眸芯科技（上海）有限公司
Molchip Technology（Shanghai) Ltd.

安全启动方案

文档版本　00B04

发布日期　2020-11-12

商标声明

 ，MOLCHIP，眸芯和其他眸芯商标均为眸芯科技（上海）有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

担保声明

您购买的产品、服务或特性等应受眸芯科技商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，眸芯科技对本文档内容不做任何明示或默示的声明或保证。由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

保密

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

前 言

概述

本文档安全启动设计方案。

读者对象

本文档（本指南）主要适用于以下工程师：

- 技术支持工程师
- 软件开发工程师

产品版本

产品名称	产品版本
	V100

修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

Date 日期	Revision Version 修订版本	Revision Notes 修订说明
2020-11-12	00B04	第 4 版本

目 录

1 安全启动介绍

安全启动支持普通安全启动和加密安全启动；其差异在于普通安全启动镜像中的数据，RSA Signature，RSA Public Key 为明文；加密安全启动镜像中的数据，RSA Signature，RSA Public Key 为密文；

1.1 普通安全镜像

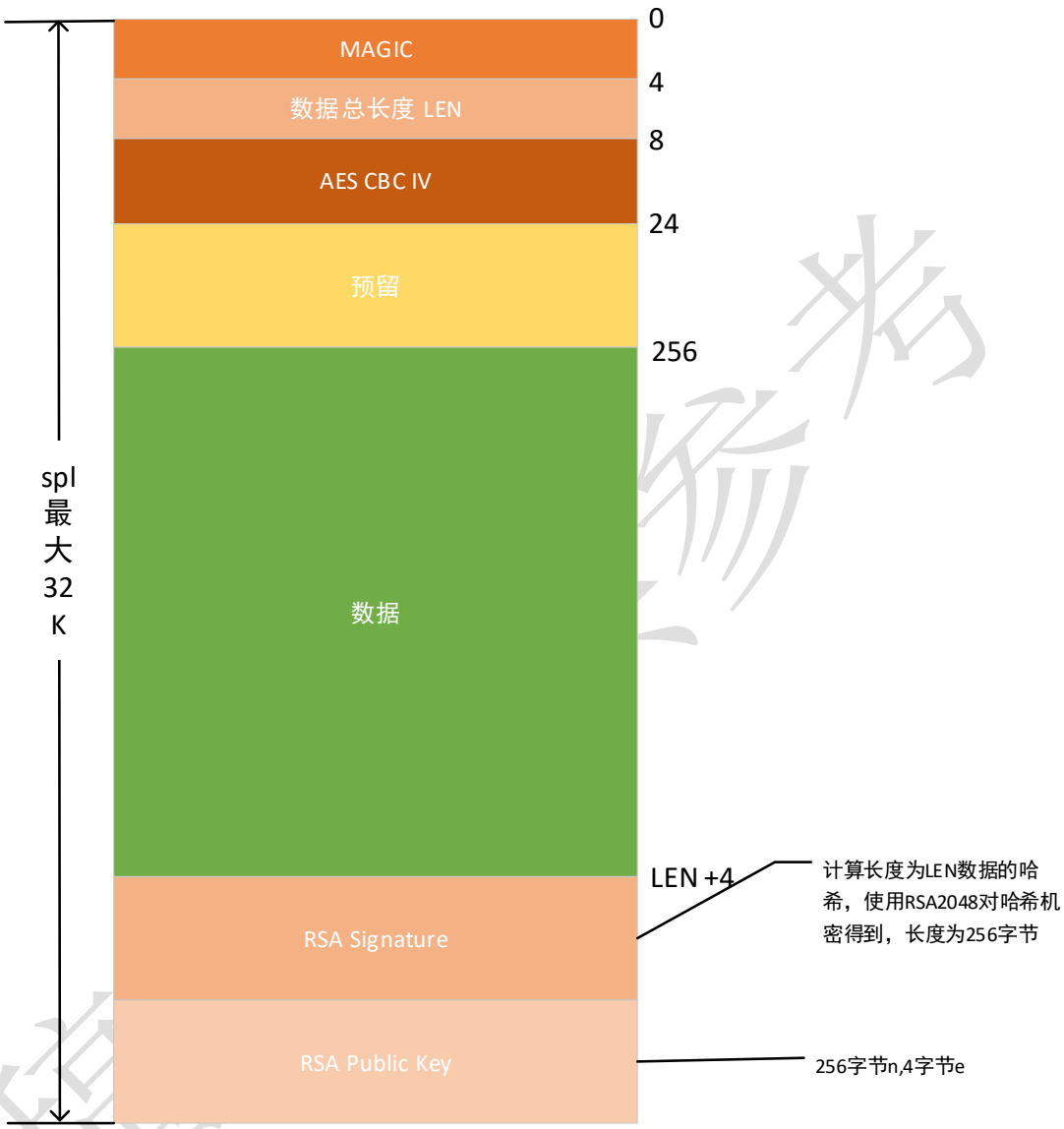

图 1-1 普通安全镜像

普通安全启动 SPL 镜像有 SPL 数据，RSA 签名和 RSA 公钥组成；

其中，RSA 只支持 RSA2048，AES CBC 的 IV 值为 0；

1.2 加密安全镜像

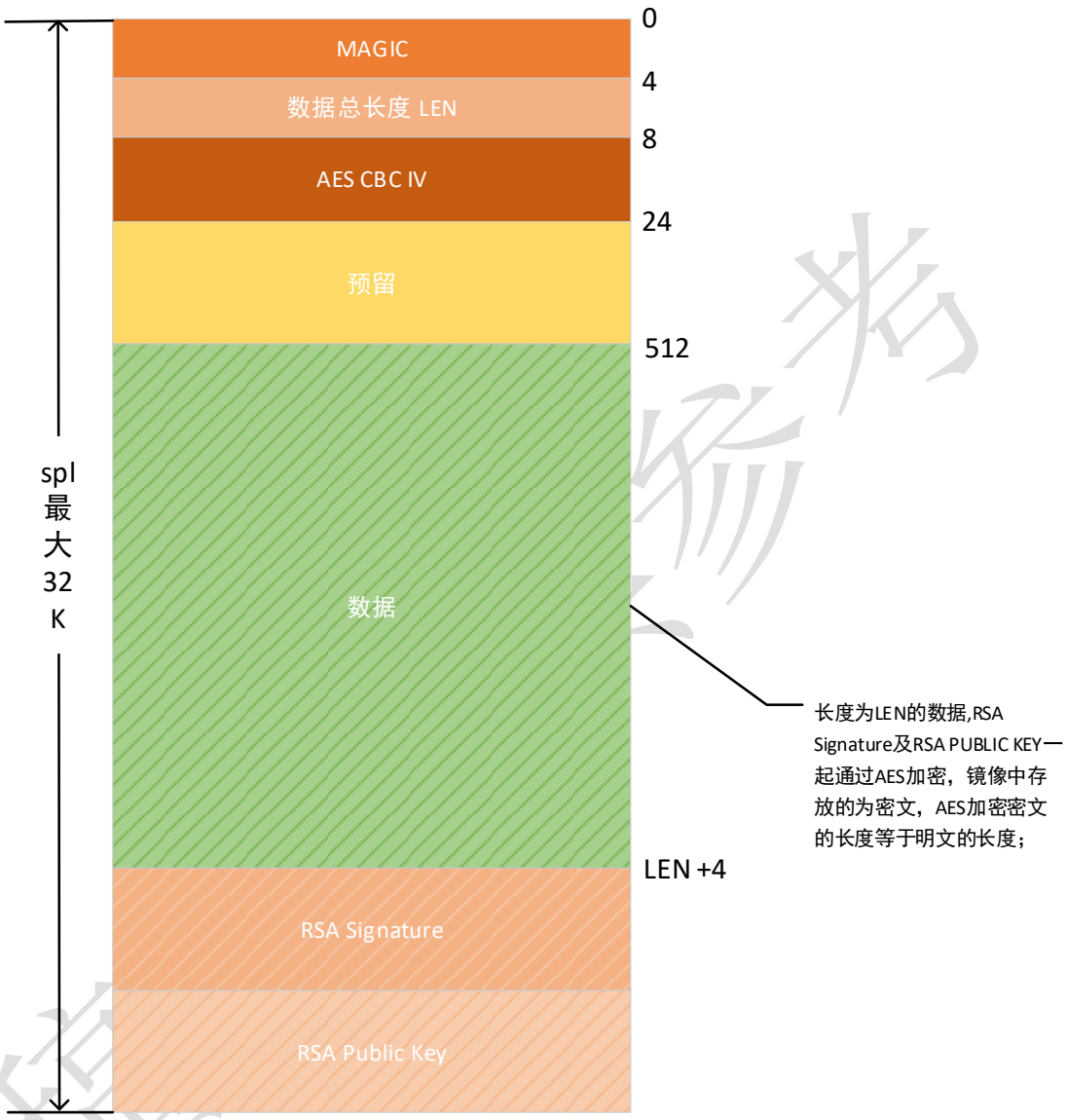

图 1-2 加密安全镜像

加密安全 SPL 镜像有 SPL 数据，RSA 签名和 RSA 公钥一起通过 AES CBC 加密组成；

其中，RSA 只支持 RSA2048，AES CBC IV 的值为非 0 值；

1.3 安全启动流程

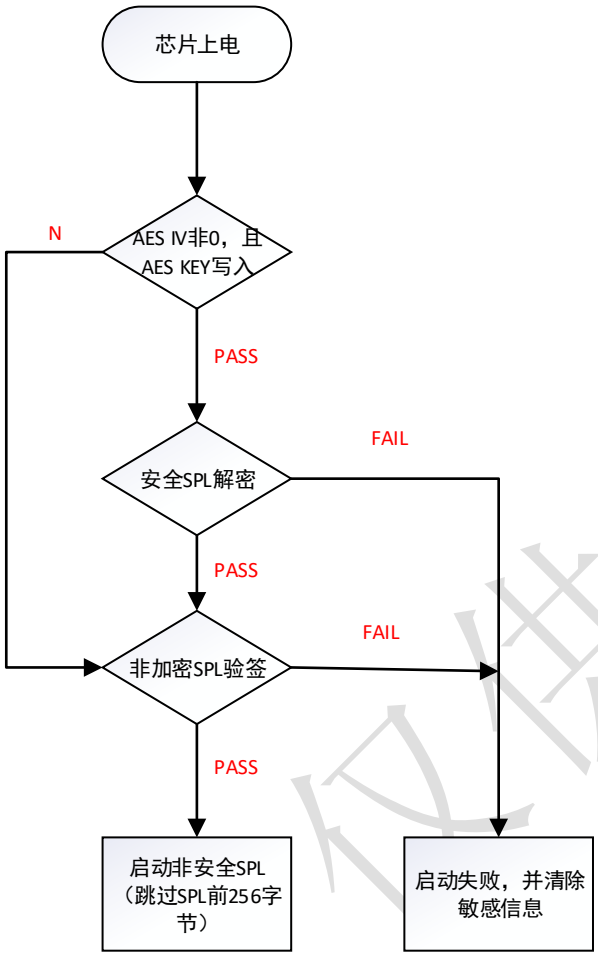

图 1-3 安全启动流程

1.4 安全启动签名目录

安全启动签名目录：

|------ osdrv/opensorce/uboot/uboot/scripts

|------ secure 安全签名脚本目录

|------| aes.cfg AES 加密，KEY 和 IV 配置文件

|------| aes_encode.sh AES CBC-128 加密脚本

|------| rsa_priv.pem 长度为 2048Bit 密钥存放文件

|------| rsa_sign.sh RSA2048 加签脚本

|------| readme.txt 帮助文件

执行 AES-CBC-128 加密之前，先配置 aes.cfg 文件里的 KEY 和 IV 值，例如：

KEY=13246BE7E1008B951110009325241312

IV=262738393a3b4c4d4e4f000000000000

进行 RSA2048 签名前，先生成密钥文件，生成密钥命令为：

openssl genrsa -out rsa_priv.pem 2048

rsa_priv.pem 为要生成的存储 RSA 密钥的文件

1.5 安全启动文件生成

拷贝生成的 uboot-spl-header.img 文件到 secure 目录，执行

./rsa_sign.sh u-boot-spl-header.img rsa_priv.pem 0x100 pss

参数

- ✓ u-boot-spl-header.img 为要签名的文件
- ✓ rsa_priv.pem 为 RSA2048 密钥文件
- ✓ 0x100 为 u-boot-spl-header.img 文件 header 长度，为加签文件 u-boot-spl-header.img 要进行哈希运算的位置偏移地址；
- ✓ pss 参数为使用 pkcs_v21 签名；

RSA2048 加签后签名在 u-boot-spl-header.img 中的位置紧跟在代码之后，签名包括签名数据，RSA 公钥 N 和 E；

如果需要进行 AES-CBC-128 加密，对配置文件 aes.cfg 中 KEY 和 IV 进行配置，执行

./aes_encode.sh u-boot-spl-header.img aes.cfg 0x100

参数

- ✓ u-boot-spl-hader.img 为要进行 AES 加密的文件
- ✓ aes.cfg 为 AES KYE 和 IV 配置文件
- ✓ 0x100 为 u-boot-spl-header.img 进行加密数据位置的偏移；

实例：

1 进入 secure 目录；

2 进行 RSA 加签（rsa_priv.perm 已生成）

./rsa_sign.sh u-boot-spl-header.img rsa_priv.pem 0x100 pss

3 配置 aes.cfg 文件中的 KEY 和 IV

KEY=13246BE7E1008B951110009325241312

IV=262738393a3b4c4d4e4f000000000000

4 进行 AES CBC 对称加密

./aes_encode.sh u-boot-spl-header.img aes.cfg 0x100

5 往 EFUSE 中写公钥的哈希值

./nvmem-test write efuse0 0 16 hex:0514c6c1e96f57621685529aebc7808d

./nvmem-test write efuse0 48 4 hex:dfe75c2b

./nvmem-test write efuse1 48 12 hex:7a0d27c51991404701654a78

6 向 EFUSE ENTRY16 开始写入 128bit 的 AES 的私钥，并 LOCK AES 私钥，AES 私钥写入时，需要四字节大端模式

./nvmem-test write efuse0 16 16 hex:E76B2413958B00E19300101112132425

7 使能安全启动

./nvmem-test write efuse0 60 4 hex:01000033

./nvmem-test write efuse1 60 4 hex:00000008

nvmem-test 为读写 efuse 的应用程序；可以在项目代码工程 mpp/test/platform_driver_test/nvmem 目录下获取；

使用签过名的 u-boot-pdl-header.img u-boot-sdl.bin 和 u-boot-sdl-header.img u-boot.bin 下载到 flash 对应分区，启动；

1.6 安全启动解密

目前 bootloader 阶段没有 AES 和 EFUSE 的驱动，SPL 校验 Uboot 可以通过 boot code 提供的函数指针进行校验；

校验通用流程为：

1. EFUSE ENTRY60 的安全启动使能（sec）位置 1
2. 判断 AES IV 是否不全为 0 及 EFUSE 中写入安全启动 KEY；
3. 使用 AES IV 和 EFUSE 中的 KEY 进行 AES CBC 解密；
4. AES 解密成功或者 1 项条件不满足，根据 bootloader header 中的数据偏移地址及数据长度计算数据的哈希值，并和 EFUSE 中的 256bit 哈希比较；
5. 哈希比较结果相同，进行 RSA2048 验签；
6. RSA2048 验签通过，跳转到 bootloader 代码地址执行；

1.7 EFUSE 驱动函数接口

EFUSE 写操作流程：

1. 配置 EFUSE 使能寄存器 AP_PERI_CLK_CTRL 位 ckg_efuse_en 为 1 和 AP_PERI_APB_CLK_CTRL 位 ckg_apb_efuse_en 为 1 ；
2. 配置 EFUSE_CONFIG(0x04)位 ECE 为 1；
3. 配置 EFUSE_CMD(0x0)为 8，等待状态寄存器 WflagAutoComplete 置位，自动扫描所有 ENTRY，记录每个 ENTRY 是否被 program 过；
4. 配置 EFUSE_CMD(0x0)为 2，把要写操作的 ENTRY 写入位[4：9]，要写的值写入为[12：19]，等待状态寄存器 WComplete 置位；

EFUSE 读操作流程：

1. 配置 EFUSE 使能寄存器 AP_PERI_CLK_CTRL 位 ckg_efuse_en 为 1 和 AP_PERI_APB_CLK_CTRL 位 ckg_apb_efuse_en 为 1 ；
2. 配置 EFUSE_CONFIG(0x04)位 ECE 为 1；
3. 配置 EFUSE_MATCH_Key(0x08)为 0x92fc0025；
4. 配置 EFUSE_CMD(0x0)为 1，等待状态寄存器 LoadUserCMDComplete 置位；
5. 配置 EFUSE_CMD(0x0)为 3，把要读操作的 ENTRY 写入位[4：9]，等待状态寄存器 RComplete 置位；
6. 从 EFUSE_DOUT(0x28)中读出 EFUSE 中值；

EFUSE 驱动实现在文件 mc-efuse.c 中，EFUSE 作为 NVMEM 设备注册到 NVMEM 驱动，EFUSE 驱动成功加载后，会在目录下生成文件结点，：

/sys/bus/nvmem/device/nvmem，用户层可以通过此节点以文件读写方式读写 EFUSE 中的数据；

其他驱动需要读取 EFUSE 中数据时，可通过标准 NVMEM 对外函数接口读写 EFUSE 中数据，NVMEM 对外函数接口在文件 nvmem-consumer.h 中；

以 RTC 校准数据在 EFUSE ENTRY60 和 ENTRY61 为例，使用方法为：

DTS 文件 EFUSE 结点加入 RTC 校准 CELL 信息，

```
efuse: efuse@1BA00000 {
    #address-cells = <1>;
    #size-cells = <1>;
    compatible = "mc,efuse";
    reg = <0x1BA00000 0x1000>;
    mc,efuse-size = <64>;
    clocks = <&apapb_gate CKG_EFUSE>,
        <&apapb_gate CKG_APB_EFUSE>;
    clock-names = "clk_efuse",
        "clk_apb_efuse";
    tsens_calib: calib@3c {
        reg = <0x3c 0x2>;
    };
    status = "ok";
};
```

tsens_calib:calib@3c {

reg = <0x3c 0x2>;

} 表示 EFUSE 中第 60，61 字节为 RTC 校准数据；

DTS 文件 RTC 设备节点加入 nvmem-cell 和 nvmem-cell-names 信息：

```
rtc@1A800000 {
    compatible = "mc,rtc";
    interrupts = <GIC_SPI 7 IRQ_TYPE_LEVEL_HIGH>;
    reg = <0x1A800000 0x400>;
    clocks = <&apapb_gate CKG_APB_RTCT>;
    clock-names = "ckg_apb_rtct_en";
    nvmem-cells = <&tsens_calib>;
    nvmem-cell-names = "calib";
    status = "okay";
};
```

在 RTC 驱动中，通过 nvmem_cell_get 函数获取 rtc 校准信息的 nvmem cell，然后通过函数 nvmem_cell_read 读取对应 cell 中的数据，具体实现为:

```
char *mc_rtc_calib_read(struct device *dev, const char *cname)
{
    struct nvmem_cell *cell;
    ssize_t data;
    char *ret;

    cell = nvmem_cell_get(dev, cname);
    if (IS_ERR(cell))
        return ERR_CAST(cell);

    ret = nvmem_cell_read(cell, &data);
    nvmem_cell_put(cell);

    return ret;
}
```

1.8 EFUSE 中数据是否有写过的判断方法

在 EFUSE 控制器中有 2 个状态寄存器，EFUSE_STATUS1(0X30) 对应到 entry0~entry31，EFUSE_STATUS2(0x34)对应到 entry32~entry63；当其中 bit 位值为 0 时，表示此 entry 没有确认过是否可写或者不可写；当其中 bit 位值为 1 时，表示此 entry 已经被硬件读过，并且确认是可写的；

利用 CMD_WFLAG（0x0）或者 CMD_WFLAG_AUTO（0x8）可以得到此寄存器的值；也就是在读此寄存器前，需要先发送 CMD_WFLAG 或者 CMD_WFLAG_AUTO 命令；

1.9 EFUSE 中 LOCK 位说明

EFUSE 的 entry63 为 LOCK entry；

bit0~bit3 对应 EFUSE APB lock，也就是其中值为 1 时，软件无法读取 EFUSE 中的值，每 1 个 bit 对应 16 个 entry 的 efuse（bit0 对应 lock entry0 ~ entry15， bit1 对应 lock entry16~entry31，bit2 对应 lock entry32~entry47，bit3 对应 lock entry48~entry63）；

bit4~bit7 对应 EFUSE AHB lock，也就是 AES 控制器中 32 个字节 AES KEY 寄存器，每 1 个 bit 对应 8 个字节的 AES 控制器中的 KEY；当其 bit 位为 1 时，软件无法通过 AES 控制器中的 KEY 寄存器读到对应的 EFUSE 值；（bit4 对应 lock AES_KEY0~AES_KEY1，bit5 对应 lock AES_KEY2~AES_KEY3，bit6 对应 lock AES_KEY4~AES_KEY5，bit7 对应 lock AES_KEY6~AES_KEY7）;所以这 4 个位跟具体哪个 EFUSE entry 无关，只跟 AES 控制器中的 AES_KEY 寄存器相关；

1.10 AES 驱动接口

AES 加密流程：

1. 配置 AHB_PERI_CLK_CTRL 位 ckg_ahb_aes_en 为 1；
2. 配置 AES Initial Vector 到 IV 寄存器（0x30-0x3c），IV 寄存器为大端模式；
3. 配置 AES KEY 到 Security Key 寄存器（0x10-0x2C），Key 寄存器为大端模式；
4. 配置 EncryptControl（0x0）寄存器为加密模式，选择 encrypt algorithm 位[1：3]（AES/DES/TDES）算法，并配置 operation mode（ECB/CBC/CTR/CFG/OFB）位[4:6]；
5. 配置 DMA_SRC（0x48）源地址和 DMA_DST（0x4c）目的地址及 DMA_TRANS_SIZE（0x50）数据大小；
6. 配置 DMA_CTRL（0x54）位 0 DMA_EN 为 1，等待 INT_SRC（0x60）位 0 DONE_INTR 置位；

AES 解密流程：

1. 配置 AHB_PERI_CLK_CTRL 位 ckg_ahb_aes_en 为 1；
2. 配置 AES Initial Vector 到 IV 寄存器（0x30-0x3c），IV 寄存器为大端模式；
3. 配置 AES KEY 到 Security Key 寄存器（0x10-0x2C），Key 寄存器为大端模式；
4. 配置 EncryptControl（0x0）寄存器为解密模式，选择 encrypt algorithm 位[1：3]（AES/DES/TDES）算法，并配置 operation mode（ECB/CBC/CTR/CFG/OFB）位[4:6]；
5. 配置 DMA_SRC（0x48）源地址和 DMA_DST（0x4c）目的地址及 DMA_TRANS_SIZE（0x50）数据大小；
6. 配置 DMA_CTRL（0x54）位 0 DMA_EN 为 1，等待 INT_SRC（0x60）位 0 DONE_INTR 置位；

AES 算法驱动实现在 mc-aes.c 文件，支持的算法会注册到 crypto engine，其提供的接口函数在 crpyto.h；AES 驱动加载成功后，可以通过命令 cat /proc/crypto 查看支持的的算法，例如 AES 的 CBC：

Name	cbc(aes)
Driver	mc-cbc-aes
Module	kernel
Priority	300
Refcnt	1
Selftest	passed
Internal	no
Type	ablkcipher

Async	yes
Blocksiz	16
min keysiz	16
max keysize	32
ivsize	16
geniv	<default>

AES 对用户接口是基于 cryptodev 代码，加解密通过 cryptodev 框架的 IOCTL 操作，具体可以参考 AES 测试的 demo 程序；

1.11 AES 用户层加解密使用

1.11.1 AES 加密解密

AES 框架是基于开源代码 cryptodev 框架，建议加解密源地址和目的地址按照 16 字节对齐，示例函数：

int aes_ctx_init(struct cryptodev_ctx* ctx, int cfd, const char *key, unsigned int key_size,enum cryptodev_crypto_op_t alg)

- ctx 为包含设备文件描述符及加解密 session 的结构体；
- cfd 为 cryptodev 设备文件描述符；
- key 加解密使用的 key；
- key_size 加解密 key 的长度；
- alg 加解密使用的算法；

该函数在使用加解密前调用；

aes 加密函数；

int aes_encrypt(struct cryptodev_ctx* ctx, const void* iv, const void* plaintext, void* ciphertext, size_t size);

- iv 为加解密时初始化向量；
- palintext 为加密的源数据；
- ciphertext 为加密后的数据；
- size 为要加解密数据的大小；

aes 解密函数；

int aes_decrypt(struct cryptodev_ctx* ctx, const void* iv, const void* ciphertext, void* plaintext, size_t size);

- iv 为加解密时初始化向量；
- palintext 为加密的源数据；
- ciphertext 为加密后的数据；
- size 为要加解密数据的大小

1.11.2 AES 分级加密解密

1. 使用 key 初始化

   memcpy(test_key, CIPHER_KEY, CIPHER_KEY_LEN);

   memcpy(test_key + CIPHER_KEY_LEN, cipher_key.key, cipher_key.klen);

   aes_ctx_init(&ctx, cfd, test_key, CIPHER_KEY_LEN + cipher_key.klen, CRYPTO_AES_CBC);
   完成 ctx 初始化，初始化时 key 的形式为：

前面 4 个字节为固定字符“CKIS”，之后为解密时真正的密钥；

2. 使用 key 解密要加解密使用 key 的密文

   memcpy(ciphertext, ckey, cipher_key.klen);

   aes_decrypt(&ctx, cipher_key.iv, ciphertext, plaintext, cipher_key.klen);

3. 使用解密出 key 的明文解密要加解密使用 key 的密文

   memset(test_key, 0, sizeof(test_key));

   memcpy(test_key, USE_CIPHER_KEY, USE_CIPHER_KEY_LEN);

   aes_ctx_init(&ctx, cfd, test_key, cipher_key.klen, CRYPTO_AES_CBC);

   调用初始化函数时的 key 使用固定的字符"CIPHER_KEY"

   aes_decrypt(&ctx, cipher_key.iv, ciphertext, plaintext, cipher_len);

   调用解密函数

实列代码：参考 SDK 目录 mpp/test/platform_driver_test/crypto/cipher.c 文件中的 aes_test_key 函数；

1.11.3 AES 使用 EFUSE 中的 key 加密解密

1. 使用 efuse 中 key 初始化；

   memset(test_key, 0, sizeof(test_key));

   memcpy(test_key, KEY_IN_EFUSE, KEY_IN_EFUSE_LEN);

   test_key[KEY_IN_EFUSE_LEN] = 0x20;

   aes_ctx_init(&ctx, cfd, test_key, cipher_key.klen, CRYPTO_AES_CBC);

   初始化时 key 的格式为：

   前 4 个字节固定为："KISE"，第 5 个字节为 key 在 efuse 中的 entry 位置；

   其中往 efuse 写入 key 时，需要先对 key 进行 word 大小端转换；比如原始 key 为："\xc2\x86\x69\x6d\x88\x7c\x9a\xa0\x61\x1b\xbb\x3e\x20\x25\xa4\x5a"

   写入到 efuse 中的 key 则需要为：

   "\x6d\x69\x86\xc2\xa0\x9a\x7c\x88\x3e\xbb\x1b\x61\x5a\xa4\x25\x20"

2. 调用加解密函数；

   aes_decrypt(&ctx, cipher_key.iv, ciphertext, plaintext, cipher_key.klen);

1.11.4 AES 使用 EFUSE 中的 key 分级加密解密

1. 使用 efuse 中 key 初始化；

   memcpy(test_key, KEY_IN_EFUSE, KEY_IN_EFUSE_LEN);

test_key[KEY_IN_EFUSE_LEN] = 0x20;

memcpy(test_key + KEY_IN_EFUSE_LEN + 4, CIPHER_KEY, CIPHER_KEY_LEN);

aes_ctx_init(&ctx, cfd, test_key, cipher_key.klen, CRYPTO_AES_CBC);

初始化时 Key 时固定格式："KISE"20000000 "CKIS"

前四个字节为字符"KISE"，中间 4 个字节为 key 在 efuse 中的起始位置，最后四个字节固定为"CKIS"；

2. 使用 efuse 中 key 解密加解密使用 key 的密文；

aes_decrypt(&ctx, cipher_key.iv, ciphertext, plaintext, cipher_key.klen);

3. 使用解密的 key 加解密；

memset(test_key, 0, sizeof(test_key));

memcpy(test_key, USE_CIPHER_KEY, USE_CIPHER_KEY_LEN);

aes_ctx_init(&ctx, cfd, test_key, cipher_key.klen, CRYPTO_AES_CBC);

调用初始化函数时的 key 使用固定的字符"CIPHER_KEY"；

aes_decrypt(&ctx, cipher_key.iv, ciphertext, plaintext, cipher_len);

最后调用解密函数

实列代码：参考 SDK 目录 mpp/test/platform_driver_test/crypto/cipher.c 文件中的 test_aes_efuse_key 函数；

备注：unsigned char * ckey = "\x96\x15\xa0\x41\x7c\xb7\x9d\xcc"

"\x04\x6d\xd0\x3a\x82\x38\x00\xa4"; 此为分级加解密时 key 的密文，这个由原始 key .key = "\xc2\x86\x69\x6d\x88\x7c\x9a\xa0"

"\x61\x1b\xbb\x3e\x20\x25\xa4\x5a", 四字节大小端转换后，按照 aes cbc 加密得到；

2 附录

2.1 AES

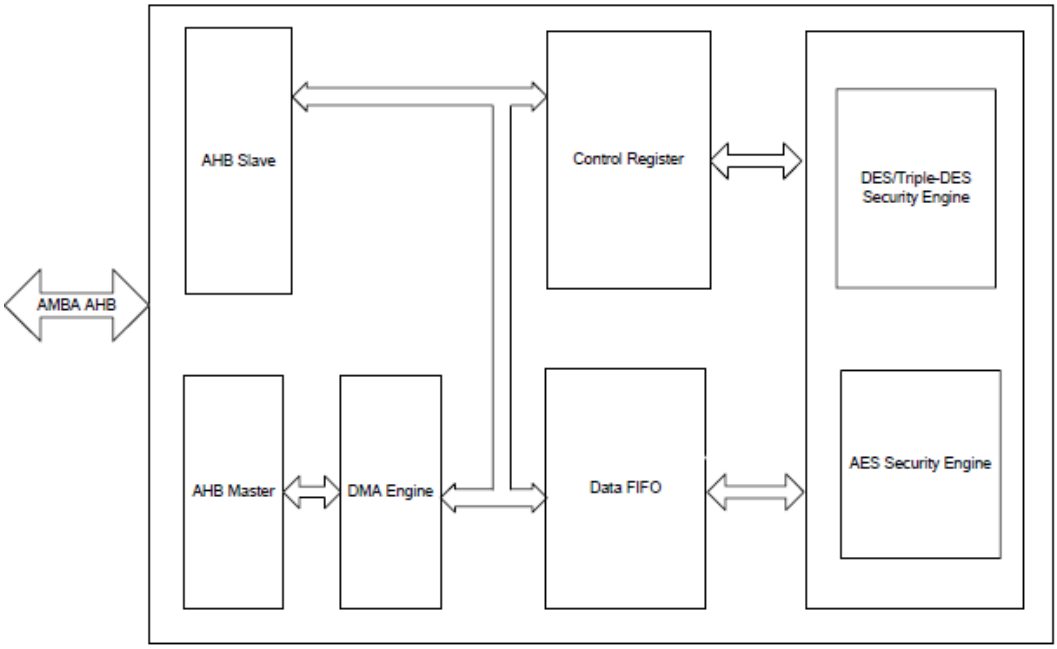

图 2-1 AES 模块图

AES 特性：

DES/TRIPPLE DES 加密解密，符合 NIST 标准；

AES 128/192/256 bit 加密解密，符合 NIST 标准；

支持 DMA 功能；

AES 加密解密使用 KEY 可以从 EFUSE 中读出；

支持具体算法为：

DES/TRIPPLE DES

- ECB
- CBC
- CFB
- OFB

AES

- ECB
- CBC
- CFB
- OFB
- CTR

2.2 EFUSE

EFUSE 以 ENTRY 为单位进行读写操作，每个 ENTRY 8 个 bit，一共 128 个 ENTRY；

EFUSE 中 HASH 字段存储 RSA 公钥 32 字节的哈希值，只有在设置了 APB LOCK（ENTRY 63 的 bit 0 和 ENTRY 127 的 bit 3）位后，软件读不到其 128 位值，通过配置硬件，自动完成哈希的比较；

EFUSE 中存储的 AES KEY，在设置了 APB LOCK（ENTRY63 的 bit 1）位后，软件读不到其 128 位值，设置了 AHB LOCK（ENTYR 63 的 bit 4 和 bit 5）位后，可以传输的 AES 控制器，作为 AES 加解密的私钥；

4 组用户 AES 密钥，也只有设置了 APB LOCK（ENTY63 的 bit 2，ENTRY 127 的 bit 0，bit 1，bit 2）位后，软件读不到其对应的值；设置了 AHB LOCK（ENTYR 63 的 bit 4，bit 5 和 ENTRY 127 的 bit 4，bit 5）位后，可以传输到 AES 控制器，作为 AES 加解密的私钥；

Category	Word Num		BIT MAP																														Entry Num Start
32b		Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	124
		Scope Name	EFUSE LOCK								Customer ID																								
	15	Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
32b		Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	120
		Scope Name	hash31								hash30								hahs29											hash28				
	14	Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
32b		Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	116
		Scope Name	hash27								hash26								hash25											hash24				
	13	Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
32b		Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	112
		Scope Name	hash23								hash22								hash21											hash20				
	12	Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
384b		Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	108
		Scope Name																																	
	11	Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
		Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	104
		Scope Name																																	
	10	Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
		Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	100
		Scope Name																																	
	9	Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
		Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	96
		Scope Name																																	
	8	Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
		Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	92
		Scope Name																																	
	7	Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
		Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	88
		Scope Name																																	
	6	Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
		Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	84
		Scope Name																																	
	5	Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
		Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	80
		Scope Name																																	
	4	Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
		Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	76
		Scope Name																																	
	3	Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
		Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	72
		Scope Name																																	
	2	Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
		Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	68
		Scope Name																																	
	1	Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
		Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	64
		Scope Name																																	
	0	Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Category	Word Num		BIT MAP																														Entry Num Start
		Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
32b	15	Scope Name	EFUSE LOCK								AES-TEST																							sec	60
		Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
32b	14	Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	56
		Scope Name	Manufacture Series Number																																
		Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
32b	13	Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	52
		Scope Name																																	
		Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
32b	12	Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	48
		Scope Name	hash19								hash18								hash17								hash16								
		Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
128b	11	Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	44
		Scope Name																																	
		Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	10	Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	40
		Scope Name																																	
		Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	9	Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	36
		Scope Name																																	
		Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	8	Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	32
		Scope Name																																	
		Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
128b	7	Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	28
		Scope Name	aes-key[127:96]																																
		Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	6	Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	24
		Scope Name	aes-key[95:64]																																
		Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	5	Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	20
		Scope Name	aes-key[63:32]																																
		Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	4	Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	16
		Scope Name	aes-key[31:0]																																
		Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
128b	3	Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	12
		Scope Name	hash15								hash14								hash13								hash12								
		Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	2	Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	8
		Scope Name	hash11								hash10								hash9								hash8								
		Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	1	Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	4
		Scope Name	hash7								hash6								hash5								hash4								
		Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	0	Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0
		Scope Name	hash3								hash2								hash1								hash0								
		Initial Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	