# Linux RISCV 安全
# 开发指南

**版本号: 1.1**
**发布日期: 2025.2.19**

# 版本历史

| 版本号 | 日期 | 制/修订人 | 内容描述 |
| --- | --- | --- | --- |
| 1.0 | 2021.4.19 | AWA0480 | 添加基础模板 |
| 1.1 | 2025.2.19 | XAA0175 | 更新对 H135/H136 的支持 |

# 目　录

# 1 概述

## 1.1 编写目的

本文档主要介绍我司 RISCV 架构安全方案的功能。这包含安全启动 (Secure Boot)、安全存储（Secure Storage）等方面。

## 1.2 适用范围

表 1-1: 适用产品列表

| 产品名称 | 内核版本 | 驱动文件 |
| --- | --- | --- |
| F133 | Linux-5.4 | 无 |
| H135/H136 | Linux-6.6 | 无 |

## 1.3 相关人员

使用于 SUNXI 平台的安全驱动和安全应用的开发/维护人员。

## 1.4 相关术语

- API：Application Program Interface 应用程序接口
- SUNXI：指 Allwinner 的一系列 SOC 硬件平台
- CE：Crypto Engine，算法引擎，以前称作 SS
- AES：Advanced Encryption Standard，高级加密标准
- TRNG：True Random Number Generator, 真随机数发生器
- PRNG：Pseudo Random Number Generator，伪随机数发生器
- HUK：Hardware Unique Key，硬件唯一 key
- SSK：Secure Storage Key，安全存储 key

# 2 Secure Boot

Secure Boot，即安全启动，是一个**安全系统必不可少**的组成部分，是本文后续安全功能的基础。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。
- 设置安全的硬件环境。

## 2.1 安全启动原理

Linux 安全方案基于私钥签名-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。验证顺序为 brom->sboot->monitor(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。

## 2.2 生成安全固件

Longan 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，安全固件的打包步骤如下：

```
$ cd {SDK}
$ ./build.sh config
==> 选择相应的平台
$ ./build.sh
==> 编译。
$cd {SDK}/build
$ ./createkeys
==> 生成一组用于签名的密钥，不需要每次执行。生成的密钥路径位于out/{CHIP}/common/keys/。
$ cd {SDK}
$ ./build.sh pack_secure
==> 打包生成安全固件。
```

## 2.2.1 安全的配置

在生成安全固件之前，必须要进行以下方面的配置，才能成功的生成安全固件。

### 2.2.1.1 生成签名密钥

**⚠ 注意**

**在首次进行安全固件打包之前，必须运行一次./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。**

createkeys 脚本会根据dragon_toc*.cfg生成一组用于签名的密钥，生成的密钥保存在out/{CHIP}/common/keys/目录下。执行./build.sh pack_secure 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。


图 2-1: dragon-toc 配置文件说明

以 R328 为例，其dragon_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/dts 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（有的方案为 RootKey_Level_0.bin, RootKey_Level_0.pem 与 rotpk.bin）为根密钥相关文件，要重点保护。

Trustkey.bin 与 Trustkey.pem（RootKey_Level_0.bin 与 RootKey_Level_0.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。

#### 2.2.1.2 安全固件版本的配置

**注：./build.sh pack_secure 打包完成后**，生成的安全镜像位于out/目录下，文件名为{CHIP}_linux_{BOARD}_<uart0/card0>_secure_v[NUM].img。其中 NUM 为固件版本号，由version_base.mk文件决定。

SUNXI 提供了一种安全固件防回退机制，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。另外此功能需要烧写 efuse，所以务必保证 efuse 供电充足。**

## 2.2.2 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 secure enable bit。
2. 烧写 rotpk.bin 到 efuse 中 rotpk 区域。
3. 烧写安全固件到 flash 中。

#### 2.2.2.1 使能安全启动

**⚠ 注意**

- **烧写 secure enable bit 后，会让设备变成安全设备，此操作是不可逆的。后续将只能启动安全固件，启动不了非安全固件。**
- **如果既烧写了 secure enable bit，又烧写了 rotpk.bin，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 secure enable 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 secure enable bit，但是设备出厂前必须要烧写 rotpk.bin。**
- **烧写 efuse 操作的时候，请注意给 efuse 供电。通常在烧写安全固件、升级 sboot/uboot、DragonSN 烧 key 到 efuse 等场景会写 efuse。**

当需要把设备使能成安全设备的时候，需要配置 device/config/chips/{CHIP}/configs/{BOARD}/uboot-board.dts 文件

```
&target {
  burn_secure_mode = <1>
}
```

安全固件烧写的过程中，当会检测 burn_secure_mode 为 1 的时候，会烧写了 secure enable bit，否则烧写失败。

如何判断 secure enable bit 是否烧写?

- 因为只有 secure enable bit 烧写后才能启动安全固件，所以如果是安全启动，secure enable bit 就一定烧写了。安全启动过程中有一些特有的打印，如 “SBOOT is starting!”、“sboot commit...”、“OLD version:...”、“NEW version: ...”、“secure enable bit: 1”等等，可用来进行判断。
- 执行cat /sys/class/sunxi_info/sys_info，如果输出的结果中 sunxi_secure 为 secure，则表明 secure enable bit 已经烧写。

#### 2.2.2.2 烧写 ROTPK

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、Secure storage、efuse 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。

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uboot-board.dts或者sys_config.fex文件中 [target] 项下。如果未显式配置，按照 burn_key=0 来处理。
- 打包安全固件，烧写到 flash 中。

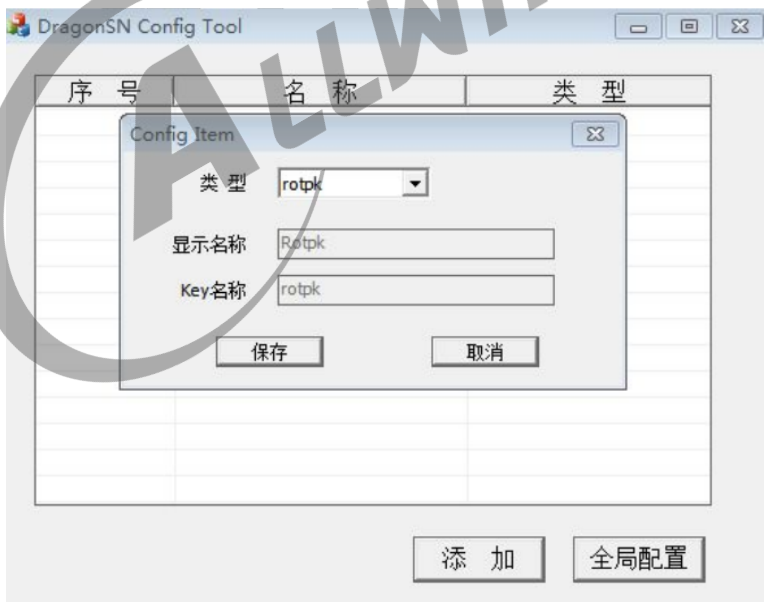


图 2-2: rotpk 烧写配置

- 在 PC 端对 DragonSN 工具进行配置。打开 DragonSNConfig.exe，如上图所示，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。
- 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，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，匹配后才烧录该 rotpk.bin。

# 3 安全存储

开放 OS 中，数据没有基于硬件的防护，极易被拦截或篡改。数据在传输过程中被窃取或篡改，恶意数据在传输环节中被注入，因为此为了保护设备上敏感数据或高价值的数据，所以 SUNXI 平台提供一种安全存储的实现方式。

## 3.1 安全存储的原理

SUNXI 在 efuse 上预留 2 组 AES 的 key:SSK 和 HUK

- 支持 AES 类算法：ECB/CBC/CTR 等算法，更多算法请看 CE Spec。
- 这些 key 都有读写保护位。
- 当这些 key 的读保护位后，只有 CE 可以访问，其他设备无法访问。

安全存储基于芯片硬件的加解密引擎 (CE)，支持各类对称加密算法等，其次利用芯片内置 efuse 的安全密钥，对数据做加解密处理，由于密钥只要 CE 可以访问，其他设备无法访问密钥。因此其安全性大大提高。

## 3.2 烧写 SSK

DragonSN 烧 ssk.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uboot-board.dts或者sys_config.fex文件中 [target] 项下。如果未显式配置，按照 burn_key=0 来处理。
- 打包安全固件，烧写到 flash 中。

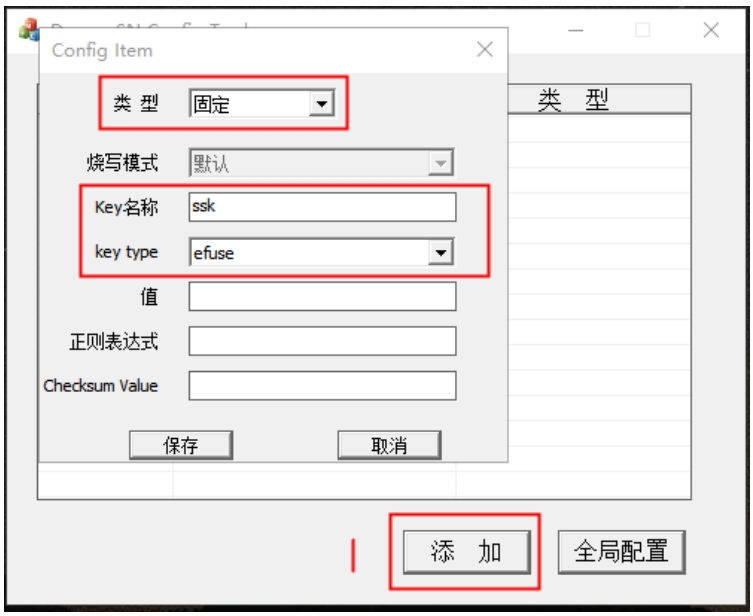


图 3-1: ssk 烧写配置

- 在 PC 端对 DragonSN 工具进行配置。打开 DragonSNConfig.exe，如上图所示，点击 “添加”，在 “类型” 一栏下拉菜单中选择 ssk，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。
- 运行 DragonSN.exe 工具，配置 ssk.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 ssk.bin，在烧录过程中，会将烧写 efuse 中 key 和 PC 端传下来 key 进行对比，一致后才会烧写读写保护位。

如何判断 SSK 是否烧写成功？- 查看它的读写保护位是否为 1。

- 对比加解密的出来的结果是否和 PC 端计算出来的结果一一致。

# 3.3 烧写 HUK

## 3.3.1 HUK 的生成

基于芯片硬件的加解密引擎 (CE)，它支持 TRNG 算法，生成一个可靠的不可预测随机数作为 key，提供 key 的安全性。

## 3.3.2 HUK 的烧写配置

使能烧写 HUK 的配置，修改{SDK}/brandy/brandy-2.0/spl/board/{CHIPS}/sboot.mk配置文件，如下：

```
CFG_SUNXI_KEY_PROVISION=y
```

在烧录过程中，会将烧写 efuse 中 key 和生成 key 进行对比，一致后才会烧写读写保护位。

如何判断 HUK 是否烧写成功?

- 查看它的读写保护位是否为 1。
- 对比加解密的出来的结果是否和 PC 端计算出来的结果一一致。

# 4 openssl 的接口

如果应用层想采用硬件 CE 进行开发, 则需要利用 openssl 标准的接口, 才能调用 CE 驱动。OpenSSL 的接口说明, 可以在官网中找到对应的算法接口。下文以 demo 形式演示 OpenSSL 的几种应用,demo 源文件需要放在 OpenSSL 中，编译和运行都需要 OpenSSL 的动态库支持。

## 4.1 openssl 的配置与编译

### 4.1.1 openssl 的配置

openssl 现在代码库，已经适配好一些标准算法和架构平台的的配置，并且和 longan 的配置文件绑定在一起了，因此只需要在 longan 下进行配置即可。

```
$cd {SDK}
$./build.sh config
```

### 4.1.2 openssl 的编译说明

openssl 现在代码库，已经和 longan 的 liunx 的编译工具绑定在一起了，因此当 longan 配置好后，进行如下编译

```
$cd openssl-1.0.0
$make clean
$make
```

### 4.1.3 openssl 的库文件的生成

如果应用层用 openssl 进行开发, 则需要包含 openssl 的库文件进行开发。openssl 的库文件的生成, 命令如下:

```
$cd openssl-1.0.0
$make install
```

执行成功后, 会在 openssl-1.0.0/out 目录下生成以下文件:

```
openssl-1.0.0/out
└── usr
    └── ssl
        ├── bin    // OpenSSL可执行文件
```

```
    ├── certs   // 目前为空，可存放数据证书
    ├── include  // OpenSSL的接口头文件
    ├── lib     // OpenSSL会用到的动态库，cd 包括所有engine
    ├── man     // 帮助手册（man命令需要的格式）
    ├── misc    // 其他工具ls
    ├── openssl.cnf  // OpenSSL的配置文件
  └── private   // 目前为空
```

应用层在进行开发时, 需要链接 lib 目录下 3 个动态库文件:

```
├── libcrypto.so.1.0.0
├── libssl.so.1.0.0
├── libaf_alg.so
```

# 4.2 openssl demo 用例说明

## 4.2.1 使用 af_alg 引擎

因为要使用 af_alg 引擎，需要在初始化 OpenSSL 时显式的指定加密引擎。

```
ENGINE *openssl_engine_init(char *type)
{
    ENGINE *e = NULL;
    const char *name = "af_alg";

    OpenSSL_add_all_algorithms();
    ENGINE_load_builtin_engines();

    e = ENGINE_by_id(name);
    if (!e) {
        DBG("find engine %s error\n", name);
        return NULL;
    }

    ENGINE_ctrl_cmd_string(e, "DIGESTS", type, 0);
    return e;
}

void openssl_engine_free(ENGINE *e)
{
    if (e != NULL)
        ENGINE_free(e);

    ENGINE_cleanup();
    EVP_cleanup();
}
```

## 4.2.2 使用安全密钥进行 AES 加解密 demo

Linux 的应用层调用 openssl 的 API 来使用安全 KEY 进行加解密。

说明

- **如果需要 SSK 进行加解密，因此 KEY 的字符串选择 CE_KS_SSK。**
- **如果需要 HUK 进行加解密，因此 KEY 的字符串选择 CE_KS_HUK。**

```
/* The identification string to indicate the key source. */
#define CE_KS_INPUT       "default"
#define CE_KS_SSK         "KEY_SEL_SSK"
#define CE_KS_HUK         "KEY_SEL_HUK"
#define CE_KS_RSSK        "KEY_SEL_RSSK"

unsigned char g_inbuf[SS_TEST_BUF_SIZE] = {0};
unsigned char g_outbuf[SS_TEST_BUF_SIZE] = {0};
unsigned char g_key[AES_KEY_SIZE_256] = {
        0xFF, 0xEE, 0xDD, 0xCC, 0xBB, 0xAA, 0x99, 0x88,
        0x77, 0x66, 0x55, 0x44, 0x33, 0x22, 0x11, 0x00,
        0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
        0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF};
const unsigned char g_iv[AES_BLOCK_SIZE] = {
        0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
        0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF};

int main(int argc, char *argv[])
{
  int ret = 0;
  int enc = 0;
  int inl = 0;
  int outl = 0;
  FILE *in = NULL;
  FILE *out = NULL;
  ENGINE *e = NULL;
  EVP_CIPHER_CTX ctx = {0};
  const EVP_CIPHER *e_cipher = NULL;

  if (argc != PT_NUM) {
    usage();
    return -1;
  }

  in = fopen(argv[PT_IN_FILE], "rb");
  if (in == NULL) {
    DBG("Failed to fopen(%s)! \n", argv[PT_IN_FILE]);
    return -1;
  }
  out = fopen(argv[PT_OUT_FILE], "wb");
  if (out == NULL) {
    DBG("Failed to fopen(%s)! \n", argv[PT_OUT_FILE]);
    ret = -1;
    goto error;
  }

  if (strncmp(argv[PT_ENC_DIR], "enc", 3) == 0)
    enc = 1;

  e = openssl_engine_init();
  if (e == NULL) {
    ret = -1;
    goto error;
  }
  e_cipher = ENGINE_get_cipher(e, NID_aes_128_cbc);
```

```
    if (e_cipher == NULL) {
        ret = -1;
        goto error;
    }

    EVP_CipherInit(&ctx, e_cipher, g_key, g_iv, enc);
    for (;;) {
        inl = fread(g_inbuf, 1, SS_TEST_BUF_SIZE, in);
        if (inl <= 0) {
            if (inl < 0)
                DBG("read(%d) return %d. \n", SS_TEST_BUF_SIZE, inl);
            break;
        }

        if (inl > 0) {
            EVP_CipherUpdate(&ctx, g_outbuf, &outl, g_inbuf, inl);
            DBG("Update: inl %d, outl %d \n", inl, outl);
            fwrite(g_outbuf, 1, outl, out);
        }
    }
    EVP_CipherFinal(&ctx, g_outbuf, &outl);
    DBG("Update: outl %d \n", outl);
    if (outl > 0)
        fwrite(g_outbuf, 1, outl, out);

error:
    if (in != NULL)
        fclose(in);
    if (out != NULL)
        fclose(out);

    EVP_CIPHER_CTX_cleanup(&ctx);
    openssl_engine_free(e);
    return ret;
}
```

详细的 demo 文件请查看 openssl-1.0.0/ss_test/目录下。

**著作权声明**



**商标声明**



**免责声明**