BK7236 启动分析与设计

**BL1，BL2**

**安全升级，安全调试**

博通集成电路（上海）股份有限公司

上海浦东张江张东路1387号41栋

电话：86-21-51086811

传真：86-21-60871089

[www.bekencorp.com](http://www.bekencorp.com)

**目录**

[1. 文档介绍 7](#_Toc127460060)

[1.1. 文档范围 7](#_Toc127460061)

[1.2. 读者对象 7](#_Toc127460062)

[1.3. 阅读建议 7](#_Toc127460063)

[1.4. 问题反馈 7](#_Toc127460064)

[1.5. 参考资料 7](#_Toc127460065)

[1.6. 术语与缩写 7](#_Toc127460066)

[1.6.1. 述语 7](#_Toc127460067)

[1.6.2. 缩写 7](#_Toc127460068)

[2. 需求分析 8](#_Toc127460069)

[3. 竞争分析 8](#_Toc127460070)

[3.1. BK7238 分析 8](#_Toc127460071)

[3.2. BK7256 分析 8](#_Toc127460072)

[3.3. STM32 分析 8](#_Toc127460073)

[4. 开源软件分析 8](#_Toc127460074)

[4.1. BL1 8](#_Toc127460075)

[4.1.1. 核心数据结构 9](#_Toc127460076)

[4.1.2. 流程抛析 9](#_Toc127460077)

[4.1.3. Manifest 解析 14](#_Toc127460078)

[4.1.4. OTA 17](#_Toc127460079)

[4.2. BL2 17](#_Toc127460080)

[4.2.1. 核心数据结构 18](#_Toc127460081)

[4.2.2. 关键流程 19](#_Toc127460082)

[5. 硬件分析 20](#_Toc127460083)

[5.1. SoC 架构 20](#_Toc127460084)

[5.2. TE200 20](#_Toc127460085)

[5.3. OTP 21](#_Toc127460086)

[5.4. 寄存器 21](#_Toc127460087)

[6. 原理分析 21](#_Toc127460088)

[7. 专利分析 21](#_Toc127460089)

[7.1. BL1 21](#_Toc127460090)

[7.1.1. 抵抗故障注入攻击方法 21](#_Toc127460091)

[7.1.2. 安全启动补丁 21](#_Toc127460092)

[7.1.3. 深睡眠启动 21](#_Toc127460093)

[7.1.4. FLASH CRC 21](#_Toc127460094)

[7.2. BL2 21](#_Toc127460095)

[8. 方案概述 21](#_Toc127460096)

[8.1. 方案选型 21](#_Toc127460097)

[9. BL1 设计 21](#_Toc127460098)

[9.1. BL1 移植 22](#_Toc127460099)

[9.2. OTP 设计 22](#_Toc127460100)

[9.2.1. OTP 分区 22](#_Toc127460101)

[9.2.2. OTP 硬件区 24](#_Toc127460102)

[9.2.3. OTP 软件区 27](#_Toc127460103)

[9.2.4. OTP 访问 28](#_Toc127460104)

[9.2.5. OTP 权限控制 28](#_Toc127460105)

[9.3. EFUSE 设计 29](#_Toc127460106)

[9.4. FLASH 布局 29](#_Toc127460107)

[9.4.1. MPW1 29](#_Toc127460108)

[9.4.2. MPW2 30](#_Toc127460109)

[9.4.3. MPW3 30](#_Toc127460110)

[9.5. FLASH 读写 30](#_Toc127460111)

[9.5.1. 实际实现 30](#_Toc127460112)

[9.5.2. 方案选型 31](#_Toc127460113)

[9.5.3. BL1 FLASH 免写 31](#_Toc127460114)

[9.6. 启动流程 32](#_Toc127460115)

[9.6.1. 与山海启动差异点 32](#_Toc127460116)

[9.6.2. Boot Mode 寄存器 32](#_Toc127460117)

[9.6.3. 启动流程 33](#_Toc127460118)

[9.7. 启动性能 33](#_Toc127460119)

[9.8. FIH 攻击 34](#_Toc127460120)

[9.9. BL2 升级 35](#_Toc127460121)

[9.9.1. 概述 35](#_Toc127460122)

[9.9.2. BC 设计 36](#_Toc127460123)

[9.9.3. BL2 BF 调整策略 36](#_Toc127460124)

[9.9.4. OTA Slot 选择 37](#_Toc127460125)

[9.9.5. 异常场景说明 37](#_Toc127460126)

[9.10. 验签失败处理 38](#_Toc127460127)

[9.11. IMAGE 打包 38](#_Toc127460128)

[9.11.1. 概述 39](#_Toc127460129)

[9.11.2. 仿真 OTP 生成 39](#_Toc127460130)

[9.11.3. Public Key 解析 40](#_Toc127460131)

[9.12. 版本管理 40](#_Toc127460132)

[9.13. 待完成项 41](#_Toc127460133)

[9.14. 风险点 41](#_Toc127460134)

[10. BL2 设计 41](#_Toc127460135)

[10.1. 方案概述 42](#_Toc127460136)

[10.2. BL2 升级 42](#_Toc127460137)

[10.2.1. 版本降级攻击 42](#_Toc127460138)

[10.3. APP 升级 42](#_Toc127460139)

[10.4. BL2 移植 42](#_Toc127460140)

[10.5. 差分升级 42](#_Toc127460141)

[10.6. 安全调试 42](#_Toc127460142)

[11. 安全部署 42](#_Toc127460143)

[12. 工具 42](#_Toc127460144)

[13. 示例设计 43](#_Toc127460145)

[14. 测试 43](#_Toc127460146)

[14.1. 配置 43](#_Toc127460147)

[14.1.1. EFUSE 43](#_Toc127460148)

[14.1.2. OTP 43](#_Toc127460149)

[14.1.3. FLASH AES KEY 44](#_Toc127460150)

[14.1.4. Public Key Hash 烧录 46](#_Toc127460151)

[14.2. BL1用例 47](#_Toc127460152)

[14.3. BL1测试示例 47](#_Toc127460153)

[14.3.1. 注意事项 48](#_Toc127460154)

[14.3.2. 芯片上跑传统下载 49](#_Toc127460155)

[14.3.3. 芯片上跑 BL1 49](#_Toc127460156)

[14.3.4. 芯片FLASH 中跑 BL1 53](#_Toc127460157)

[14.3.5. FPGA 上跑 BL1 54](#_Toc127460158)

[14.3.6. 芯片内存中跑 BL1 54](#_Toc127460159)

[14.3.7. 仿真 BL1 54](#_Toc127460160)

[14.3.8. TRNG 测试 55](#_Toc127460161)

[14.4. BL2 用例 58](#_Toc127460162)

[14.5. BL2 测试示例 58](#_Toc127460163)

[14.5.1. BL1 -> Armino S 58](#_Toc127460164)

[14.5.2. BL1 -> TFM -> Armino NS 58](#_Toc127460165)

[14.5.3. BL1 -> BL2 -> TFM 58](#_Toc127460166)

[14.5.4. BL1 -> BL2 -> TFM -> Armino NS 58](#_Toc127460167)

[14.5.5. BL1 -> BL2 -> TFM -> Armino NS -> CPU1 58](#_Toc127460168)

[14.5.6. OTA – Armino 58](#_Toc127460169)

[14.5.7. OTA – Armino + TFM + CPU1 58](#_Toc127460170)

[14.5.8. OTA -> BL2 + Armino + TFM + CPU1 58](#_Toc127460171)

[14.5.9. OTA – CPU1 58](#_Toc127460172)

[14.5.10. OTA – BL2 58](#_Toc127460173)

[14.5.11. Debug 58](#_Toc127460174)

[14.5.12. 量产版本部署 58](#_Toc127460175)

[15. 成本 58](#_Toc127460176)

[16. 可移植性 58](#_Toc127460177)

[17. 风险，问题与优化 58](#_Toc127460178)

[18. 其他 58](#_Toc127460179)

**更新历史**

|  |  |  |  |
| --- | --- | --- | --- |
| **版本** | **描述** | **日期** | **作者** |
| v0.1 | 增加 BL1 | 2023-2-13 | Peter |
| v0.2 | 增加 BL2，安全升级 |  | 曾鑫 |
| v0.3 | 增加 BL2，安全调试 |  | 曾鑫 |
| v0.4 | 增加 BL2，安全部署 |  | 曾鑫 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 文档介绍

## 文档范围

包括 BL1，BL2以及 OTA 的设计。

## 读者对象

主要读者为：

1. BL1/BL2 开发与维护者
2. TF-M 以及安全相关开发与维护者
3. QA

## 阅读建议

建议：

1. 如果您想了解 BootROM 设计，请阅读 [BL1 设计](#_BL1)。
2. 如果您想了解如何在芯片上调试 BootROM，请阅读 [BL1测试示例](#_BL1测试示例)。
3. 如果您想了解开发阶段 BootROM 上已做测试，请阅读 [BL1 测试用例](#_BL1用例)。

## 问题反馈

## 参考资料

## 术语与缩写

### 述语

* 安全启动

### 缩写

* PSA

# 需求分析

# 竞争分析

在开发过程中完成本节。

## BK7238 分析

重点分析 BK7238 升级，省内存算法。

## BK7256 分析

重点分析 BK7256 的安全启动，升级，调试，以及类 TrustZone 机制。

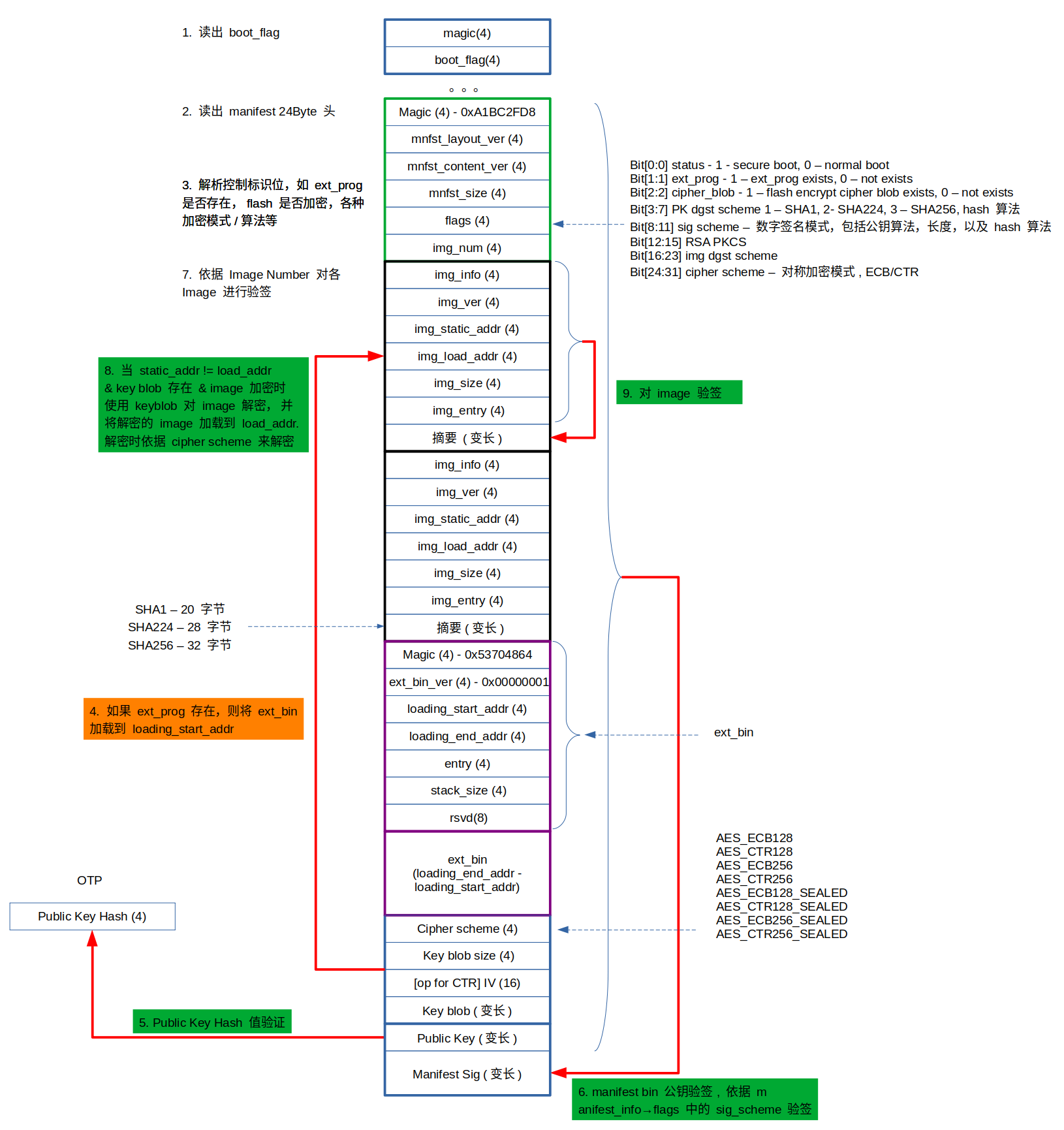
## STM32 分析

# 开源软件分析

## BL1

BK7236 BL1 以山海的 BootROM 为基础来开发，本节对关键内容作分析。

### 核心数据结构



### 流程抛析

#### SB 使能与关闭

关键点:

1. 在 manifest 中有一项 mnft\_desc\_cfg.sec\_boot, 它被编码到 manifest BIN 中 manifest\_info->flags 的 bit(0)
2. 当置 1 时， secure boot 使能
3. 当置 0 时， secure boot 关闭
4. 两者差异在于, secure boot 多了下面的流程:
   1. verify Public Key Hash
   2. verify Manifest Sig

#### Primary/Recovery Manifest

关键点:

1. OTP 中存放了 Primary Version, Recovery Version
2. Primary/Recovery Manifest 文件中也有一项 mnft\_desc\_cfg.mnft\_ver
3. 要求 Primary/Reocvery Manifest 文件中的版本号不小于对应的 OTP 版本号，小于则升级失败，通过这种方式进行 anti-rollback
4. 从代码看来，Primary/Recovery 之间没有直接联系，版本号比较完全独立
5. 最终决定走 primary/recovery 取决于两个因素:
   1. LAYOUT\_CTRL\_PARTITION\_FLASH\_OFFSET 中第二个字的值 (boot\_type), 为 1 走 primary, 为 0 走 recovery
   2. 当 preferred 失败时，才走另一个

对 secure boot 用处的推测:

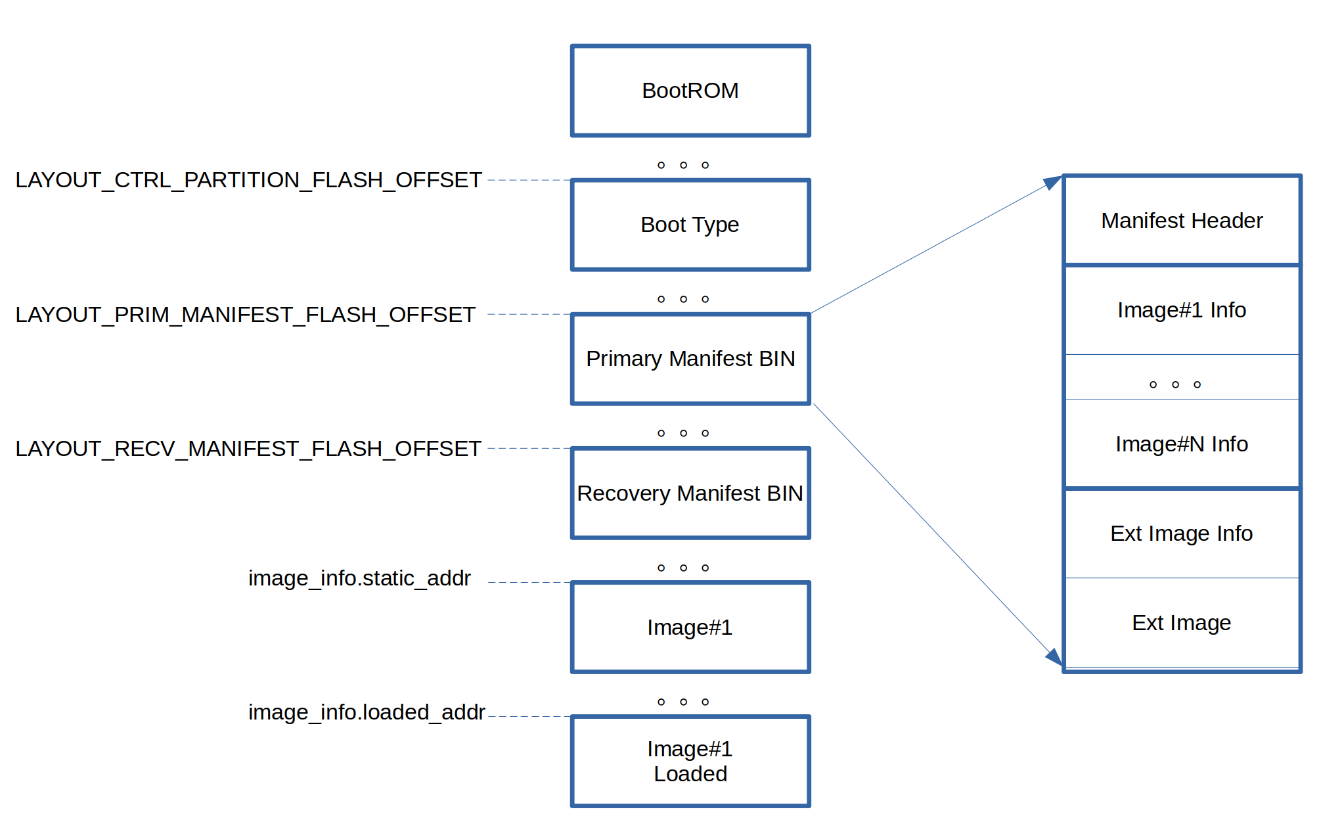
1. 用于支持 S 初始化与 NS 初始化, 如 primary 中的地址均为 S 地址，Recovery 中的地址均为 NS 地址
2. 用于支持下一级 bin (一般为 bootloader, 理论上也可为 app bin) 的升级，当前在 Primary 时，则写 Recovery, 否则写 Primary, 如果是这种实现，那 Primary/Recovery 版本号之间应该是有关系的，似乎又不合理

#### LCS 对 boot 影响

影响:

1. 如果 LCS 为 CM/DM, 则可关闭 secure boot
2. 否则必须使能 secure boot

#### 关键地址



#### 数字签名与消息摘要

BootROM 中涉及一次数字签名, 一次解密，一次加密，N + 2 次(消息摘要) 计算, N 为 image 数目:

1. Public Key Hash 计算 (digest)
2. Manifest Bin Hash 计算 (digest)
3. 使用公钥对 Manifest Bin Hash 进行加密 (加密)
4. 使用 keyblob + keyladder 对 image 进行解密
5. 对 N 个 Image 计算 Hash 值 (digest)

其中 Minfest Bin Hash 计算 + 对之加密构建验签过程。

#### 加载与跳转

两类加载:

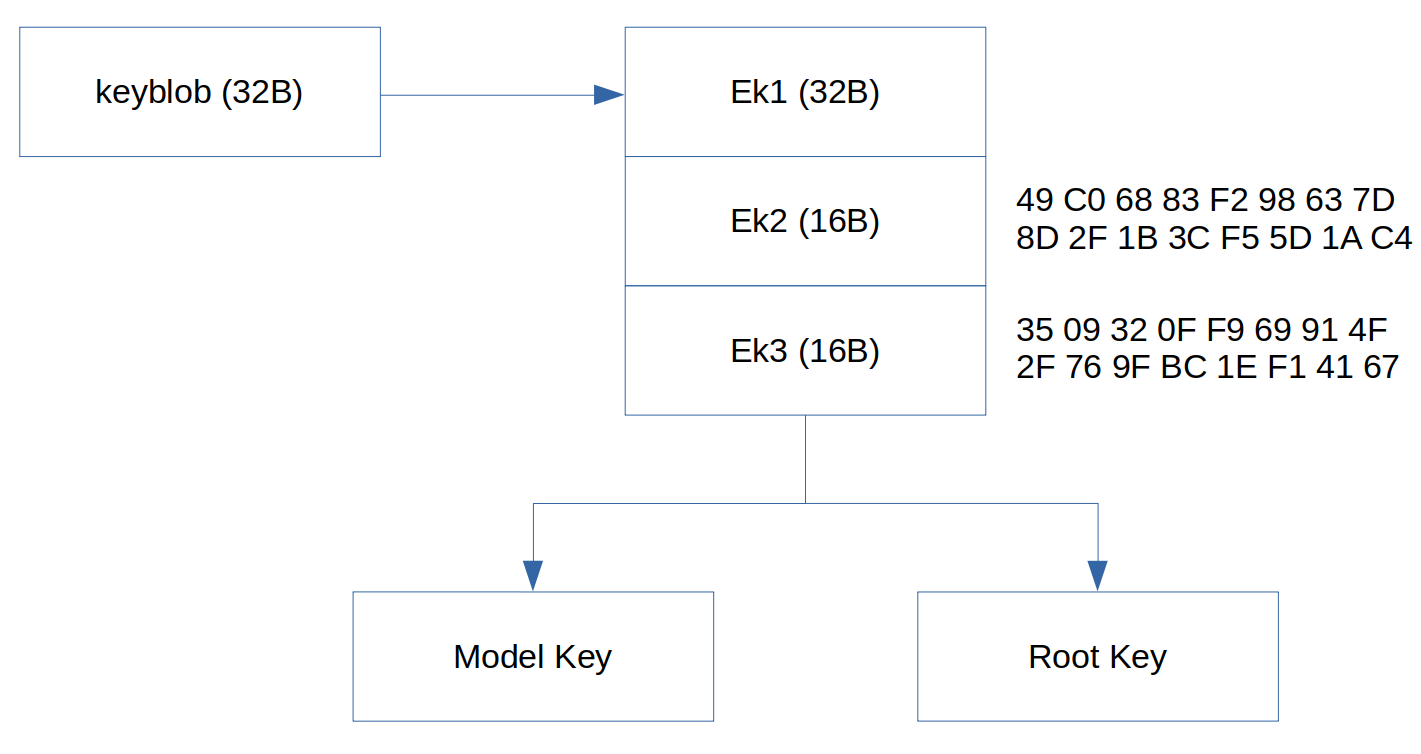
1. 加载 Ext Image
2. 加载 Image

两类跳转:

1. 跳向 Ext Image Entry
2. 跳向 Image Entry

#### Keybob 使用

文档中未详细说明 keyblob + keyladder 的使用，但推测实现机制如下:



步骤:

1. secure\_boot\_tool 与 TE200 都知道 EK2/EK3
2. secure\_boot\_tool 与 TE200 使用相同的派生算法派生出 Model Key/Root Key
3. secure\_boot\_tool 使用 model key 加密 image
4. TE200 使用 model key 解密 image

注意：

1. 整个算法的安全性建立在派生算法以及固定的 Ek2/Ek3 上。
2. 派生算法 secure\_boot\_tool/TE200 的实现者知道
3. EK2/EK3 bootrom 源代码中可找见
4. 而 EK1 则在 manifest bin 中可找着

因此这个算法存在明显的安全漏洞，这个漏洞似乎比直接将 model key 烧到 OTP 中更大。

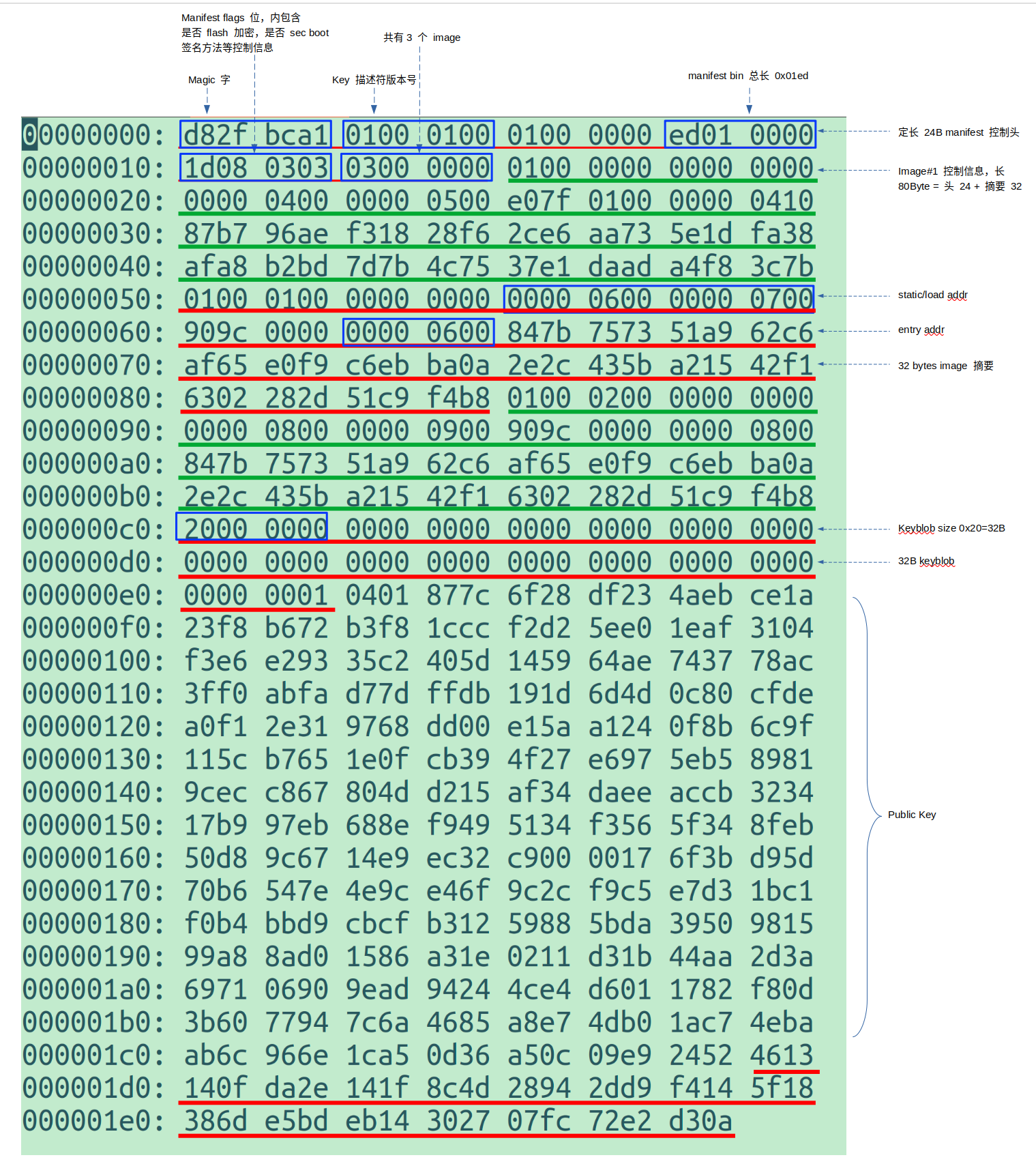
#### Key 的使用

1. RootKey 用于 FOTA
2. ModelKey 用于 Flash 加密 (但 bk7236 未用到)
3. FOTA Key 为 hard-coded key, 用于
4. Public Key 用于对 manifest 验签

#### 版本号与 Anti-Rollback

1. FOTA 用于防止升级的版本 rollback
2. Primary Version 防止地 Primary Manifest 版本 rollback, 这通常是指 manifest layout
3. Recovery Version 防止地 Recovery Manifest 版本 rollback, 这通常是指 manifest layout

### Manifest 解析



对应的 Key Descriptor

{

"//": "Manifest key descriptor, arm CHINA",

"//": "This is the manifest key descriptor of secure boot.",

"//": "This descriptor file is the sample descriptor.",

"//": "This descriptor file is used for SHANHAI IPSS system demonstration sample.",

"//": "Note that fmt\_ver is reserved by the secure boot tool.",

"key\_desc\_cfg": {

"fmt\_ver": "0x00010001"

},

"mnft\_sig\_cfg": {

"pubkey\_hash\_sch": "SHA256",

"mnft\_sig\_sch": "ECDSA\_521\_SHA256",

"mnft\_prvkey": "ec\_521\_privkey.pem",

"mnft\_pubkey": "ec\_521\_pubkey.pem"

},

"img\_dgst\_cfg" : {

"img\_hash\_sch": "SHA256"

},

"//": "Optional, only required if image(s) encrypted.",

"//": "In the system demonstration sample, the recovery bootloader is encrypted.",

"img\_enc\_cfg": {

"enc\_sch": "AES-ECB",

"enc\_key": "B75AFF7160BE351026753874D38846BEDE916E167043382B313FF2FA54250B6E",

"bits\_enc\_key": 256,

"//": "optional if enc\_sch doesn't need it",

"iv": "11111111111111111111111111111111",

"//": "In our BootROM hal implementation, the key blob is not used in converting to",

"//": "the real encryption key. The HAL implementation compares the key blob",

"//": "and selects the actual image encryption key. Details see hal\_crypto\_aes.c",

"key\_blob": "0000000000000000000000000000000000000000000000000000000000000001",

"bits\_key\_blob": 256

}

}

对应的 Manifest Descriptor

"//": "Manifest descriptor, arm CHINA",

"//": "This is the manifest descriptor of secure boot.",

"//": "There are three sections: mnft\_desc\_cfg, imgs, ext\_bin (optional).",

"//": "mnft\_desc\_cfg - the configuration of the manifest file.",

"//": "imgs - the image information.",

"//": "ext\_bin - the extended binary information",

"//": "This descriptor file is used for IPSS Class-B system demonstration sample.",

"//": "In primary boot path, we utilize the basic functionalities of secure boot:",

"//": " 1. image verification, with RSASSA\_1024\_PKCS\_V15\_SHA1 signature scheme and SHA224 image digest scheme",

"mnft\_desc\_cfg": {

"fmt\_ver": "0x00010001",

"mnft\_ver": "0x00000001",

"sec\_boot": true

},

"imgs": [

{

"is\_enc": true,

"static\_addr": "0x00040000",

"load\_addr": "0x00050000",

"entry": "0x10040000",

"path": "prim\_bl.bin"

},

{

"is\_enc": true,

"static\_addr": "0x00060000",

"load\_addr": "0x00070000",

"entry": "0x00060000",

"path": "spe.bin"

},

{

"is\_enc": true,

"static\_addr": "0x00080000",

"load\_addr": "0x00090000",

"entry": "0x00080000",

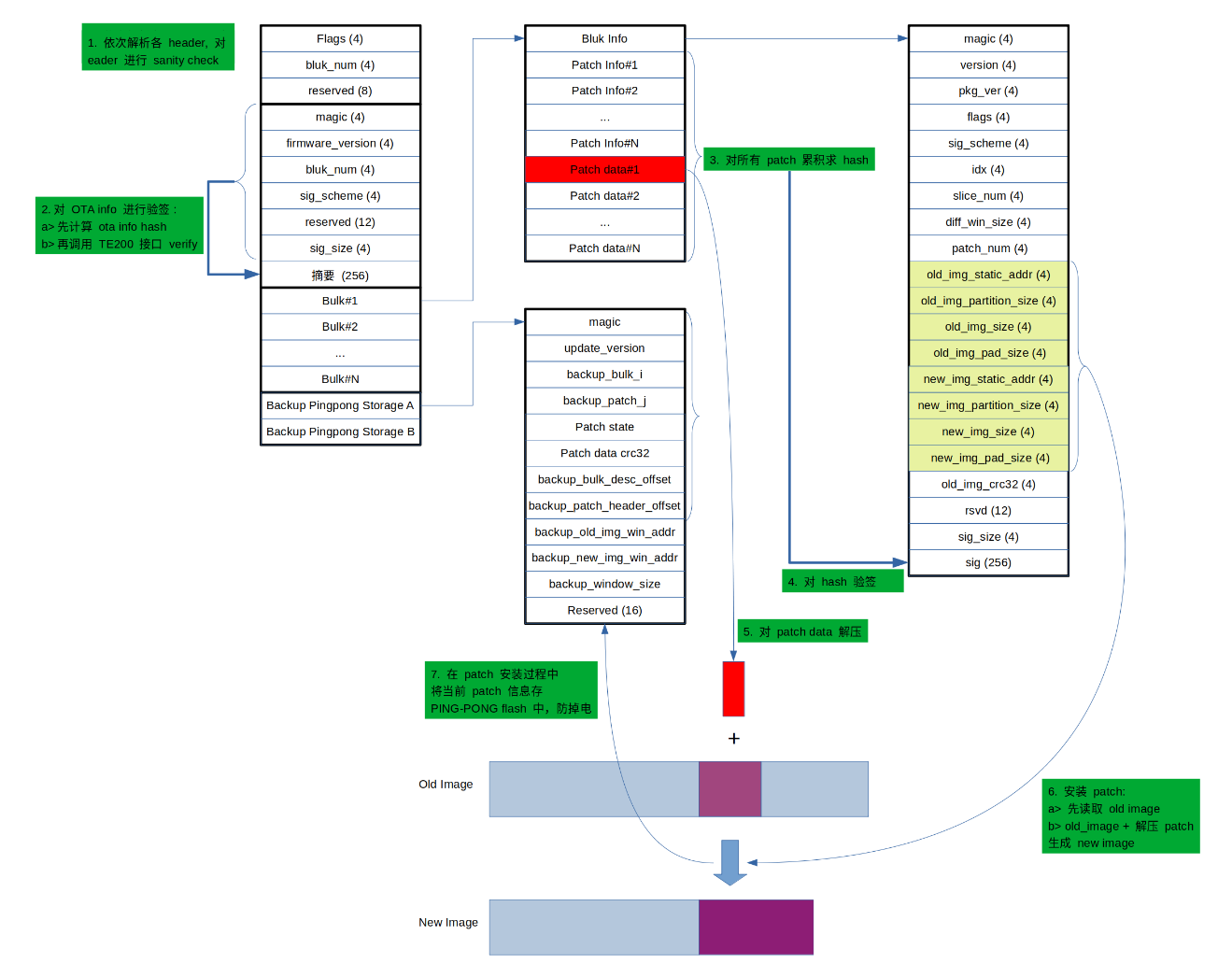
"path": "freertos.bin"

}

]

}

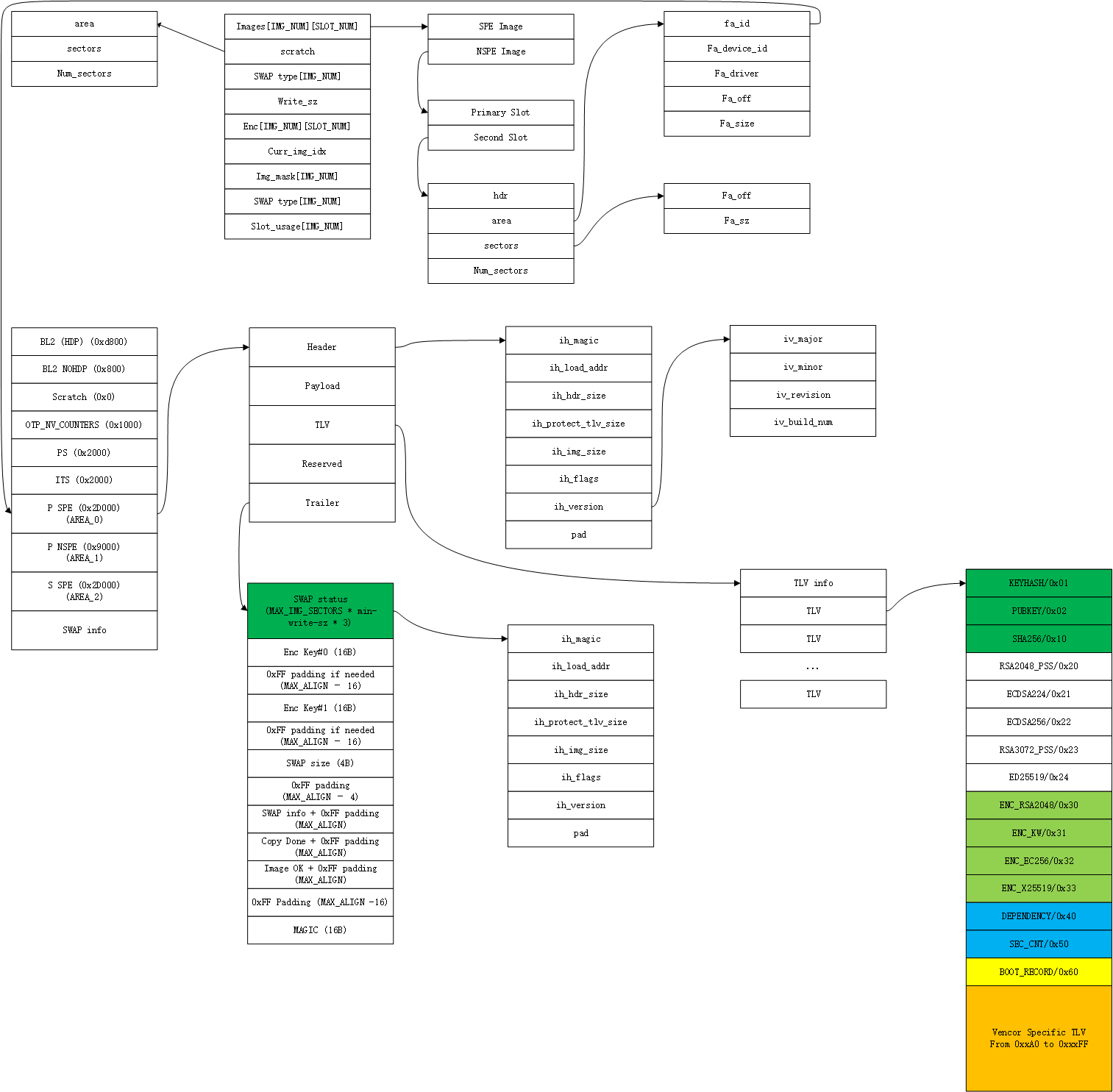
### OTA



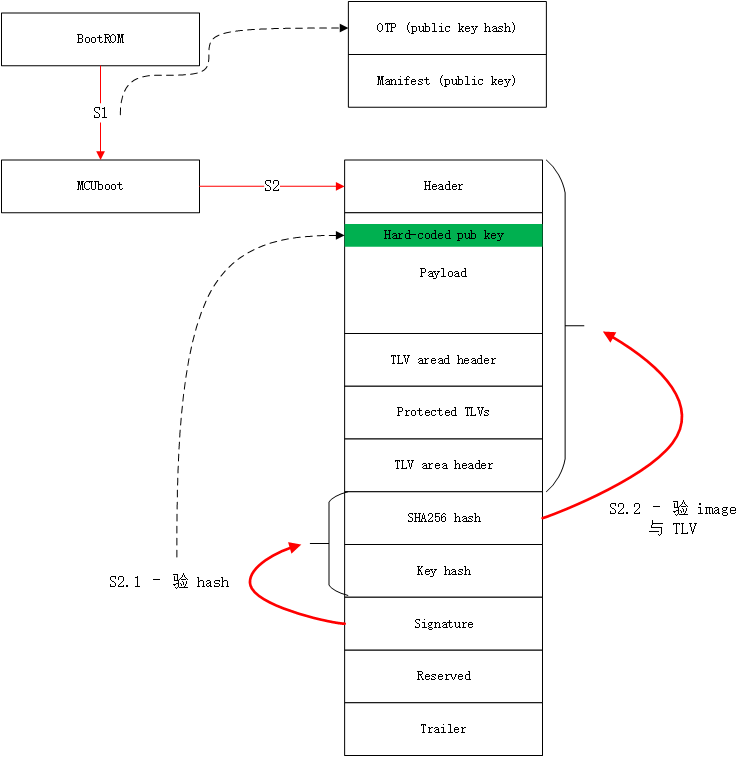
## BL2

BK7236 BL2 基于 MCUBOOT，MCUBOOT 官方文档比较完善，更多细节参考官方文档。

### 核心数据结构



### 关键流程



流程如下：

1. S1 - BootROM 对 mcuboot 进行验签名
2. S2 - MCUboot 对各 Images 验签:
   1. S2.1 - 安全验签: 使用 MCUboot hard-coded public key 对 Images 中的签名进行验签
   2. S2.2 - 完整性校验：校验 TLV 中的 SHA256 hash 值

public key 有两种方式:

1. 默认使用 hard-coded public key 进行验签名
2. 当使用其他方式时，需要使能 MCUBOOT\_HW\_KEY，并实现 boot\_retrieve\_public\_key\_hash() API.

# 硬件分析

## SoC 架构

BK7236 2022版本，采用22 nm，定位于低功耗、高安全的Wi-Fi 6 SoC，以适用于最广泛且有品质要求的IoT市场，并且在16 Mbit FLASH SIP这个约束下，本产品做到同类产品die size最小，目标die size 4.8 mm2 （要求形状：2+\*2.4+，方便SIP FLASH）。

数字电路首次采用ARM Star M33和山海安全单元以及PUF，实现现有IoT Wi-Fi最高级别的安全（PSA Level 2，暂时不抵抗旁信道攻击SPA/DPA）。这里M33分别运行80（120 更好）MHz和240 （320或者360更好）MHz，主M33（M33，不支持浮点）使用80 MHz是为了实现低漏电（必须做到：3.3V DCDC下stand-by能做到10 uA），副M33（M33F支持浮点）使用240 MHz是为了实现高速计算（比如电机控制算法）。

数字电路Wi-Fi/BT以及外设等主要从BK7256继承（基本不存在删减），同时继续提高可靠性和降低功耗。

射频电路从BK7238继承（同时根据7231N郭博士或者雅典MPW提升版本，提升发射性能和接收NF），并进一步降低接收功耗；电源管理系统直接采用后续BK7239结构，实现最低的待机和工作功耗。



## TE200

请参考 < Shanhai\_TrustEngine\_Technical\_Reference\_Manual\_10103001\_0101\_00\_en.pdf>

## OTP

请参考 < PSRT\_022LZ07D\_B23A\_Application\_Note\_v1.0.pdf>

## 寄存器

请参考 <BK7236\_address\_map.xlsx>

# 原理分析

概述基本安全理论。

# 专利分析

## BL1

下述专利在是开发 BL1 过程中形成，其中 CRC 专利为之前技术。

### 抵抗故障注入攻击方法

请参考相关专利。

### 安全启动补丁

请参考相关专利。

### 深睡眠启动

请参考相关专利。

### FLASH CRC

请参考相关专利。

## BL2

# 方案概述

## 方案选型

# BL1 设计

BL1 基于山海的 BootROM，本节重点关注 BK7236 特有部分，山海原生 BL1 可参考 < IPSS\_Class\_B\_Software\_Integration\_Manual\_10370047\_0102\_00\_en>。

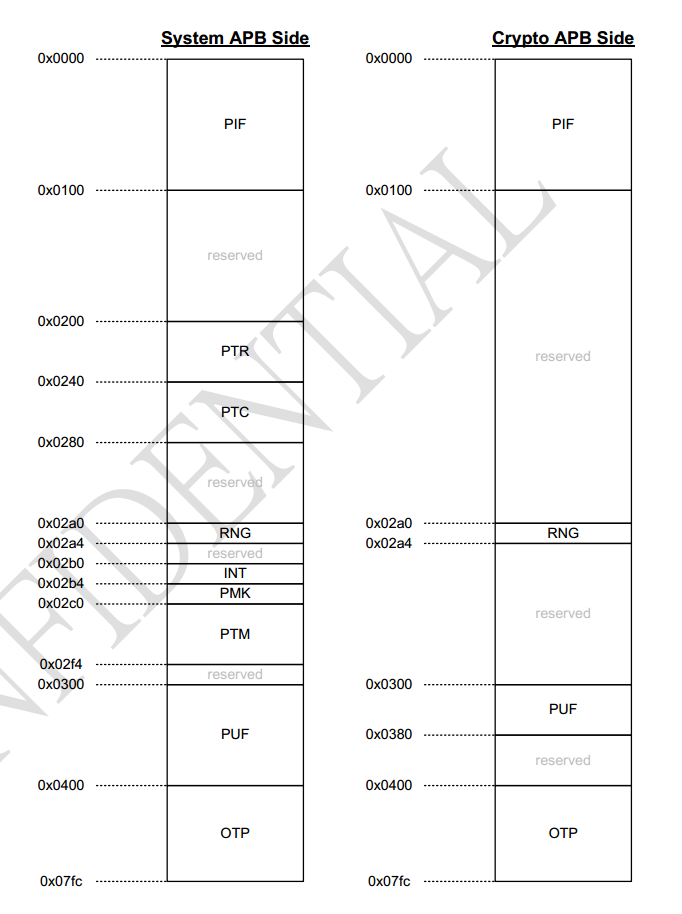
## BL1 移植

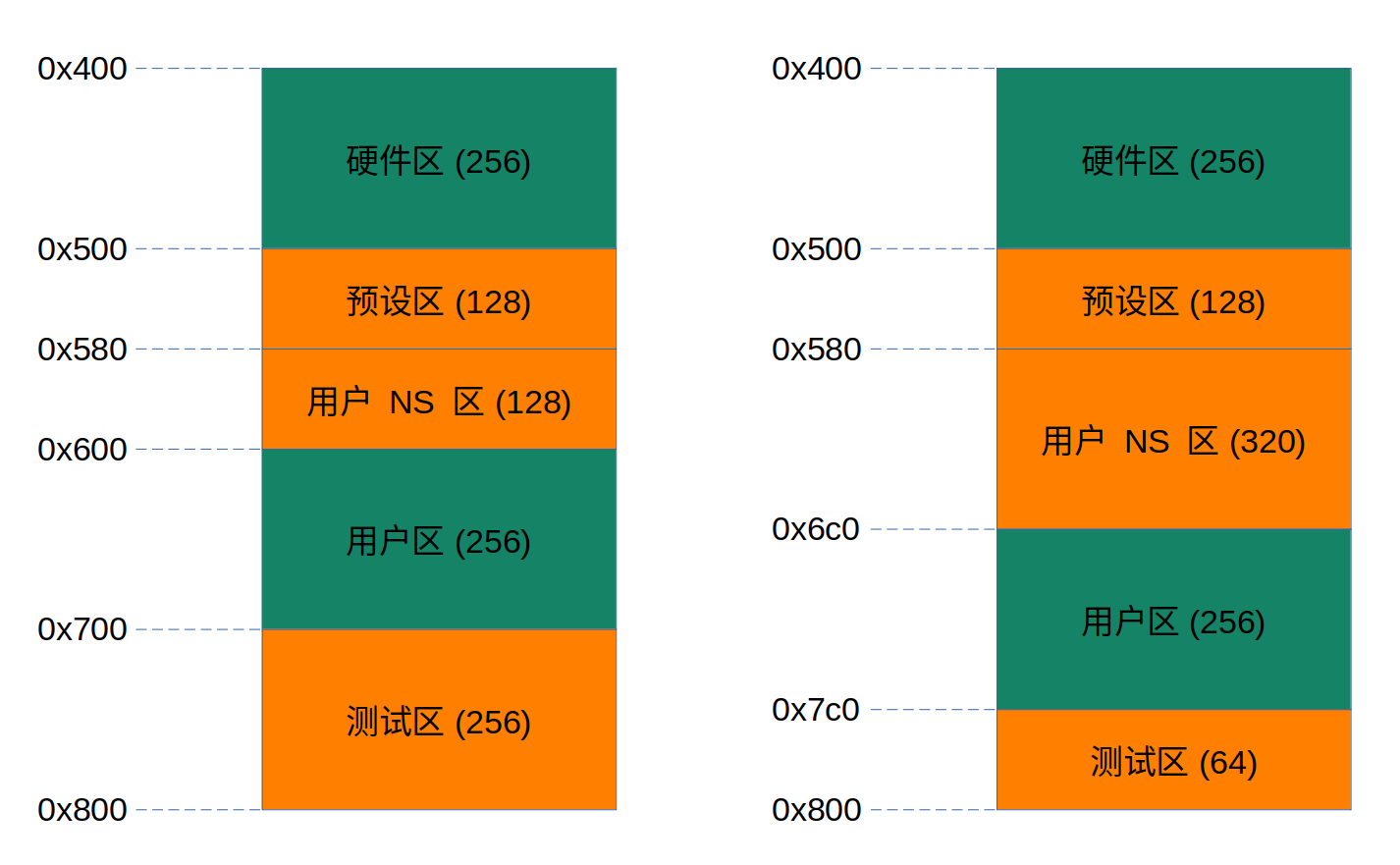


## OTP 设计

### OTP 分区

OTP数据区总共8Kbit，其中0x400~0x4FF给硬件使用，0x500~0x7FF 给软件使用。





### OTP 硬件区



其中硬件划分为3部分：

1. 0x400~0x45F 存放memcheck结果
2. 0x460~0x47F 存放flash\_key
3. 0x480~0x4FF 预留给硬件使用。

#### Memcheck

Memcheck地址范围为：0x4b100400~0x4b10045f

每个纠错地址占用2byte,96byte 可存放48个纠错地址。

列举如下：

|  |  |  |
| --- | --- | --- |
| memcheck模块地址 | 所属模块 | 地址位宽（包括地址有效指示） |
| 0 | USB | 12 |
| 1 | H264\_0 | 15 |
| 2 | H264\_1 | 15 |
| 3 | sharememory0 | 15 |
| 4 | sharememory1 | 15 |
| 5 | sharememory2 | 16 |
| 6 | sharememory3 | 16 |
| 7 | sharememory4 | 16 |
| 8 | cpu0\_0 | 12 |
| 9 | cpu0\_1 | 12 |
| 10 | cpu0\_2 | 13 |
| 11 | cpu0\_3 | 13 |
| 12 | cpu0\_4 | 13 |
| 13 | cpu1\_0 | 12 |
| 14 | cpu1\_1 | 12 |
| 15 | cpu1\_2 | 13 |
| 16 | cpu1\_3 | 13 |
| 17 | cpu1\_4 | 13 |
| 18 | wifi\_0 | 13 |
| 19 | wifi\_1 | 13 |
| 20 | wifi\_2 | 13 |
| 21 | enc | 13 |

上图中，通过总线可以按上图所示“memcheck模块地址”读出各个模块的纠错地址。每个“memcheck模块地址”可以读出32bit纠错地址数据，高16bit和低16bit各存一个纠错地址，纠错地址格式为：{地址有效指示，纠错地址}。地址有效指示：用于指示纠错地址是否有效，1为纠错地址有效。

#### FLASH Key

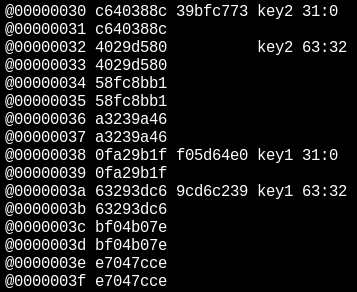
Flash enc key:

Flash enc\_key地址范围为：0x4b100460~0x4b10047f

对照下图OTP 仿真bin文件，0x30: key2[31:0] 对应的APB地址为0x4b100460

otp aes address:

|  |  |
| --- | --- |
| 30 | key2 31:0 |
| 31 | key2 31:0 |
| 32 | key2 63:32 |
| 33 | key2 63:32 |
| 34 | key2 95:64 |
| 35 | key2 95:64 |
| 36 | key2 127:96 |
| 37 | key2 127:96 |
| 38 | key1 31:0 |
| 39 | key1 31:0 |
| 3a | key1 63:32 |
| 3b | key1 63:32 |
| 3c | key1 95:64 |
| 3d | key1 95:64 |
| 3e | key1 127:96 |
| 3f | key1 127:96 |



3.XTS\_AES\_encrypt, if

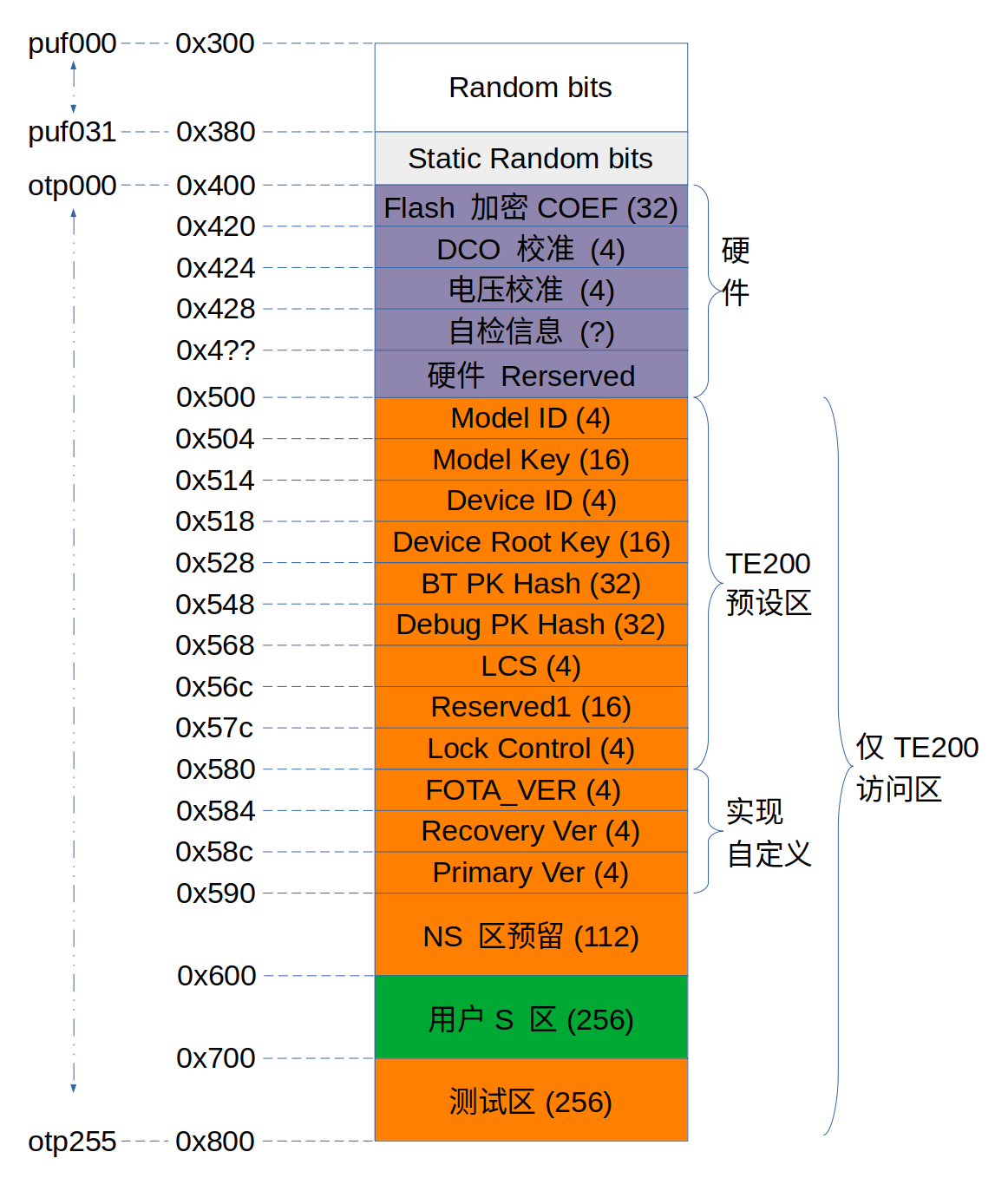
key1=128'hcaea65cdbb75e9169ecd22ebe6e54675,

key2=128'h10a58869d74be5a374cf867cfb473859,

the format is

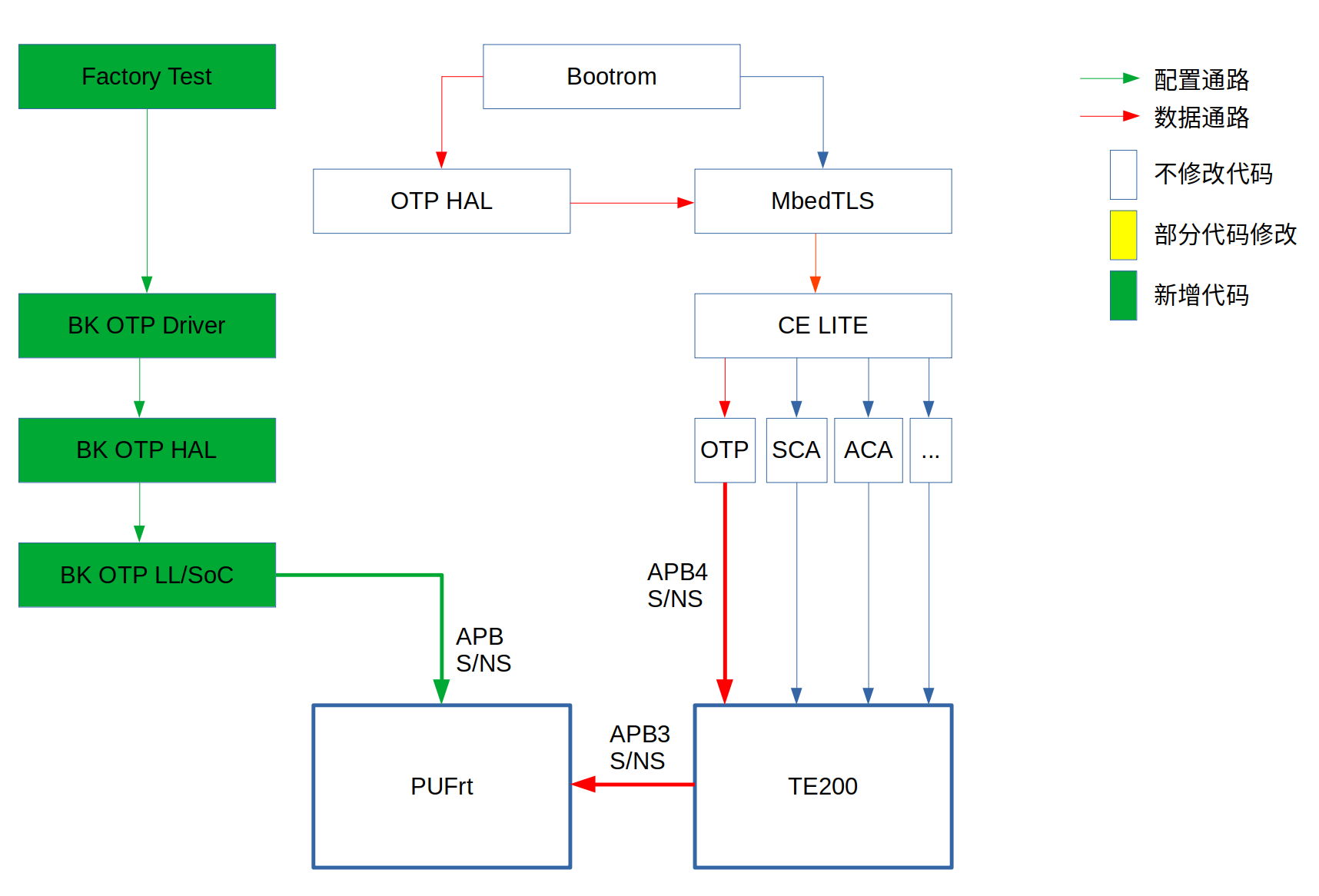
XTS\_AES\_encrypt.exe code.bin caea65cdbb75e9169ecd22ebe6e5467510a58869d74be5a374cf867cfb473859 0

### OTP 软件区



Recovery/Primary Version 指向同一 OTP 地址，即 offset 为 0x584 处，用于处理 BL2 升级时版本降级攻击。

### OTP 访问



访问方案如上图所示：

* 仅 Factory 测试 CPU 会直接访问 OTP
* 其他部分软/硬件完全保持不变

注意 BL2/TFM 访问 OTP 方式与 BL1 相似，通过 TE200 来访问 OTP S 区。

### OTP 权限控制

|  |  |
| --- | --- |
| OTP 区 | 控制方式 |
| OTP 软件预设区 | 由 TE200 硬件来控制，参考 < Shanhai\_TrustEngine\_Technical\_Reference\_Manual> A2.1.4 节了解细节。 |
| OTP S/NS 区 | 由 TE200 的 Lock Control 来控制，可以在 BL2 或者 TFM 中来控制。 |
| OTP 硬件 | 加密字等有两种控制式：   1. SoC 层面可以将 CPU 访问 OTP 通路完全锁死。 2. PUF 也提供了权限控制通路。 |

## EFUSE 设计

仅列出与 BootROM 相关的 EFUSE 设计。

|  |  |  |
| --- | --- | --- |
| BIT | 字段 | 描述 |
| BIT0 | Security boot enable | 置位安全启动使能，必须走验签。 |
| BIT1 | SB debug mode | 置位关闭所有打印。 |
| BIT2 | Fast boot | 置位则快速启动关闭，深睡眠醒过来之后始终走安全启动流程。 |
| BIT3 | Bootrom mode | 置位则走安全启动模式。 |
| BIT4 | XTAL | 置位则使能 PLL |
| BIT29 | FLASH AES | 置位则 FLASH AES 加密使能 |
| BIT30 | SPI download disable | 置位则禁用 SPI 下载模式。 |
| BIT31 | JTAG disable | 置位则 JTAG disable。 |

## FLASH 布局

### MPW1



### MPW2



### MPW3



## FLASH 读写

### 实际实现

|  |  |  |
| --- | --- | --- |
| 版本 | Manifest | 指令 |
| MPW1 | 数据 | 指令 |
| MPW2 | 指令 | 指令 |
| MPW3 | 数据 | 指令 |
| 量产 | 数据 | 指令 |

### 方案选型

|  |  |  |  |
| --- | --- | --- | --- |
| 方案 | 方案描述 | 优点 | 缺点 |
| 方案一 | 先对 BL2 加 CRC/加密，然后山海工具加签。山海内部全部走数据口。 | 对加 CRC/AES 之后的 BIN进行了完整性验证，确保访问时不会因为 CRC 错误卡死。  BootROM 内部不需要考虑指令/数据口访问问题，简化了逻辑。 | 如果 BL2 走 XIP 方式，P/S BL2 要使用不同地址编译，OTA 管理复杂。  如果 BL2 走 Load 方式，则实现较为复杂，因为山海内部会先将 BL2 加载到 RAM 再做验签处理，此时 RAM 里边的内容是带了 CRC 的。 |
| 方案二 | MANIFEST/BIN 全部走指令。 | 内部逻辑相对方案三略微简单。 | MANIFEST可能不对，导致CPU卡死。 |
| 方案三 | MANIFEST走数据，BIN走指令。 | 不存在 MANIFEST卡死的问题，也不存在方案一中 CRC/AES 或者OTA 升级麻烦问题。 | 如果 BL2 的 CRC 出错，则 CPU 会卡死。 |

量产版本中选择了方案三，因此后续 OTA 升级时需要对 带 CRC 的 BL2 完整性做验证，验证完之后，BL2 这个 FLASH 要防止被更改。

### BL1 FLASH 免写

为避免引入兼容性问题，BL1 中避免了 FLASH 写操作。原山海中更新 Boot Flag 操作被挪到 BL2 中实现。

## 启动流程



### 与山海启动差异点

差异点如下：

1. 增加深度休眠处理。
2. 增加 UART 下载处理。
3. 增加 HW/SW FIH 处理。
4. Secure Boot 使能时，即使 Manifest 中设置 Secure boot 为 False 也不生效。
5. 增加 Boot Flag 在 BL2 中的配置与调整。
6. 增加 LCS 在 BL2 中的配置与调整。
7. 通过 Boot Flag 来支持 BL2 升级。

### Boot Mode 寄存器

当配置 Boot Mode 为 1 时，硬件会执行下述操作：

1. 触发系统 Reset
2. 自动跳到 Flash 0 地址执行
3. 掉电 ROM

7236 的实现如下：

1. Flash 0 地址里写 TFM BIN 地址
2. BootROM 中仅在 Deep Sleep RST 时设置 Boot Mode，在普通 RST 时，软件直接跳转到 RAM 中的 bootloader。
3. 在 Bootloader 跳转到 TFM 前，设置 Boot Mode，触发系统 RST 从 flash 0 地址执行，触发 ROM 掉电。

Control 分区前 8 个子节为 BootLoader Entry Address。

### 启动流程

#### 首次上电启动

从 BootROM 上电后，硬件 boot\_mode 寄存器为 1, 软件对 primary boot path 进行校验。 校验完成之后，软件设置 boot\_mode 到 0， 并跳转到 Bootloader Entry 地址处。

#### 深度休眠启动

Deep Sleep 醒来之后，因为 boot\_mode 寄存器为 0， 硬件从 flash 0 地址处执行。

Deep Sleep 醒过来之后 RAM 信息均丢失，因为 flash 0 地址处必须放置 flash 地址，即 TFM 地址。 显然，control code 未经验签名，无法保证安全性，但既然选择了 fast boot 模式，本身也无安全可言。 未来要增加一个 Efuse 选项，用于走 deep sleep 安全启动模式。

#### OTA RST

OS 下载完之后，设置 boot flag 为 R，reset 进入 bootrom，对 R Bootloader 进行验证， 验证完成之后，跳转到 R bootloader entry； 在 R bootloader 中验证完 image 之后， 更新 Control Partition 中的 start entry 地址, 然后设置 boot\_flag 为 P。

还有一种做法是，每次在验证完成之后，如果发现 entry address 与实际不同，就更新.

#### Secure Debug RST

TBC。

## 启动性能



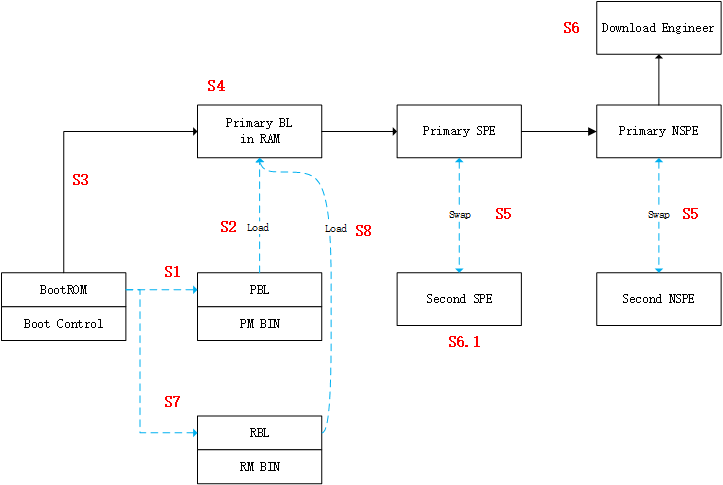
详情参考[BL1测试](#_BL2_用例)。

## FIH 攻击

这块可参考专利。

## BL2 升级

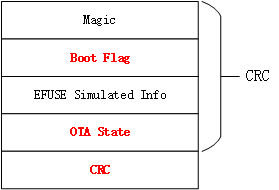
### 概述



升级场景说明

* S1 - 初始时，boot\_flag 为 P，BR 对 PBL 验签
* S2 - BR copy PBL 到 ram 地址执行，跳 PBL, **在 BR 中设置 boot\_flag 寄存器为 P**
* S3 - BR jump 到 Ram BL
* S4 - Ram BL 校验 BC，使用寄存器值修正 boot\_flag，如果 boot\_flag 与 BC OTA 信息不一致，打出 warning info
* S5 - 如发生了升级，RAM BL 对 SPE/NSPE 两个 slot 做 swap
* S6 - Armino Download 引擎将调用 SPE 接口启动升级
* S6.1 - SPE 检查 BC，关闭 R Manifest 与 RBL partition 写保护，返回 RAM
* S6.2 - Armino Download 引擎下载 manifest.bin 与 RBL
* S6.3 - Armino Download 引擎调用 SPE 接口，更新 BC，boot flag，以及 CRC, 重起
* S6 - BR 验签 RBL，copy RBL 到 ram 地址执行，跳 RBL，完成升级与 boot path 切换

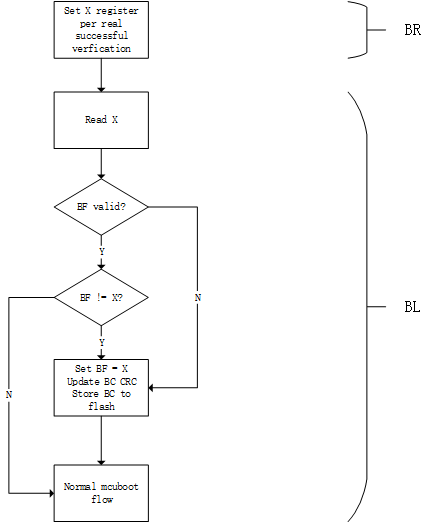
### BC 设计



如上图所示:

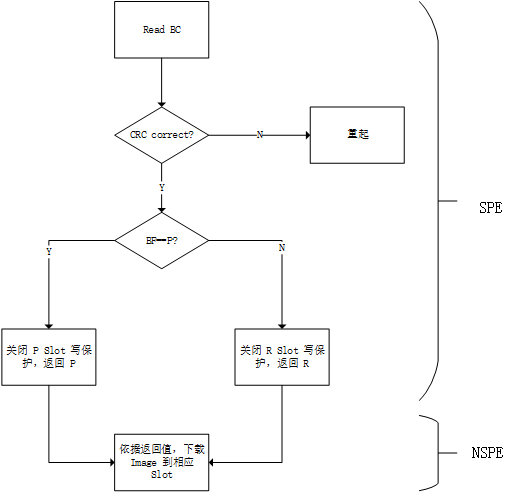
* BF - 即为 boot flag，然后山海 boot flag，可为 P or R
* OTA State - 用于指示 OTA 状态，主要用于检查 boot\_flag 与实际运行的 BL 不一致时打出 debug info 如，可设计为：P\_OTA\_DOWNLOADING, P\_OTA\_DOWNLOADED, R\_OTA\_DOWNLOADING, R\_OTA\_DOWNLOADED
* CRC - 对整个 BC 做 CRC
* 为进一步提升 BC 的可靠性，可考虑将 BC backup 起来，但当前还没有使用 backup 的充分理由
* 在当前方案中，如果升级的是 BL，则 Image 与 Manfiest 会被直接下载到 P/R slot，是基于 BF 来决定的，因此， 在 Download 阶段要确保 BL 值正确。

### BL2 BF 调整策略



其中 X register 设置涉及到 BR 变更。

### OTA Slot 选择



### 异常场景说明

#### RS1 - 下载失败

下载失败不会更改 BC 任何值，不影响下次启动。

#### RS2 - 下载了非法的 Manifest.bin[ℑ](file:///\\192.168.0.30\exchange\ming.liu\vnc\vnc22\bk7236\_build\html\design\bl2\bl2_ota_solution.html#rs2-manifest-bin)

BR 验签会失败，BR 会选择旧的 BL，具体可分两种情况分析：

* 升级 PBL 时，boot\_flag 被 Download 改成 P，但 P 验签失败，则 BL 中将 boot\_flag 调整成 R
* 升级 RBL 时，boot\_flag 被 Download 改成 R，但 R 验签失败，则 BL 中将 boot\_flag 调整成 P

#### RS3 - 下载了非法的 BL

同 RS2

#### RS4 - Manifest.bin 合法，但版本不正确

同 RS2。

#### RS5 - BL 合法，但版本不正确

同 RS2。

#### RS6 - 初始烧录 BC 值非法

在 BL 中调整为正确值。

#### RS7 - SPE 更新 BF 时掉电

分两种情况：

* 升级 P 时掉电，BF 未被更新，则重起后，BR 从 R 启动，升级失败
* 升级 P 时掉电，BF 已成功被更新为 P，则此时 P 的 download 已完成，BR 会优先去验证 P，升级成功
* 升级 P 时掉电，BF 已成功被更新为非法值，BR 走 P，升级成功
* 升级 R 时掉电，BF 未被更新，则重起后，BR 从 P 启动，升级失败
* 升级 R 时掉电，BF 已成功被更新为 R，则此时 R 的 download 已完成，BR 会优先去验证 P，升级成功
* 升级 R 时掉电，BF 已成功被更新为非法值，BR 走 P，升级失败

因此，在下载完成后但 BC 更新失败情况下，升级可能成功，也可能失败。但无论哪种情况，都会在 BL 中修正 BF。

#### RS8 - Image 被破坏时

分两种情况:

* 只有一个 BL，则当 BL 被破坏时，则再也无法恢复，除非通过 uart download
* 有两个 BL，则当其中一个被破坏时，则会走另一个，此时 BF 会在 BL 中被修正

## 验签失败处理

## IMAGE 打包

本节主要描述在 BL1 开发阶段使用的打包流程，量产版本中需要使用新的 partition 管理工具。

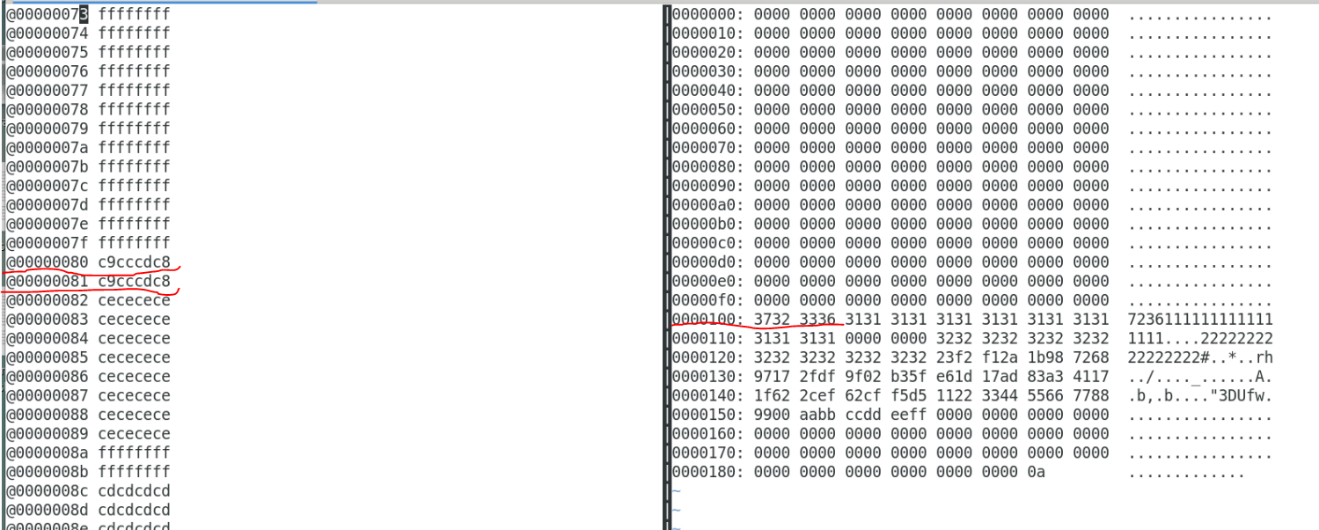
### 概述



功能如下：

1. create - 生成一个 image
2. dump - 以文本形式 dump image
3. crc16 - 给 image 增加 crc16, 待完成
4. aes - 给 image 增加 AES 待完成
5. merge - 依据 jason 配置生成 image 或者 merge image
6. asii - 生成仿真时用到的单字节流

### 仿真 OTP 生成



如上图所示，右则为原始 otp.bin，左则为仿真用要求的 otp.txt。转换关系如下：

1. otp.bin 中每 4 字节为一读取单元，在 otp.txt 中生成一模一样的两行
2. otp.txt 中字节只序与 otp.bin 中相反，且每 bit 取反

### Public Key 解析

首先使用 openssl 从 .pem 中解析出 .DER，如下所示。 然后取前 65 字节计算 hash 值。 在 debug 时通常只要找到大概位置就能猜出 public key 在 manifest 中的位置

## 版本管理

|  |  |  |
| --- | --- | --- |
| 芯片 | TAG | 关键改动 |
| MPW1 | v1.0 | * 适配山海 BootROM * MANIFEST走DBUS * 增加 OTP FLASH 模拟 * 增加 FLASH Control 控制 * Port Armino 驱动 * 。。。 |
| MPW2 | v2.0 | * MANIFEST 走 CBUS * 支持 SW FIH * 支持HW FIH * 支持 26/40M XTAL * BootROM 减小至 64K * 去掉 Scramble 代码 |
| MPW3 | v3.0 | * MANIFEST 走DBUS * 0x100 处增加 BK7236 magic code * 调整 flash layout: 将 primary manifest/bl2 放到 recovery 之前 (与 bl2 保持一致） * MSP Stack 由 1024 调整到 10240 * 增加对 dead loop 检测：   + PLL 写时 poll sys 0xB 寄存器，loop 10200 次 (约 60ms)   + EFUSE 读时，loop 102000 次 (约 600ms) * 汇编全部改 C，并调整启动流程:   + deep sleep (PMU\_AON/EFUSE/PRRO 判断)   + PLL 使能   + Load * 4. main |
| 量产 | v4.0 | * OTP NS size 由 256 改成 128 * LCS 读版本号的方式由数 OTP 中配置值中 bit 1 的个数，改成数连续 bit 1 的个数。这样，在 MPW1/2/3 中，版本号 0x00000F0F 得到的版本号为 8，在 MPW4 中为 4。 * Primary/Secondary Manifest Version 使用同一个 OTP 版本号，用于支持 BL2 升级时 anti-rollback，OTP size 为 4 |

## 待完成项

下面相关工作需要在 BL2 中完成：

1. BOOT FLAG 处理。
2. 对 LCS 的 double check。
3. OTP S/NS 权限配置。

## 风险点

# BL2 设计

TBC by 曾鑫。

## 方案概述

## BL2 升级

### 版本降级攻击

在 BL1 中 Primary/Secondary 分别维护了一个版本号，而 Primary 与 Secondary 的版本不会交叉检验，**这是当前 BL1 中的一个漏洞**。因此需要在 OTA 升级前确保先做交叉检验!!!

BL1 中为 Primary/Secondary BL2 的版本号存放方式如下表所示：

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 类型 | OTP offset | OTP size |
| MPW1~MPW3 | Primary | 0x4b100589/137 | 4 |
| Secondary | 0x4b100589/136 | 1 |
| MPW4 | Primary | 0x4b100588/136 | 4 |
| Secondary | 0x4b100588/136 | 4 |

其存放内存的二进制 1 的个数表示版本号。

## APP 升级

## BL2 移植

## 差分升级

## 安全调试

# 安全部署

TBC by 曾鑫。

# 工具

TBC by 曾鑫。

# 示例设计

# 测试

## 配置

### EFUSE

#### 方法一：CLI 配置

步骤：

1. 先在 legacy download 模式下烧录 Armino
2. 使用 EFUSE cli 命令将 EFUSE bit3 置位：efuse\_test write 0 8

#### 方法二：HID Download Tool SPI 配置

待完成。

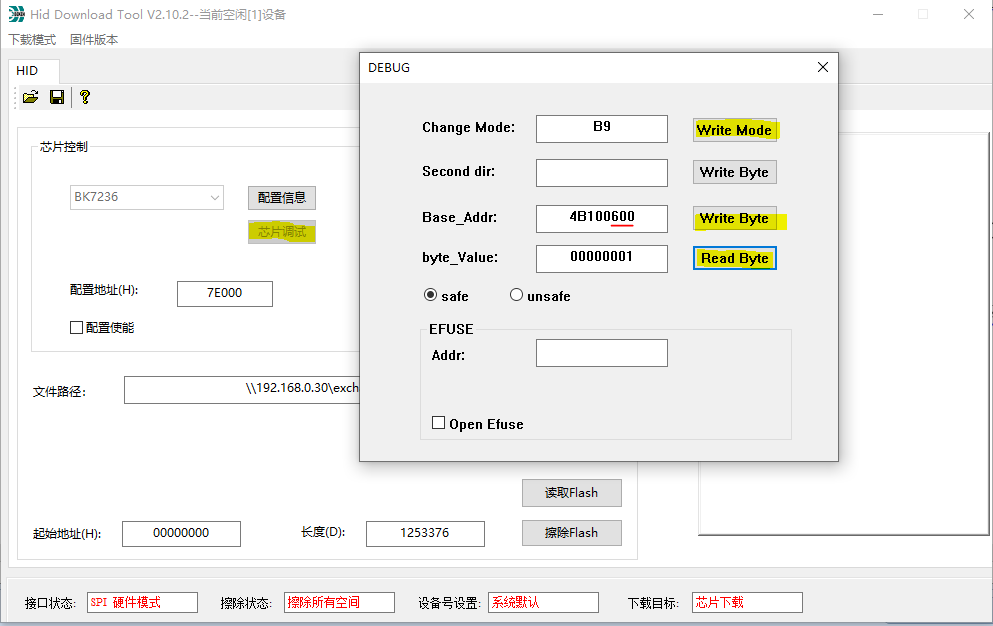
#### 方法三：脚本批量配置

BK7236 育强尚未完成。

### OTP

#### 方法一：HID Download Tool SPI 配置

1. 使用最新SPI 下载工具 HID Download Tool 配置 OTP，其中版本V2.10.2 验证可用。
2. 需要确保SPI 硬件模块已经烧录了育强支持 BK7236 OTP SPI 写的固件，否则读写会有问题。
3. 使用 SPI 硬件烧写方式。
4. 为避免SPI 读写导致 watchdog，读写前先点 Write Mode之后再读写。



#### 方法二：脚本批量配置

BK7236 育强尚未完成。

### FLASH AES KEY

#### 加密工具加密

加密命令如下：

${AES\_TOOL} encrypt -infile ${INSTALL\_DIR}/app.bin -keywords 73c7bf397f2ad6bf4e7403a7b965dc5ce0645df039c2d69c814ffb403183fb18 -outfile ${INSTALL\_DIR}/app\_raw\_xts.bin -startaddress 0

${CRC\_TOOL} -crc ${INSTALL\_DIR}/app\_raw\_xts.bin

AES key: **(high) 73c7bf397f2ad6bf4e7403a7b965dc5ce0645df039c2d69c814ffb403183fb18 (low)** 将 KEY 分成 KEY1/KEY2 两部分:

1. KEY1 - 73c7bf397f2ad6bf4e7403a7b965dc5c
2. KEY2 - e0645df039c2d69c814ffb403183fb18

每个 KEY 有四个字，因此，加密工具使用的 KEY 为:

1. KEY1.word1 - 73c7bf39
2. KEY1.word2 - 7f2ad6bf
3. KEY1.word3 - 4e7403a7
4. KEY1.word4 - b965dc5c
5. KEY2.word1 - e0645df0
6. KEY2.word2 - 39c2d69c
7. KEY2.word3 - 814ffb40
8. KEY2.word4 - 3183fb18

#### OTP 烧录

FLASH AES KEY 存放在 OTP 0x4b100460~0x4b100480 处，共 32 Bytes。 通过 SPI 工具烧录。

烧录的地址与 KEY words 对应关系如下表所示，加密工具输入的 KEY 与 OTP 工具写入 密钥对应关系如下：

1. KEY LE - 即 KEY2 低地址，KEY1 高地址
2. WORD LE - 在 KEY1/KEY2 内部，按 word 小端排列，word4 放低地址。
3. WORD 内部，Byte 字节序不变

因此，烧录工具使用的 KEY，或者说 KEY 在 OTP 中的放置为:

1. KEY2.word4 - 3183fb18
2. KEY2.word3 - 814ffb40
3. KEY2.word2 - 39c2d69c
4. KEY2.word1 - e0645df0
5. KEY1.word4 - b965dc5c
6. KEY1.word3 - 4e7403a7
7. KEY1.word2 - 7f2ad6bf
8. KEY1.word1 - 73c7bf39

如下表所示：

|  |  |  |
| --- | --- | --- |
| OTP Addr | OTP value | Word |
| 4b100460 | 3183fb18 | key2.word4 |
| 4b100464 | 814ffb40 | key2.word3 |
| 4b100468 | 39c2d69c | key2.word2 |
| 4b10046c | e0645df0 | key2.word1 |
| 4b100470 | b965dc5c | key1.word4 |
| 4b100474 | 4e7403a7 | key1.word3 |
| 4b100478 | 7f2ad6bf | key1.word2 |
| 4b10047c | 73c7bf39 | key1.word1 |

### Public Key Hash 烧录

1. 在 $BOOTROM/\_build 下找到 puk\_digest.bin，里面包含要写向 OTP 的 Public Key Hash。
2. 逐字向 OTP 写 HASH 值，特别要注意字节序问题，如下图所示。

puk\_digest.bin 中的内容：.

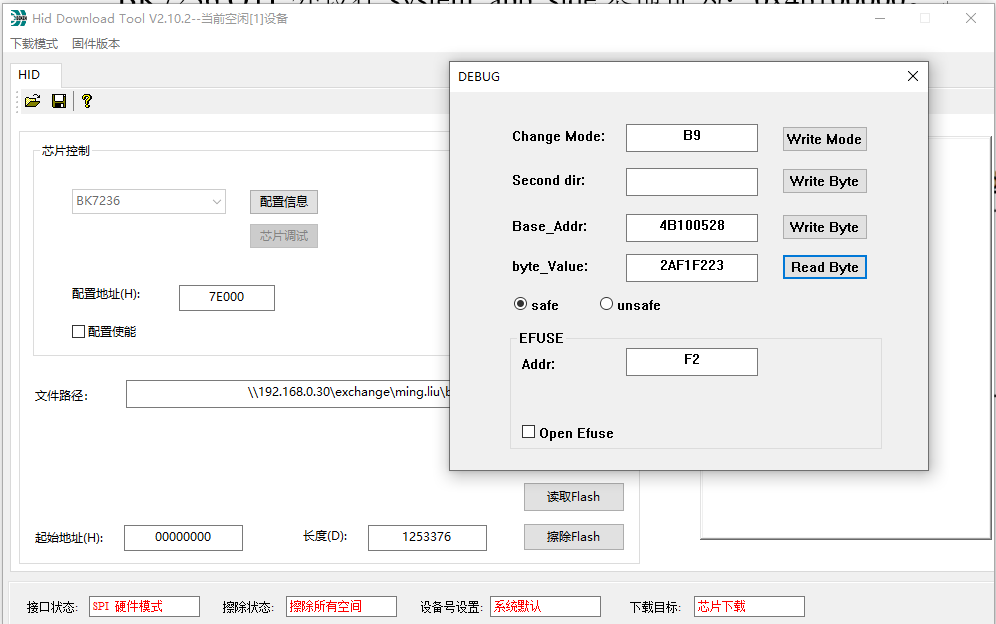
23f2 f12a 1b98 7268 9717 2fdf 9f02 b35f

e61d 17ad 83a3 4117 1f62 2cef 62cf f5d5

通过 SPI 工具写时对应关系：

|  |  |  |
| --- | --- | --- |
| OFFSET | puk\_digest.bin | 写入的内容 |
| 4b100528 | 23f2 f12a | 2af1f223 |
| 4b10052c | 1b98 7268 | 6872981b |
| 4b100530 | 9717 2fdf | df2f1797 |
| 4b100534 | 9f02 b35f | 5fb3029f |
| 4b100538 | e61d 17ad | ad171de6 |
| 4b10053c | 83a3 4117 | 1741a383 |
| 4b100540 | 1f62 2cef | ef2c621f |
| 4b100544 | 62cf f5d5 | d5f5cf62 |

OTP 写入截图示例：



HASH 要写入的内容较多，切记每写一个 word 都要进行反复确认：

1. 地址是否正确？
2. 值是否正确？
3. 写完之后读出
4. 上一字读写完成之后先清除地址与值，再开始写下一个字。

## BL1用例

安全启动测试用例及测试结果参考。

## BL1测试示例

本节仅针对 BL1 的开发与高度来介绍，芯片量产时，需要针对 BL1/BL2/TFM/Armino/CPU1/OTA 来写一份完整的量产配置文档。

### 注意事项

#### 测试顺序

BL1 的很多验证涉及 EFUSE/OTP 配置，这些配置都是不可逆的，因此当对测试流程不太清楚时，要充分考虑测试顺序，最好找一位同事将测试顺序评审通过了再着手真正的测试，以免因为测试步骤出错，导致芯片变砖。

下表是推荐的测试顺序：

|  |  |  |
| --- | --- | --- |
| 顺序 | 测试组 | 说明 |
| 1 | 使用模拟 OTP 验证 | 可以[在 FPGA 上跑 BL1](#_FPGA_上跑_BL1)，或者[芯片 FLASH 中跑 BL1](#_芯片FLASH_中跑_BL1) |
| 2 | 直接将 EFUSE/OTP 依赖配置固化 | 只要不是跑固化 BootROM，则可以直接更改代码的方式固化某些特定处理，如跳过 Legacy Boot 等 |
| 3 | 芯片 FLASH 中跑 BL1 验 OTP | 首次验 OTP 时，应该优先考虑 [芯片 FLASH 中跑 BL1](#_芯片FLASH_中跑_BL1)，因它可以在不使能 EFUSE 中安全启动 Bit 的前提下跑真正的 OTP，这带来的好处是，即使 OTP 配坏了，板子依然可用，因为 UART 下载模式不会使用 OTP。 |
| 4 | 通过 Manifest 使能 Secure Boot | 当要验证固化的 BootROM 时，先通过 EFUSE 使能 Secure Boot Mode，但不要通过 EFUSE 使能 Secure Boot，而是通过Manifest 来使能 Secure Boot。这样带来的好处是，如果 Public Key Hash 配置出错，板子依然可用（这时除了不验签，其他流程一样），EFUSE 中 Secure Boot 使能仅在量产前才打开。 |
| 5 | EFUSE 打开 PLL/FLASH AES | 这两项只要配置正常，不会有问题。 |
| 6 | OTP 权限 | 通过山海 Lock Control 来控制访问权限，一旦配置就不可逆向。 |
| 7 | EFUSE 关闭 Debug  EFUSE 禁用 FAST BOOT | 要注意，Debug 一旦关闭固化 BootROM 就不会有任何打印了，FAST Boot 一旦关闭，Deep Sleep 醒来之后就会一直走安全启动 |
| 8 | EFUSE Secure Boot 使能  EFUSE 禁用 SPI 下载 | 这几个配置仅在量产生阶段建议打开 |

#### FLASH Layout

特别注意不同 MPW 之间 Layout 不同，FLASH 使用指令口，数据口访问规则也可能不同，因此选择的测试方法要与打包/配置方法一致，否则就起不来。

#### Manifest 版本号

要注意，Manifest 版本号在 MPW3 做了修改，如果在同一块板子上固化 BootROM 与 FLASH 中 BOOTROM 验签的对象相同时，版本号更新会出问题，因此不要这么做。

Manifest 版本号处理是 BootROM 中唯一主动更改 OTP 内容的地方，因为 Manifest JSON 中的版本号要小新设置。

#### LCS

MPW1/2/3 的 LCS 非 0 时处理应该有 BUG，因此仅量产版本才允许将 LCS 改成非 0。

#### EFUSE 配置

当依赖于 CLI 来配置 EFUSE 时，一旦使能了 Secure Boot Mode，则仅当跳转到带有 CLI 的 Armino BIN 之后才能有机会通过 CLI 更改 EFUSE，因此在配置 EFUSE 事项时需三思。

### 芯片上跑传统下载

当安全启动未使能时，芯片默认走 UART 下载模式，团队内部开发阶段一般都选择传统下载模式。传统下载模式很简单，唯一要注意的点是下载的 BIN 在 0x100 偏移地址处必须加上 BK7236 字段，否则 BootROM 不会做跳转。

### 芯片上跑 BL1

本节以 MPW1 为例来介绍如何在芯片上跑 BL1，其他芯片类似。

在芯片上跑 BL1 指的是在 MPW或者量产芯片上跑芯片中固化的 BootROM中的安全启动。要求必须使能 Secure Boot 模式，并且烧写相应的 OTP。通常，如果是首次验证固化 BootROM，特别是当 EFUSE/OTP 烧写流程，如字节序等拿不准时，应该先偿试在芯片的 FLASH 中跑 BootROM，跑通了之后，再考虑跑固化在芯片上的安全启动。

推荐步骤：

1. 验证安全启动模式
2. 验证安全启动
3. 验证版本处理
4. 验证生命周期处理
5. 验证 PLL
6. 验证 FLASH 加密
7. 验证 FAST BOOT
8. 验证禁用 DEBUG
9. 验证禁用 SPI 下载（慎用，小心变砖）

#### S1 – 使能安全启动 – 无验签

##### S11 - 使能安全启动

配置 EFUSE bit3 使能安全启动，方法见 [EFUSE 配置](#_EFUSE_配置)。

##### S12 - 在 OTP 未用区域写 1

当 Secure Boot Mode 使能之后， BootROM初始化 OTP 时会检查 OTP 是否为全 0，如果为全 0则出错，因此至少得写一个 bit 为1。为避免影响正常 OTP，选择 0x600 offset 处烧录 1。烧录方法参考 [OTP 配置](#_OTP_配置)。

##### S13 - 编译 BL2.bin

当 BootROM 进入 Secure Boot Mode 且 Secure Boot 未使能时，BootROM 会走山海安全启动流程，要求被启动的下一级 BIN 按山海要求来打包，但此编译出来的 BIN 需要使用相关脚本打包。正常 BootROM 会启动 BL2.bin，因此需要使用山海工具对 BL2.bin 打包。为简化测试步骤，本次使用 BootROM 仓库中 hello world 程序当 BL2.bin，直接使用 BootROM 现成的打包工具进行打包。

另外，由于 EFUSE 中 Secure Boot 未使能，因此本次验证使用 BootROM 不会对被启动的 BIN 作验签（即不检验 manifest 中的 Boot Public Key 是否正确），因此不需要向 OTP 中烧录 Boot Public Key Hash。

步骤：

1. 下载 BootROM [最新代码](http://192.168.0.6/wangzhilei/bk7236_bootrom_s)， checkout 到 master 分支，代码根目录记为 $BOOTROM，
2. 编译 all.bin: b1.sh mpw1 verify，这种方式编译出来的 BIN 会使用 $BOOTROM/config/ec256\_sha256\_mpw1/config 下的配置来打包 all.bin，具体把包流程参考 $BOOTROM/build\_all.sh 脚本。
3. 通过 SPI 进行烧录，注意：当 SB 模式使能时，UART 下载已被禁用

##### S14 - 重启

烧录完 BL2.bin 之后掉电重起，如果看到打出 hello bootloader 则表示测试通过。

#### S2 – 验证安全启动 – 有验签

在 S1 的基础之上，使能 Secure Boot 就可以验证完整的安全启动流程了，具体步骤如下。

##### S21 – 使能 Secure Boot

有两种方式可以使能 Secure Boot:

1. 通过 Manifest 来使能



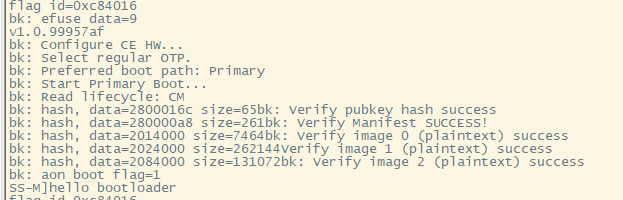
1. 将 EFUSE bit0 置1。

通过 EFUSE 使能的话就无法再关闭，因此在开发阶段，建议先通过 Manifest 使能，这样万一 Public Hash Key 配置不对，板子还能用。

##### S22 – 烧录 Boot Public Key Hash

具体烧录方法参考 [Public Key Hash 烧录](#_Public_Key_Hash)。

烧完 Public Key Hash 之后，当出现下述 Log 时表示验证通过：



#### S3 - 验证 Manifest 版本

##### S31 – 验证初始版本

将 Primary Manifest JSON 文件中版本号配置成 0x00000001，启动，然后使用 SPI 工具读取 0x4b100589 地址处的值是否被改为版本1，如果是，则验证通过。

##### S32 – 验证 OTP 写 PRIM 版本

步骤：

1. 在 OTP 0x4b100589 处写 0x02，此处要特别注意，因为 SPI 工具写时要求 4 字节对齐，因此应该向 0x4b100588 处写，写时要注意字节序问题。
2. 重启板子，将会提示版本号小于 OTP 中版本，会走 Recovery 启动。

##### S33 – 验证 PRIM 版本

步骤：

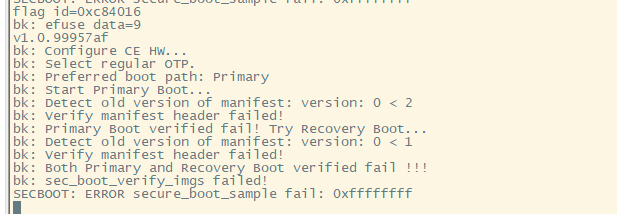
1. 通过 Primary Manifest JSON 将版本号提升到 2
2. 重启板子，则会走 Primary BL2

S34 – 验证 Recovery 版本

步骤：

1. 将 Primary manifest 版本设置为 0
2. 将 OTP Recovery 版本设置为 1
3. 将 Recovery Manifest 版本设置为 0

这时 Primary/Recovery 版本验证都会失败，启动失败，如下图所示：



#### S4 - 验证 PLL

通过 EFUSE bit4 使能 PLL 进行验证。

#### S5 - 验证 FLASH 加密

两个关键点：

1. 配置正确的 FLASH AES KEY，参考 [FLASH AES KEY 配置](#_FLASH_AES_KEY)。
2. 编译 AES 加密版本，使用： ./b1.sh xxx yyy 1 来进行编译，最后一个参数 1 表示加密。

#### S6 - 验证 FAST BOOT

##### S7 FAST BOOT使能

EFUSE 默认使能 FAST BOOT 模式，本用例可归到深度休眠用例中，已独立验证通过。FAST BOOT 关闭时尚。

##### S72 FAST BOOT 关闭

暂未验证。

#### S7 - 验证 LCS

LCS 在 MPW1/2/3 中有 BUG，要在 FLASH 中跑 BL1 来验证。

#### S8 - 验证禁用 DEBUG

将 EFUSE bit1 置位，如果没有任何 log 打出，且能跳转，则验证通过。当前未做验证。

#### S9 - 验证禁用 SPI 下载

特别提示：当 SPI 下载禁用之后，只能通过 OTA 进行升级，小心板了变砖!!!

##### S91 – 再次确认 SPI 下载可禁

通常 SPI 下载仅在量产版本上才可禁用，其他情况都不应该禁用 SPI 下载!

确实需要禁用 SPI 下载时，请确认：

1. 如果允许这块板子变砖，那随便禁用。
2. 板子已烧录 BIN 不支持 OTA，不应该禁用。
3. 板子已烧录 BIN OTA 不稳定，不应该禁用。
4. 板子已烧录 BIN OTA 不支持 BL2 升级，不应该禁用。
5. 还希望继续通过 SPI 下载，不应该禁用。

##### S92 – 禁用 SPI 下载

EFUSE bit30 置位禁用 SPI 下载，当前未做验证。

### 芯片FLASH 中跑 BL1

通常在 FLASH 中跑的 BL1 用于验证最新的 BL1，通常 BL1 有改动时就应该使用这种方式验证。

本示例用于在 MPW1 板子的 FLASH 上跑最新 BootROM，当我们需要在芯片上调试最新 BootROM 时要用到这种方法。

技术上，当芯片使能或者未使能 Secure Boot 模式时都可以将 BL1 搞到 FLASH 中去跑，但通常应该在未使能 Secure Boot 模式的芯片上去跑，这会避免很多麻烦，而且简单，安全，不容易出错。

S1 – 配置

在BootROM的 build/mk/config.mk 中有一个宏 FPGA，为 Y 时表示代码跑在 FPGA上，为 N 时跑在芯片上，当跑在芯片上时指跑在 0x06000000 的固化的 ROM 地址空间。在 FLASH 中模拟类似在 FPGA 上跑，因此 FPGA 置 Y。

步骤：

1. 配置
   1. FPGA = N
   2. SIMU\_OTP\_IN\_FLASH = N
2. 编译

使用 b1.sh mpw3\_fpga verify\_bootrom 编译

1. 下载

通过 uart 下载 all.bin 进行测试

当不使用真实 OTP 时，则将 SIMU\_OTP\_IN\_FLASH 置 Y。

### FPGA 上跑 BL1

在 FPGA 上跑 BL1 与在 FLASH 中跑 BL1 流程几乎一样，不同点在于 FPGA 上没有OTP，必须将 SIMU\_OTP\_IN\_FLASH 置 Y。

另外要注意，FPGA 的主频可能与芯片不同，如果不同，则应该手动通过代码进行更改。

### 芯片内存中跑 BL1

#### 芯片安全启动未使能

当芯片安全启动模式未设置时，方法与在 FLASH 中跑 BL1 类似。不同点是，FLASH 中 BL1 验签的 BIN 为要在内存中跑的 BL1，然后将 manifest 的 load address 设置成内存地址即可。这样，启动流程如下：

1. 固化 BootROM 走传统下载模式，跳转到 FLASH 中的 BL1。
2. FLASH 中的 BL1 进行验签，通过之后将被验证的 BL1 搬到内存中，然后跳转到内存中的 BL1。
3. 执行内存中的 BL1，内存中的 BL1 对自身进行验签。

当然，UART 下可只下载一个简单的程序，这个程序要做的事情就是将 BL1 搬到内存中然后再跑。

#### 芯片安全启动使能

安全启动使能时会更简单，直接将固化 BL1 验签的对象设置为要调试的 BL1 即可，固化 BootROM 与要调试的 BootROM 验同一个对象。但这会有 Manifest Firmware 麻烦，因此仅当不需要验证 Manifest Firmware才可选择这种方法。

### 仿真 BL1

编译仿真 BL1 版本时，FPGA 置 N，这也是默认值，也是 FPGA 唯一置 N 的场景。与芯片不同的点在于，它必须使用真实的 OTP（与 FPGA 相反），但 OTP 配置与芯片 OTP 配置不同。

#### OTP 配置

#### 编译

### TRNG 测试

#### 移植 trng\_calibration

trng calibration 测试时，设备其他功能需要关闭。BK7236 的做法是将 Shanhai-CB-SW-r1p2-00eac0 中 Bootloader/trng\_calibration 移植到 BootROM。

可用于 TRNG 校准的代码：<http://192.168.0.6/wangzhilei/bk7236_bootrom_s/-/tree/stable/verify_mpw_trng>

官方版本不好 debug，因为 trng calibration 代码为加扰代码，而 trng\_tool 也是以二进制提供， 再加上测试时会占掉 uart0 口，整个过程没有任何打印。verify\_mpw\_trng 版本增加了相应的 debug 机制，可通过 build/mk/config.mk 中的 LOG\_UART1 与 TRNG\_CAL\_DEBUG 来开关

#### 编译 trng\_tool

成功执行下述步骤之后，trng\_tool 就可以在 ubuntu 上使用了:

1. sudo apt install libdivsufsort-dev libbz2-dev
2. 将 Shanhai-CB-SW-r1p2-00eac0 copy 到 ubuntu
3. cd Shanhai-CB-SW-r1p2-00eac0
4. ./build.sh -t libassessment - 编译 libassessment.so
5. ./build.sh -t tools - 编译 trng\_tool
6. sudo LD\_LIBRARY\_PATH=./ ./trng\_tool -c trng\_gather.json

编译 libassessment 时会遇到 “FOTA Download Agent Customer Configuration Failed” 错误, 咱们没有 FOTA，依据提示随便配置一下就好。

#### 测试概述

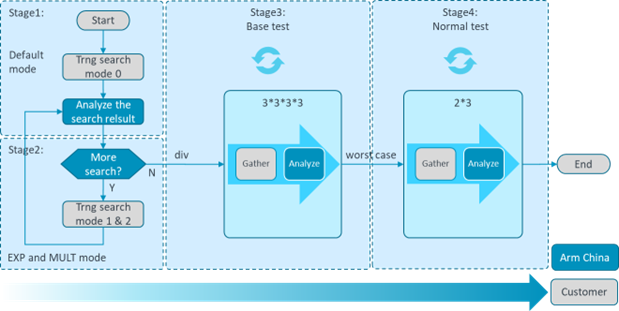
测试分三大步，由博通集成与 Arm 一起完成:

1. S1 - Search Test - 找出最小 sample clock divider。
2. S2 - Base Test - 测试在各种条件下随机数质量
3. S3 - Normal Test - 极端条件下完整随机数质量

不同的测试主要依据下述条件归类:

1. Inverter chain group selection - 取 1~15
2. 电压 - 可取 高，常压，低
3. 温度 - 可取 高，常温，低
4. Process corner - 可取 Fast(FF), Typical(TT), Slow(SS)

参考下图，图中 stage1/2 对应 S1。



#### S1 - Search Test

目的: 找出最小 sample clock divider。

测试条件: 常温, 常压, typical process corners

依据 sample clok divider 可细三个模式:

1. default (mode 0) -
2. EXP - divider step 为指数递增
3. MUTI - divider step 线性递增

测试命令: sudo LD\_LIBRARY\_PATH=./ ./trng\_tool -c trng\_search.json

S1 要做两轮测试，先依据 Arm 给的配置文件做第一轮，拿到数据之后给 ARM 分析， ARM 基于分析给出第二份 json 配置，同样的方法测试，测完之后反馈数据到 ARM。ARM 基于这两轮数据，得到 divider 值。

据 ARM 反馈，bk7236 的 TRNG Entropy 挺好的，divider 为 10。

#### S2 - Base Test

这轮主要基于电压(H/L/T)，温度 (H/L/T)，process corner (FF/SS/TT) 以及三组 div (10/15/23)， 来测试，一共有 3\*3\*3\*3 = 81 不同的配置，每个配置需要在指定的条件下进行测试。

最后将测得的数据反馈到 ARM，他们进行分析。

温度： [-40, 125] 电压：[0.7, 1.0] (正常 0.8, 0.9)

#### S3 - Normal Test

主要是在 T\_T\_TT 条件下选择多块（3 块以上）板子做不同的测试。

#### 结论

ARM 反馈的结论如下：

1. entropy 很好
2. 分频配置为 div10

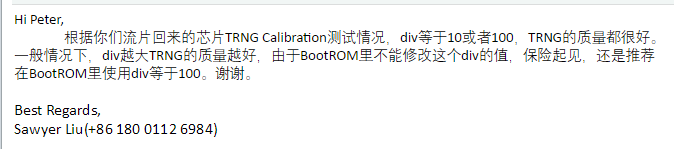
#### 相关资料

TRNG 相关的资料可参考:

1. <Shanhai-CB-HW-r1p1-00eac0/Shanhai\_Hardware\_Application\_Note\_10103005\_004\_en.pdf>
2. <Shanhai-CB-SW-r1p2-00eac0/IPSS\_Class\_B\_Software\_Integration\_Manual\_10370047\_0102\_00\_en.pdf> 3.8.2 与 4.4
3. Shanhai-CB-SW-r1p2-00eac0/README - 里面有编译方法介绍

#### BL1 中配置

ARM 测试得到最佳值为 10，BootROM 中当前值为 100，关于是否要修改成 10，下面是 ARM 工程师 Sawyer 反馈：



## BL2 用例

TBC by 曾鑫。

## BL2 测试示例

TBC by 曾鑫。

### BL1 -> Armino S

### BL1 -> TFM -> Armino NS

### BL1 -> BL2 -> TFM

### BL1 -> BL2 -> TFM -> Armino NS

### BL1 -> BL2 -> TFM -> Armino NS -> CPU1

### OTA – Armino

### OTA – Armino + TFM + CPU1

### OTA -> BL2 + Armino + TFM + CPU1

### OTA – CPU1

### OTA – BL2

### Debug

### 量产版本部署

# 成本

# 可移植性

# 风险，问题与优化

# 其他