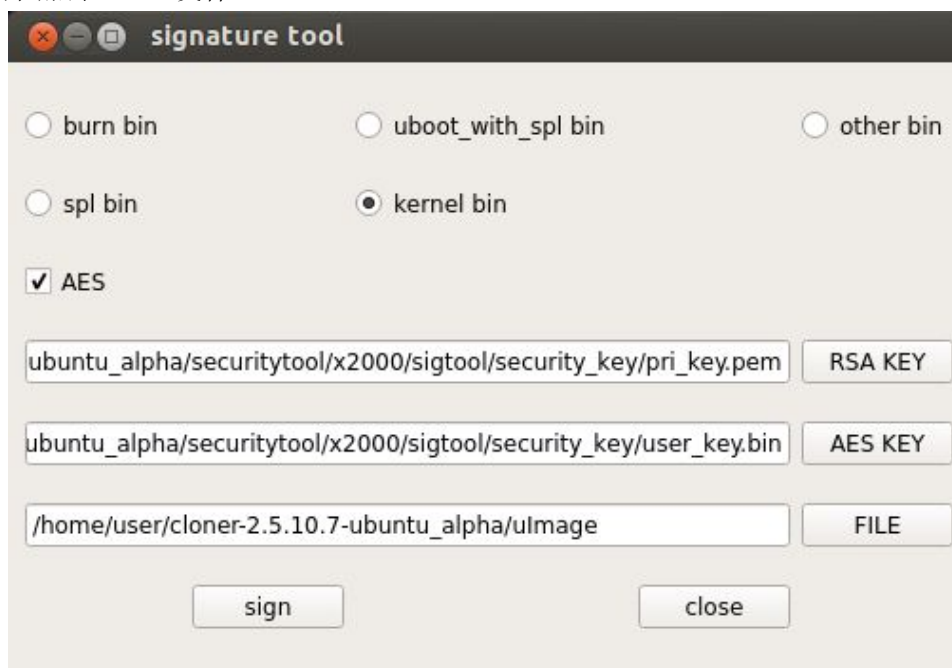
- 1) “AES” 单选框选中则加密，否则不加密。
- 2) 如果 AES 被选中，点击“AES KEY”按钮，选择 User-Key (user\_key.bin 文件)。
- 3) “RESET CKEY AES” 单选框选中则 Chip-Key 重加密，否则不重加密。
- 4) 点击“RSA KEY”按钮，选择 RAS-私钥 (pri\_key.pem 文件)。
- 5) 点击“FILE”按钮，选择要签名的 u-boot-with-spl 的固件，生成对应的-dst.bin 文件

注：Manhattan 工程不同的编译方式，生成 UBOOT 名称不同，例如：

- a. 编译整体工程时，选择 out/product/{board}/image/下生成的 uboot 文件。
- b. 单独编译 u-boot 时，选择 u-boot 路径下的 u-boot-with-spl-mbr-gpt.bin 或 u-boot-with-spl.bin 文件。

#### 4.2.4 kernel

签名和加密 kernel 文件。

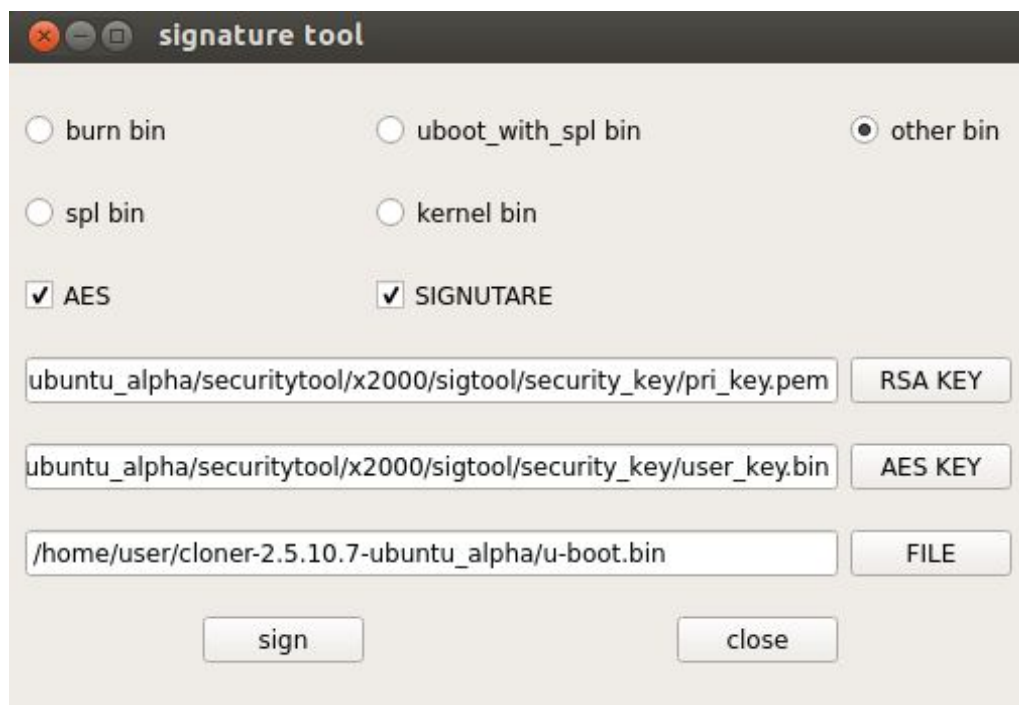


签名 Kernel 文件操作步骤:

- 1) “AES” 单选框选中则加密，否则不加密。
- 2) 如果 AES 被选中，点击 “AES KEY” 按钮，选择 User-Key (user\_key.bin 文件)。
- 3) “RESET CKEY AES” 单选框选中则 Chip-Key 重加密，否则不重加密。
- 4) 点击 “RSA KEY” 按钮，选择 RAS-私钥 (pri\_key.pem 文件)。
- 5) 点击 “FILE” 按钮，选择要签名的 kernel 文件，生成对应的-dst.bin 文件。

## 4.2.5 other bin

签名和加密其他文件或者单独的 UBOOT 文件。



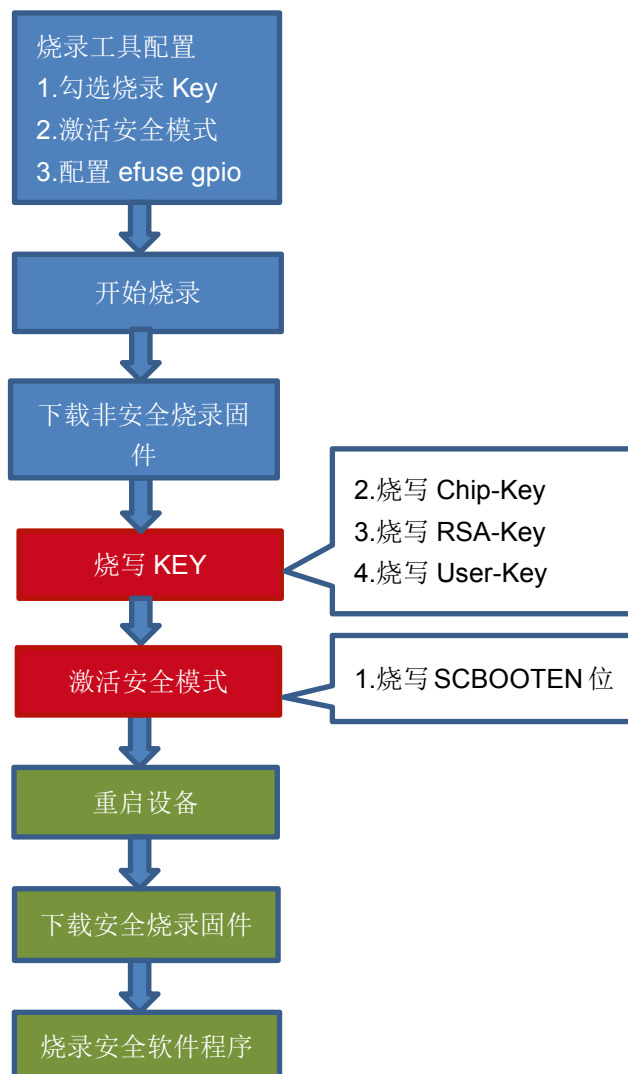
签名 UBOOT 文件操作步骤：

- 1) “AES” 单选框选中则加密，否则不加密。
- 2) 如果 AES 被选中，点击“AES KEY”按钮，选择 User-Key(user\_key.bin 文件)。
- 3) “SIGNUTARE” 单选框选中则签名，否则不签名。
- 4) 点击“RSA KEY”按钮，选择 RAS-私钥(pri\_key.pem 文件)。
- 5) 点击“FILE”按钮，选择要签名的 u-boot.bin 文件，生成对应的 u-boot-dst.bin 文件。

注：如果签名、加密成功则弹窗提醒签名成功，如果失败则弹窗提醒失败。

## 4.3 烧录工具

### 4.3.1 安全烧录流程



### 4.3.2 安全烧录配置

#### 烧录 KEY:

选中此选项会将 key 信息烧录进芯片的 EFUSE(OTP) 中, 此时并不会进入 Security boot 模式, 需要置 Security Boot 的标志位。

#### 激活安全模式:

选中此选项会置 Security boot 的标志位, 重启后会进入安全模式。必须先烧录 KEY 或同时勾选两个选项。

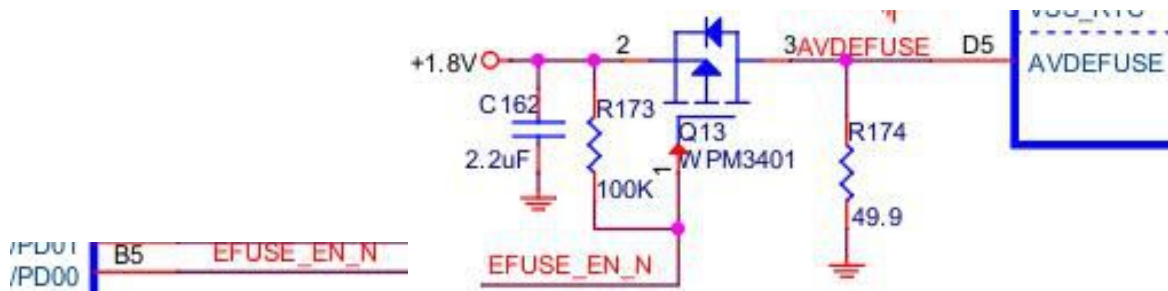
#### 强制重启:

选中此选项会在烧录 KEY 和激活安全模式后重启设备，烧录签名后加密的镜像。

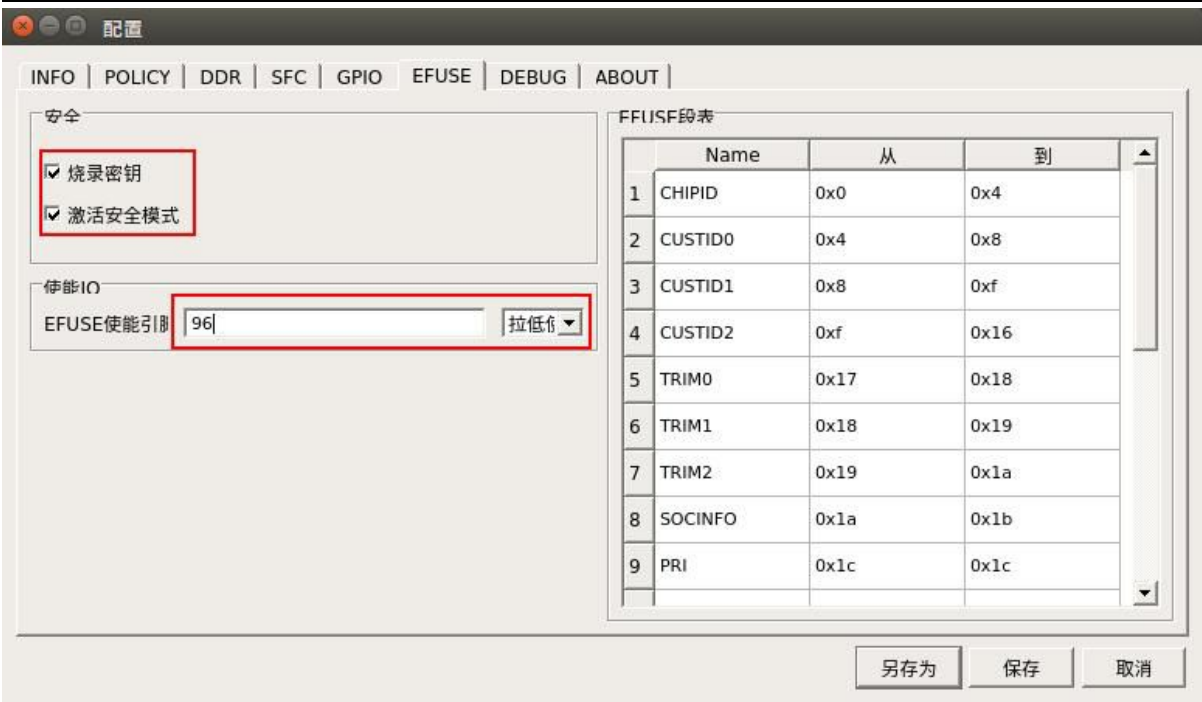


#### EFUSE GPIO:

参考 EFUSE 硬件原理图，EFUSE\_EN\_N 为 PD00，使能引脚号应配置 96，注：(32\*3)

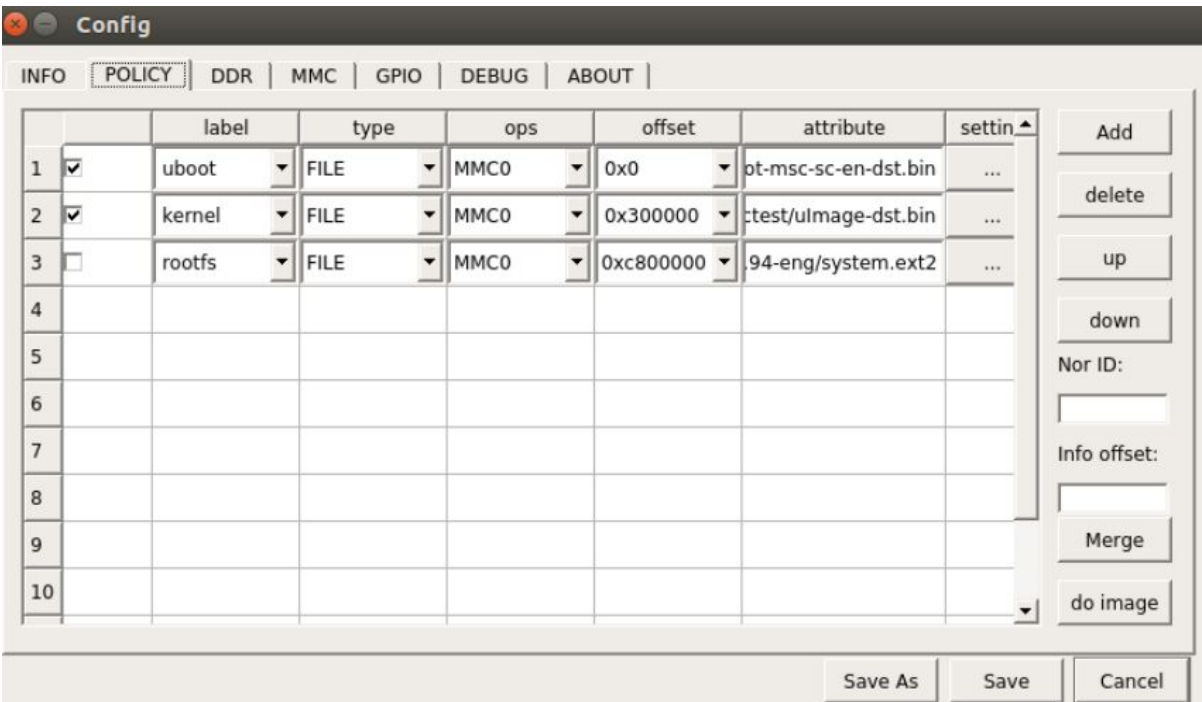


此配置根据不同的开发板略有不同，如果是 PMU 给 EFUSE 供电保持默认配置-1。



策略配置:

选择要烧录的安全固件路径后保存配置。





## 5. 安全建议

如果厂商需要使用 Security Boot，则需要进行如下步骤：

- ✧ 生成 N 和一对公私密钥 Ku 和 Kr。Kr 要有最高的保密级别，建议不出厂商的安全部门。
- ✧ 每当工程部门更新软件时将 bin 文件提供给安全部门，安全部门对此 bin 文件分段计算出 256 位的 SHA1 散列值，最开始的代码（SPL）的散列值用 N 和 Kr 作 RSA 加密后（数字签名 Kr 加密值）放在 bin 文件头的适当位置，提供给工厂用于生产或放置在网络上用于系统升级

如果厂商需要使用 User-Key 对代码加密，则需要进行如下步骤：

- ✧ 选定 256 位的 User-Key。User-Key 要有最高的保密级别，建议不出厂商的安全部门。
- ✧ 设立一个服务器由安全部门控制，在设备生产中烧录 User-Key 到每颗芯片的 OTP 时使用，以防烧录过程中 User-Key 的泄密。

注意事项：

- ✧ 防止烧录了 User-Key 但 SCB00T 位没有设置的芯片流出厂商的控制范围。因此应该在 SCB00T 设置后再烧录 User-Key。进一步而言，在烧录流程中尽早写 OTP 的 SCB00T 和 DISJTAG 位。因为没有 SCB00T，则无法控制这台设备的启动过程及其运行的代码。虽然 User-Key 不会直接泄露，但可以通过调用 SC-ROM 中的函数使用 User-Key 对任意数据进行解密。如果厂商的应用很简单，并不介意设备的拥有者能够使用 User-Key 解码任何数据，不设置 SCB00T 也没问题。
- ✧ 厂商在得到 Chip-Key 并使用完后不要记录，应该立即丢弃，以减少其泄漏的风险。厂商若需要记录每颗芯片可以记录 CHIP-ID
- ✧ RSA 的 Kr/Ku/N 对某个设备一旦选定不能改动
- ✧ AES 使用完后立即清除 key、data-in、data-out 寄存器
- ✧ 内存（片内 SRAM 或片外 DRAM）中的关键代码，比如解密等，使用完后予以擦除（写 0）。
- ✧ 启动流程进行到后面不再使用 Chip-Key 和 User-Key 时，设置寄存器（复位置 0 写置 1）使其失效。