

# Rockchip OTP 开发指南

---

文件标识: RK-KF-YF-147

发布版本: V1.8.0

日期: 2023-09-04

文件密级: ☐绝密 ☐秘密 ☐内部资料 ☒公开

## 免责声明

本文档按“现状”提供, 瑞芯微电子股份有限公司(“本公司”, 下同)不对本文档的任何陈述、信息和内容的准确性、可靠性、完整性、适销性、特定目的性和非侵权性提供任何明示或暗示的声明或保证。本文档仅作为使用指导的参考。

由于产品版本升级或其他原因, 本文档将可能在未经任何通知的情况下, 不定期进行更新或修改。

## 商标声明

“Rockchip”、“瑞芯微”、“瑞芯”均为本公司的注册商标, 归本公司所有。

本文档可能提及的其他所有注册商标或商标, 由其各自拥有者所有。

## 版权所有 © 2023 瑞芯微电子股份有限公司

超越合理使用范畴, 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。

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址: 福建省福州市铜盘路软件园A区18号

网址: [www.rock-chips.com](http://www.rock-chips.com)

客户服务电话: +86-4007-700-590

客户服务传真: +86-591-83951833

客户服务邮箱: [fae@rock-chips.com](mailto:fae@rock-chips.com)

前言

概述

本文档主要介绍 Rockchip OTP OEM 区域烧写。

产品版本

芯片名称	内核版本
RK 系列芯片	Linux 4.19
RK 系列芯片	Linux 5.10

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

修订记录

版本号	作者	修改日期	修改说明
V1.0.0	张学广	2020-10-18	初始版本
V1.0.1	张学广	2021-02-08	格式修订
V1.1.0	林平	2022-01-07	新增Secure OTP OEM区域说明
V1.2.0	林平	2022-01-14	新增判断OEM Cipher Key是否写入说明
V1.3.0	林平	2022-01-14	新增设置OTP Life cycle说明，新增 Protected OEM Zone Write lock 说明
V1.4.0	林平	2022-03-08	修改Non-Protected OEM Zone支持平台，修改UserSpace用户使用 OEM Cipher Key说明
V1.5.0	林平	2023-04-13	修改Non-Secure OTP说明，新增Secure OTP支持平台
V1.6.0	林平	2023-05-29	新增Secure OTP支持平台，新增内核集成读写 Non-Protected OEM Zone 驱动的使用说明
V1.7.0	林平	2023-07-03	新增OTP Map For OEM说明
V1.8.0	林平	2023-09-04	修改Non-Secure OTP说明，新增Secure OTP支持平台



# 目录

## Rockchip OTP 开发指南

1. 概述
2. Non-Secure OTP
  - 2.1 支持平台
  - 2.2 使用方法
3. Secure OTP
  - 3.1 OTP Map For OEM
  - 3.2 Protected OEM Zone
    - 3.2.1 支持平台
    - 3.2.2 使用方法
  - 3.3 Non-Protected OEM Zone
    - 3.3.1 支持平台
    - 3.3.2 使用方法
      - 3.3.2.1 U-Boot 使用方法
      - 3.3.2.2 UserSpace 使用方法
  - 3.4 OEM Cipher Key
    - 3.4.1 支持平台
    - 3.4.2 使用方法
      - 3.4.2.1 U-Boot 使用方法
      - 3.4.2.2 UserSpace 使用方法
  - 3.5 OTP Life Cycle
    - 3.5.1 支持平台
    - 3.5.2 权限变更
    - 3.5.3 使用方法

# 1. 概述

OTP NVM (One Time Programmable Non-Volatile Memory), 即只可编程一次的非易失性存储。作为对比, FLASH 存储可多次擦写。

OTP又将存储区域划分为安全区 (Secure OTP) 和非安全区 (Non-Secure OTP), 非安全世界 (例如U-Boot, UserSpace) 可以直接读取非安全区数据, 但是无权直接读写安全区数据, 一般敏感数据都是存储于安全区域, 只有安全世界 (例如Miniloader/SPL, OP-TEE) 可以直接读写安全区域OTP。

关于安全世界和非安全世界相关概念涉及TrustZone和TEE知识, 细节请参考《Rockchip\_Developer\_Guide\_TEE\_SDK\_CN.md》或 ARM 官方资料。

## 2. Non-Secure OTP

RK 平台 Non-Secure OTP 一般用于存储芯片型号、芯片唯一码 CPUID 等 RK 私有数据, 这些数据在芯片生产时就已经写入。

大部分 RK 平台 Non-Secure OTP 没有预留 OEM 区域, 所以内核驱动仅提供读取接口, 不提供烧写接口。

仅部分 RK 平台 Non-Secure OTP 有预留 OEM 区域(参考如下支持平台列表), 内核驱动提供烧写接口, 方便客户存储自定义数据, 比如: 序列号, MAC 地址, 产品信息等。客户可通过标准文件 API 对 OEM 区域进行读写。

### 2.1 支持平台

Platform	OTP_OEM_OFFSET	RANGE	TOTAL SIZE
RV1126/RV1109	0x100	0x100 ~ 0x1EF	240 Bytes

### 2.2 使用方法

#### OEM Read

```
/*
 * @offset: offset from oem base
 * @buf: buf to store data which read from oem
 * @len: data len in bytes
 */
int rockchip_otp_oem_read(int offset, char *buf, int len)
{
    int fd = 0, ret = 0;

    fd = open("/sys/bus/nvmem/devices/rockchip-otp0/nvmem", O_RDONLY);
    if (fd < 0)
```

```

        return -1;

    ret = lseek(fd, OTP_OEM_OFFSET + offset, SEEK_SET);
    if (ret < 0)
        goto out;

    ret = read(fd, buf, len);
out:
    close(fd);

    return ret;
}

```

## OEM Write

1, 每笔 OEM Write 前都需要使能写开关, 目的是避免误写。

```

int rockchip_otp_enable_write(void)
{
    char magic[] = "1380926283";
    int fd, ret;

    fd = open("/sys/module/nvmem_rockchip_otp/parameters/rockchip_otp_wr_magic",
O_WRONLY);
    if (fd < 0)
        return -1;

    ret = write(fd, magic, 10);
    close(fd);

    return ret;
}

```

2, 写入的数据大小及偏移需要4字节对齐, 数据写入后将被标记写保护, 相应数据写保护将在下次重启后生效。

```

/*
 * @offset: offset from oem base, MUST be 4 bytes aligned
 * @buf: data buf for write
 * @len: data len in bytes, MUST be 4 bytes aligned
 */
int rockchip_otp_oem_write(int offset, char *buf, int len)
{
    int fd = 0, ret = 0;

    /* MUST be 4 bytes aligned */
    if (len % 4)
        return -1;

    fd = open("/sys/bus/nvmem/devices/rockchip-otp0/nvmem", O_WRONLY);
    if (fd < 0)
        return -1;

    ret = lseek(fd, OTP_OEM_OFFSET + offset, SEEK_SET);
    if (ret < 0)
        goto out;
}

```

```

        ret = write(fd, buf, len);
out:
    close(fd);

    return ret;
}

```

## Demo

1, OEM 区域 偏移0的位置写入 0~15

```

void demo(void)
{
    char buf[16] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 };
    int ret = 0;

    ret = rockchip_otp_enable_write();
    if (ret < 0)
        return ret;

    rockchip_otp_oem_write(0, buf, 16);
}

```

2, 通过 OEM Read 或者 hexdump 命令查看结果, 如下为通过命令查看 OEM 区域数据

```

# hexdump -C /sys/bus/nvmem/devices/rockchip-otp0/nvmem
00000000 52 56 11 26 91 fe 21 4b 50 41 30 31 37 00 00 00
00000010 00 00 00 00 10 25 16 12 2f 0e 0f 00 08 00 00 00
00000020 00 00 00 e0 0a e0 0a 1e 00 00 00 00 00 00 00 00
00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
00000100 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
00000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
000001e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001f0 00 00 00 00 00 00 00 00 0f 00 00 00 00 00 00 00

```

## 3. Secure OTP

Secure OTP中预留多种不同的OEM Zone区域用以满足用户不同的使用需求。

### 3.1 OTP Map For OEM

下列表格列举出了各 OEM 数据在 Secure OTP 中的偏移地址和长度, 以字节为单位。

部分签署 NDA 的用户拥有 OTP 驱动源码, 可以参考下列表格自行封装实现读写 OEM 数据的接口。

针对没有 OTP 驱动源码的用户, Rockchip 提供了读写 OEM 数据的接口, 用户可以直接调用, 接口内部自动设置 OEM 数据偏移地址, 所以用户无需设置偏移地址, 只需要关心 OEM 数据的长度即可, 细节请参考后续 "Protected OEM Zone" "Non-Protected OEM Zone" "OEM Cipher Key" 等章节。

Platform	Type	Public Key Hash	Secure Boot Enable Flag	Protected OEM Zone	Non-Protected OEM Zone	OEM Cipher Key
RK3566/RK3568/RV1106	Offset	144	128	672	448	Key0: 512; Key1: 544; Key2: 576; Key3: 608
RK3566/RK3568/RV1106	Length	64	1	224	64	Key0-3: 16 or 24 or 32
RK3588	Offset	2496	32	512	2112	Key0: 160; Key1: 192; Key2: 224; Key3: 416
RK3588	Length	64	1	1536	64	Key0-3: 16 or 24 or 32
RK3528/RK3562	Offset	384	32	576	704	Key0: 192; Key1: 224; Key2: 256; Key3: 288
RK3528/RK3562	Length	64	Bit[0-3]	128	32	Key0-3: 16 or 24 or 32
PX30/RK3326/RK3308/RK3358	Offset	16	0	328	264	Not Support
PX30/RK3326/RK3308/RK3358	Length	32	1	64	64	Not Support
RV1126/RV1109	Offset	16	0	1088	Not Support	Key0: 224; Key1: 256; Key2: 288; Key3: 320; Key_fw: 80
RV1126/RV1109	Length	32	1	2048	Not Support	Key0-3: 16 or 32; Key_fw: 16

## 3.2 Protected OEM Zone

该OEM Zone区域仅供运行在OP-TEE OS上的合法Trust Application(TA应用) 调用，非安全世界无法直接读写该OEM Zone区域，不想暴露给非安全世界的敏感数据建议使用该OEM Zone区域。RK3588平台还支持关闭 Protected OEM Zone 烧写功能，一旦关闭烧写功能，将无法再烧写 Protected OEM Zone。

### 3.2.1 支持平台



Platform	Protected OEM Zone Length	Support Write Lock
RV1126/RV1109	参考 "OTP Map For OEM" 章节	Not Support
RK3308/PX30/RK3326/RK3358	同上	Not Support
RK3566/RK3568	同上	Not Support
RK3588	同上	Support
RK3528/RK3562	同上	Not Support
RV1106	同上	Not Support

### 3.2.2 使用方法

用户需先参考《Rockchip\_Developer\_Guide\_TEE\_SDK\_CN.md》文档，编译运行 rk\_tee\_user/ 目录下的CA TA应用，Demo请参考rk\_tee\_user/v2/ta/rk\_test/rktest\_otp.c，若rktest\_otp.c文件不存在则直接在TA中调用以下函数即可。

获取 Protected OEM Zone Size

```
static TEE_Result get_oem_otp_size(uint32_t *size)
{
    TEE_UUID sta_uuid = { 0x527f12de, 0x3f8e, 0x434f,
        { 0x8f, 0x40, 0x03, 0x07, 0xae, 0x86, 0x4b, 0xaf } };
    TEE_TASessionHandle sta_session = TEE_HANDLE_NULL;
    uint32_t origin;
    TEE_Result res;
    TEE_Param taParams[4];
    uint32_t nParamTypes;

    nParamTypes = TEE_PARAM_TYPES(TEE_PARAM_TYPE_NONE,
        TEE_PARAM_TYPE_NONE,
        TEE_PARAM_TYPE_NONE,
        TEE_PARAM_TYPE_NONE);

    res = TEE_OpenTASession(&sta_uuid, 0, nParamTypes, taParams, &sta_session,
        &origin);
    if (res != TEE_SUCCESS)
    {
        MSG("TEE_OpenTASession failed\n");
        return res;
    }

    nParamTypes = TEE_PARAM_TYPES(TEE_PARAM_TYPE_VALUE_OUTPUT,
        TEE_PARAM_TYPE_NONE,
        TEE_PARAM_TYPE_NONE,
        TEE_PARAM_TYPE_NONE);

    res = TEE_InvokeTACommand(sta_session, 0, 160, nParamTypes,
        taParams, &origin);
    if (res != TEE_SUCCESS)
    {
        MSG("TEE_InvokeTACommand returned 0x%x\n", res);
    }
}
```

```

        *size = taParams[0].value.a;

    TEE_CloseTASession(sta_session);
    sta_session = TEE_HANDLE_NULL;

    return TEE_SUCCESS;
}

```

## 读取 Protected OEM Zone

```

/*
 * read_offset: 偏移区间从0 - (size - 1)
 * read_data: 参数请使用TA中定义的变量
 * read_data_size: 读取长度, 以字节为单位
 */
static TEE_Result read_oem_otp(uint32_t read_offset, uint8_t *read_data, uint32_t
read_data_size)
{
    TEE_UUID sta_uuid = { 0x527f12de, 0x3f8e, 0x434f,
        { 0x8f, 0x40, 0x03, 0x07, 0xae, 0x86, 0x4b, 0xaf } };
    TEE_TASessionHandle sta_session = TEE_HANDLE_NULL;
    uint32_t origin;
    TEE_Result res;
    TEE_Param taParams[4];
    uint32_t nParamTypes;

    nParamTypes = TEE_PARAM_TYPES(TEE_PARAM_TYPE_NONE,
        TEE_PARAM_TYPE_NONE,
        TEE_PARAM_TYPE_NONE,
        TEE_PARAM_TYPE_NONE);

    res = TEE_OpenTASession(&sta_uuid, 0, nParamTypes, taParams, &sta_session,
&origin);
    if (res != TEE_SUCCESS)
    {
        MSG("TEE_OpenTASession failed\n");
        return res;
    }

    nParamTypes = TEE_PARAM_TYPES(TEE_PARAM_TYPE_VALUE_INPUT,
        TEE_PARAM_TYPE_MEMREF_INOUT,
        TEE_PARAM_TYPE_NONE,
        TEE_PARAM_TYPE_NONE);

    taParams[0].value.a = read_offset;
    taParams[1].memref.buffer = read_data;
    taParams[1].memref.size = read_data_size;

    res = TEE_InvokeTACommand(sta_session, 0, 130, nParamTypes,
        taParams, &origin);
    if (res != TEE_SUCCESS)
    {
        MSG("TEE_InvokeTACommand returned 0x%x\n", res);
    }

    TEE_CloseTASession(sta_session);
}

```

```

    sta_session = TEE_HANDLE_NULL;

    return TEE_SUCCESS;

}

```

烧写 Protected OEM Zone

```

/*
 * write_offset: 偏移区间从0 - (size - 1)
 * write_data: 参数请使用TA中定义的变量
 * write_data_size: 烧写长度, 以字节为单位
 */
static TEE_Result write_oem_otp(uint32_t write_offset, uint8_t *write_data,
uint32_t write_data_size)
{
    TEE_UUID sta_uuid = { 0x527f12de, 0x3f8e, 0x434f,
        { 0x8f, 0x40, 0x03, 0x07, 0xae, 0x86, 0x4b, 0xaf } };
    TEE_TASessionHandle sta_session = TEE_HANDLE_NULL;
    uint32_t origin;
    TEE_Result res;
    TEE_Param taParams[4];
    uint32_t nParamTypes;

    nParamTypes = TEE_PARAM_TYPES(TEE_PARAM_TYPE_NONE,
        TEE_PARAM_TYPE_NONE,
        TEE_PARAM_TYPE_NONE,
        TEE_PARAM_TYPE_NONE);

    res = TEE_OpenTASession(&sta_uuid, 0, nParamTypes, taParams, &sta_session,
&origin);
    if (res != TEE_SUCCESS)
    {
        MSG("TEE_OpenTASession failed\n");
        return res;
    }

    nParamTypes = TEE_PARAM_TYPES(TEE_PARAM_TYPE_VALUE_INPUT,
        TEE_PARAM_TYPE_MEMREF_INOUT,
        TEE_PARAM_TYPE_NONE,
        TEE_PARAM_TYPE_NONE);

    taParams[0].value.a = write_offset;
    taParams[1].memref.buffer = write_data;
    taParams[1].memref.size = write_data_size;

    res = TEE_InvokeTACommand(sta_session, 0, 140, nParamTypes,
        taParams, &origin);
    if (res != TEE_SUCCESS)
    {
        MSG("TEE_InvokeTACommand returned 0x%x\n", res);
    }

    TEE_CloseTASession(sta_session);
    sta_session = TEE_HANDLE_NULL;

    return TEE_SUCCESS;
}

```

```
}
```

关闭 Protected OEM Zone 烧写功能

```
enum rk_otp_flag_type {
    LIFE_CYCLE_TO_MISSIONED,
    OEM_OTP_WRITE_LOCK,
};
#define CMD_SET_OTP_FLAGS 170
static TEE_Result set_oem_otp_write_lock(void)
{
    TEE_UUID sta_uuid = { 0x527f12de, 0x3f8e, 0x434f,
        { 0x8f, 0x40, 0x03, 0x07, 0xae, 0x86, 0x4b, 0xaf } };
    TEE_TASessionHandle sta_session = TEE_HANDLE_NULL;
    uint32_t origin;
    TEE_Result res;
    TEE_Param taParams[4];
    uint32_t nParamTypes;

    nParamTypes = TEE_PARAM_TYPES(TEE_PARAM_TYPE_NONE,
        TEE_PARAM_TYPE_NONE,
        TEE_PARAM_TYPE_NONE,
        TEE_PARAM_TYPE_NONE);

    res = TEE_OpenTASession(&sta_uuid, 0, nParamTypes, taParams, &sta_session,
        &origin);
    if (res != TEE_SUCCESS)
    {
        MSG("TEE_OpenTASession failed\n");
        return res;
    }

    nParamTypes = TEE_PARAM_TYPES(TEE_PARAM_TYPE_VALUE_INPUT,
        TEE_PARAM_TYPE_NONE,
        TEE_PARAM_TYPE_NONE,
        TEE_PARAM_TYPE_NONE);

    taParams[0].value.a = OEM_OTP_WRITE_LOCK;
    //disable Protected OEM Zone write from 0 to 511
    taParams[0].value.b = 0;
    res = TEE_InvokeTACommand(sta_session, 0, CMD_SET_OTP_FLAGS, nParamTypes,
        taParams, &origin);
    if (res != TEE_SUCCESS)
    {
        MSG("TEE_InvokeTACommand returned 0x%x\n", res);
    }

    //disable Protected OEM Zone write from 512 to 1023
    taParams[0].value.b = 1;
    res = TEE_InvokeTACommand(sta_session, 0, CMD_SET_OTP_FLAGS, nParamTypes,
        taParams, &origin);
    if (res != TEE_SUCCESS)
    {
        MSG("TEE_InvokeTACommand returned 0x%x\n", res);
    }
}
```

```

//disable Protected OEM Zone write from 1024 to 1535
taParams[0].value.b = 2;
res = TEE_InvokeTACCommand(sta_session, 0, CMD_SET_OTP_FLAGS, nParamTypes,
                           taParams, &origin);
if (res != TEE_SUCCESS)
{
    MSG("TEE_InvokeTACCommand returned 0x%x\n", res);
}

TEE_CloseTASession(sta_session);
sta_session = TEE_HANDLE_NULL;

return TEE_SUCCESS;
}

```

以下是 TA 使用 Protected OEM Zone 参考 Demo:

```

TEE_Result demo_for_oem_otp(void)
{
    TEE_Result res = TEE_SUCCESS;
    uint32_t otp_size = 0;

    res = get_oem_otp_size(&otp_size);
    if (res != TEE_SUCCESS) {
        MSG("get_oem_otp_size failed with code 0x%x", res);
        return res;
    }
    MSG("The OEM Zone size is %d byte.", otp_size);

    uint32_t write_len = 2;
    uint8_t write_data[2] = {0xaa, 0xaa};
    uint32_t write_offset = 0;

    res = write_oem_otp(write_offset, write_data, write_len);
    if (res != TEE_SUCCESS) {
        MSG("write_oem_otp failed with code 0x%x", res);
        return res;
    }
    MSG("write_oem_otp succes with data: 0x%x, 0x%x", write_data[0],
write_data[1]);

    uint32_t read_len = 2;
    uint8_t read_data[2];
    uint32_t read_offset = 0;

    res = read_oem_otp(read_offset, read_data, read_len);
    if (res != TEE_SUCCESS) {
        MSG("read_oem_otp failed with code 0x%x", res);
        return res;
    }
    MSG("read_oem_otp succes with data: 0x%x, 0x%x", read_data[0],
read_data[1]);
    return res;
}

```

### 3.3 Non-Protected OEM Zone

该OEM Zone区域可以被U-Boot和UserSpace调用，数据会暴露在非安全世界内存中。

由于Non-Secure OTP区域较小以及安全因素等原因，目前仅部分平台Non-Secure OTP有预留OEM Zone区域，对于Non-Secure OTP没有预留OEM Zone区域的平台，用户又有在U-Boot和UserSpace读写OTP的需求，可以使用该OEM Zone区域。

#### 3.3.1 支持平台

Platform	Non-Protected OEM Zone Length
RK3308/PX30/RK3326/RK3358/RK3566/RK3568/RK3588/RV1106	参考 "OTP Map For OEM" 章节
RK3528/RK3562	同上

#### 3.3.2 使用方法

##### 3.3.2.1 U-Boot 使用方法

U-Boot 读取 Non-Protected OEM Zone，请调用 u-boot/lib/optee\_clientApi/OpteeClientInterface.c 中 trusty\_read\_oem\_ns\_otp 函数。

U-Boot 烧写 Non-Protected OEM Zone，请调用 u-boot/lib/optee\_clientApi/OpteeClientInterface.c 中 trusty\_write\_oem\_ns\_otp 函数。

以下是U-Boot 使用 Non-Protected OEM Zone 参考 Demo：

```
uint32_t demo_for_oem_ns_otp(void)
{
    TEEC_Result res = TEEC_SUCCESS;

    uint32_t write_len = 2;
    uint8_t write_data[2] = {0xbb, 0xbb};
    uint32_t write_offset = 0;

    res = trusty_write_oem_ns_otp(write_offset, write_data, write_len);
    if (res != TEEC_SUCCESS) {
        printf("trusty_write_oem_ns_otp failed with code 0x%x", res);
        return res;
    }
    printf("trusty_write_oem_ns_otp succes with data: 0x%x, 0x%x", write_data[0],
write_data[1]);

    uint32_t read_len = 2;
    uint8_t read_data[2];
    uint32_t read_offset = 0;

    res = trusty_read_oem_ns_otp(read_offset, read_data, read_len);
    if (res != TEEC_SUCCESS) {
        printf("trusty_read_oem_ns_otp failed with code 0x%x", res);
```

```

        return res;
    }
    printf("trusty_read_oem_ns_otp succes with data: 0x%x, 0x%x", read_data[0],
read_data[1]);
    return res;
}

```

### 3.3.2.2 UserSpace 使用方法

UserSpace 读写 Non-Protected OEM Zone 方法:

确认内核是否集成了读写 Non-Protected OEM Zone 驱动, 存在 kernel/drivers/nvmem/rockchip-secure-otp.c 文件则说明已集成。

内核已集成读写 Non-Protected OEM Zone 驱动, 请参考以下步骤:

1. 确认开启了TEE驱动。

确认对应平台 dtsi 中添加了如下节点:

```

firmware {
    optee: optee {
        compatible = "linaro,optee-tz";
        method = "smc";
    };
};

```

config 中添加以下两个配置:

```

CONFIG_TEE=y
CONFIG_OPTEE=y

```

若出现 /dev/tee0 和 /dev/teepriv0 节点, 说明TEE驱动已开启。

2. 确认开启了读写 Non-Protected OEM Zone 驱动。

确认对应平台 dtsi 中添加了如下节点:

```

secure_otp: secure-otp {
    compatible = "rockchip,secure-otp";
    rockchip,otp-size = <32>;#此处应该修改成对应平台 Non-Protected OEM Zone 实际大小
};

```

config 中添加以下配置:

```

CONFIG_NVMEM_ROCKCHIP_SEC_OTP=y

```

若出现 /sys/bus/nvmem/devices/rockchip-secure-otp0/nvmem 节点, 说明读写 Non-Protected OEM Zone 驱动已开启。

3. 读取 Non-Protected OEM Zone

```

/*
 * @offset: offset from Non-Protected OEM Zone, MUST be 4 bytes aligned

```

```

* @buf: buf to store data which read from Non-Protected OEM Zone
* @len: data len in bytes, MUST be 4 bytes aligned
*/
int rockchip_otp_non_protected_oem_read(int offset, char *buf, int len)
{
    int fd = 0, ret = 0;

    /* MUST be 4 bytes aligned */
    if ((offset % 4) || (len % 4))
        return -1;

    fd = open("/sys/bus/nvmem/devices/rockchip-secure-otp0/nvmem", O_RDONLY);
    if (fd < 0)
        return -1;

    ret = lseek(fd, offset, SEEK_SET);
    if (ret < 0)
        goto out;

    ret = read(fd, buf, len);
out:
    close(fd);

    return ret;
}

```

#### 4. 烧写 Non-Protected OEM Zone

```

/*
* @offset: offset from Non-Protected OEM Zone, MUST be 4 bytes aligned
* @buf: data buf for write
* @len: data len in bytes, MUST be 4 bytes aligned
*/
int rockchip_otp_non_protected_oem_write(int offset, char *buf, int len)
{
    int fd = 0, ret = 0;

    /* MUST be 4 bytes aligned */
    if ((offset % 4) || (len % 4))
        return -1;

    fd = open("/sys/bus/nvmem/devices/rockchip-secure-otp0/nvmem", O_WRONLY);
    if (fd < 0)
        return -1;

    ret = lseek(fd, offset, SEEK_SET);
    if (ret < 0)
        goto out;

    ret = write(fd, buf, len);
out:
    close(fd);

    return ret;
}

```

#### 5. 读写 Demo



```

void demo(void)
{
    char wbuf[4] = { 0, 1, 2, 3};
    char rbuf[4];
    int ret = 0;

    ret = rockchip_otp_non_protected_oem_write(0, wbuf, sizeof(wbuf));
    if (ret < 0) {
        printf("write non protected oem fail!\n");
        return;
    }

    ret = rockchip_otp_non_protected_oem_read(0, rbuf, sizeof(rbuf));
    if (ret < 0) {
        printf("read non protected oem fail!\n");
        return;
    }
}

```

也可以通过 hexdump 命令查看结果

```

# busybox hexdump -C -v /sys/bus/nvmem/devices/rockchip-secure-otp0/nvmem
00000000  00 01 02 03 00 00 00 00  00 00 00 00 00 00 00 00 |.....|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 |.....|
00000020

```

内核未集成读写 Non-Protected OEM Zone 驱动，请参考以下步骤：

UserSpace 用户需先参考《Rockchip\_Developer\_Guide\_TEE\_SDK\_CN.md》文档，编译 rk\_tee\_user/ 目录下的CA应用，然后在CA中参考

rk\_tee\_user/v2/host/rk\_test/rktest.c 中 invoke\_otp\_ns\_read 和 invoke\_otp\_ns\_write 函数的实现。

## 3.4 OEM Cipher Key

该OEM Zone区域用于存储用户密钥，密钥一旦写入不可更改，用户烧写密钥后可以使用指定密钥进行加解密操作，为保证密钥不泄露，系统只提供烧写接口没有读取接口，烧写接口和算法接口可以被U-Boot 和UserSpace调用。

### 3.4.1 支持平台

Platform	OEM Cipher Key Length	Is Support Hardware Read
RV1126/RV1109	参考 "OTP Map For OEM" 章节	Not Support
RK3566/RK3568	同上	Not Support
RK3588	同上	Support
RK3528/RK3562	同上	Support
RV1106	同上	Not Support

## 3.4.2 使用方法

### 3.4.2.1 U-Boot 使用方法

U-Boot 烧写 OEM Cipher Key，请调用 u-boot/lib/optee\_clientApi/OpteeClientInterface.c 中 trusty\_write\_oem\_otp\_key 函数。

函数 uint32\_t trusty\_write\_oem\_otp\_key(enum RK\_OEM\_OTP\_KEYID key\_id, uint8\_t \*byte\_buf, uint32\_t byte\_len) 中 key\_id 结构如下：

```
enum RK_OEM_OTP_KEYID {
    RK_OEM_OTP_KEY0 = 0,
    RK_OEM_OTP_KEY1 = 1,
    RK_OEM_OTP_KEY2 = 2,
    RK_OEM_OTP_KEY3 = 3,
    RK_OEM_OTP_KEY_FW = 10, //keyid of fw_encryption_key
    RK_OEM_OTP_KEYMAX
};
```

上诉平台均支持烧写 RK\_OEM\_OTP\_KEY0、RK\_OEM\_OTP\_KEY1、RK\_OEM\_OTP\_KEY2、RK\_OEM\_OTP\_KEY3；RV1126/RV1109 平台还额外支持烧写 RK\_OEM\_OTP\_KEY\_FW 密钥，RK\_OEM\_OTP\_KEY\_FW 密钥主要用于 BootROM 解密 Loader 固件，用户也可以使用该密钥处理业务数据或者解密 Kernel 固件。

以下是U-Boot烧写 OEM Cipher Key 参考 Demo：

```
uint32_t demo_for_trusty_write_oem_otp_key(void)
{
    uint32_t res;
    uint8_t key[16] = {
        0x53, 0x46, 0x1f, 0x93, 0x4b, 0x16, 0x00, 0x28,
        0xcc, 0x34, 0xb1, 0x37, 0x30, 0xa4, 0x72, 0x66,
    };

    res = trusty_write_oem_otp_key(RK_OEM_OTP_KEY0, key, sizeof(key));
    if (res)
        printf("test trusty_write_oem_otp_key fail! 0x%08x\n", res);
    else
        printf("test trusty_write_oem_otp_key success.\n");
    return res;
}
```

U-Boot 判断是否已经烧写 OEM Cipher Key，请调用 u-boot/lib/optee\_clientApi/OpteeClientInterface.c 中 trusty\_oem\_otp\_key\_is\_written 函数。

以下是U-Boot判断是否已经烧写 OEM Cipher Key 参考 Demo：

```

void demo_for_trusty_oem_otp_key_is_written(void)
{
    uint8_t value;
    uint32_t res = trusty_oem_otp_key_is_written(RK_OEM_OTP_KEY0, &value);
    if (res == TEEC_SUCCESS) {
        printf("oem otp key is %s", value ? "written" : "empty");
    } else {
        printf("access oem otp key fail!");
    }
}

```

部分平台还支持 Hardware Read 功能，用户可以调用 u-boot/lib/optee\_clientApi/OpteeClientInterface.c 中 trusty\_set\_oem\_hr\_otp\_read\_lock 函数，

调用该函数后CPU将无权限访问该密钥，密钥数据不出现在安全和非安全世界内存中，达到密钥与CPU隔离的目的，硬件可以自动读取该密钥送到crypto模块进行加解密运算。若RK3588使用的是RK\_OEM\_OTP\_KEY0、RK\_OEM\_OTP\_KEY1、RK\_OEM\_OTP\_KEY2，在调用该函数后会更改CPU对OTP其他数据的读写权限，比如Secure Boot、Security Level等数据将失去烧写权限，所以用户需要确认后续不会烧写OTP数据后再调用该函数。若RK3588使用的是RK\_OEM\_OTP\_KEY3时，调用该函数不会影响OTP其他数据读写权限。

以下是 RK3588 平台 U-Boot 使用 Hardware Read 功能参考 Demo：

```

uint32_t demo_for_trusty_set_oem_hr_otp_read_lock(void)
{
    uint32_t res;

    res = trusty_set_oem_hr_otp_read_lock(RK_OEM_OTP_KEY0);
    if (res)
        printf("test trusty_set_oem_hr_otp_read_lock fail! 0x%08x\n", res);
    else
        printf("test trusty_set_oem_hr_otp_read_lock success.\n");
    return res;
}

```

U-Boot 使用OEM Cipher Key进行加解密操作，请调用 u-boot/lib/optee\_clientApi/OpteeClientInterface.c 中 trusty\_oem\_otp\_key\_cipher 函数。

以下是U-Boot使用 OEM Cipher Key 参考 Demo：

```

uint32_t demo_for_trusty_oem_otp_key_cipher(void)
{
    uint32_t res;
    rk_cipher_config config;
    uintptr_t src_phys_addr, dest_phys_addr;
    uint32_t key_id = RK_OEM_OTP_KEY0;
    uint32_t key_len = 16;
    uint32_t algo = RK_ALGO_AES;
    uint32_t mode = RK_CIPHER_MODE_CBC;
    uint32_t operation = RK_MODE_ENCRYPT;
    uint8_t iv[16] = {
        0x10, 0x44, 0x80, 0xb3, 0x88, 0x5f, 0x02, 0x03,
        0x05, 0x21, 0x07, 0xc9, 0x44, 0x00, 0x1b, 0x80,
    };
    uint8_t inout[16] = {
        0xc9, 0x07, 0x21, 0x05, 0x80, 0x1b, 0x00, 0x44,
    };
}

```

```

        0xac, 0x13, 0xfb, 0x23, 0x93, 0x4a, 0x66, 0xe4,
    };
    uint32_t data_len = sizeof(inout);

    config.algo = algo;
    config.mode = mode;
    config.operation = operation;
    config.key_len = key_len;
    config.reserved = NULL;
    memcpy(config.iv, iv, sizeof(iv));

    src_phys_addr = (uintptr_t)inout;
    dest_phys_addr = src_phys_addr;

    res = trusty_oem_otp_key_cipher(key_id, &config,
                                    src_phys_addr,
                                    dest_phys_addr,
                                    data_len);

    if (res)
        printf("test trusty_oem_otp_key_phys_cipher fail! 0x%08x\n", res);
    else
        printf("test trusty_oem_otp_key_phys_cipher success.\n");

    return res;
}

```

#### 3.4.2.2 UserSpace 使用方法

UserSpace 端烧写和使用 OEM Cipher Key 与 U-Boot 端类似，*使用注意事项参考上述 U-Boot 烧写和使用 OEM Cipher Key 内容。*

UserSpace 用户烧写和使用 OEM Cipher Key 请参考 librcrypto/demo/demo\_otpkey.c， librcrypto源码和文档《Rockchip\_Developer\_Guide\_Crypto\_HWRNG\_CN.pdf》默认已集成到SDK中。

Android平台：librcrypto源码在hardware/rockchip/目录下。

Linux平台：librcrypto源码在external/目录下。

## 3.5 OTP Life Cycle

部分平台支持OTP Life Cycle，其作用是控制OTP中数据在不同生命周期具有不同的访问权限。

### 3.5.1 支持平台

Platform	OTP Life Cycle Type	说明
RK3588	Blank/Tested/Provisioned/Missioned	Blank阶段拥有最高的读写权限，Missioned阶段读写权限最低，读写权限依次递减，高权限阶段可以选择进入低权限阶段，低权限阶段不能进入高权限阶段。芯片出厂时是Provisioned阶段，OEM可以选择进入Missioned阶段，OEM从Provisioned阶段进入Missioned阶段后，部分OTP数据读写权限将发生变更。

### 3.5.2 权限变更

以下为RK3588 OTP在Provisioned阶段和Missioned阶段的读写权限列表，其中 RW 表示可读写，R 表示只读。

数据	Provisioned	Missioned	说明
Secure Boot Enable Flag	RW	R	若用户需要使用Secure Boot功能，需要开启Secure Boot功能后才能更改OTP Life Cycle，Secure Boot详见《Rockchip_Developer_Guide_Secure_Boot_Application_Note_EN.md》
RSA Public Hash	RW	R	同上
Security Level	RW	R	若用户需要使用强弱安全可选功能，需要选择Security Level后才能更改OTP Life Cycle，Security Level详见《Rockchip_Developer_Guide_TEE_SDK_CN》文档
OEM Cipher Key0-2	RW	无读写权限	详见 OEM Cipher Key 章节
FW encryption key	RW	无读写权限	主要用于加密Loader固件，BootRom启动阶段会使用该密钥解密

### 3.5.3 使用方法

目前更改OTP Life Cycle只能在安全世界修改，若要将OTP Life Cycle从Provisioned阶段改为Missioned阶段，用户需先参考《Rockchip\_Developer\_Guide\_TEE\_SDK\_CN.md》文档，编译运行 rk\_tee\_user/ 目录下的CA TA应用，然后在TA中调用以下函数即可。

```
enum rk_otp_flag_type {
    LIFE_CYCLE_TO_MISSIONED,
    OEM_OTP_WRITE_LOCK,
};
#define CMD_SET_OTP_FLAGS      170
static TEE_Result set_otp_life_cycle_to_missioned(void)
{
    TEE_UUID sta_uuid = { 0x527f12de, 0x3f8e, 0x434f,
        { 0x8f, 0x40, 0x03, 0x07, 0xae, 0x86, 0x4b, 0xaf } };
    TEE_TASessionHandle sta_session = TEE_HANDLE_NULL;
    uint32_t origin;
```

```

TEE_Result res;
TEE_Param taParams[4];
uint32_t nParamTypes;

nParamTypes = TEE_PARAM_TYPES(TEE_PARAM_TYPE_NONE,
                               TEE_PARAM_TYPE_NONE,
                               TEE_PARAM_TYPE_NONE,
                               TEE_PARAM_TYPE_NONE);

res = TEE_OpenTASession(&sta_uuid, 0, nParamTypes, taParams, &sta_session,
&origin);
if (res != TEE_SUCCESS)
{
    EMSG("TEE_OpenTASession failed\n");
    return res;
}

nParamTypes = TEE_PARAM_TYPES(TEE_PARAM_TYPE_VALUE_INPUT,
                               TEE_PARAM_TYPE_NONE,
                               TEE_PARAM_TYPE_NONE,
                               TEE_PARAM_TYPE_NONE);

taParams[0].value.a = LIFE_CYCLE_TO_MISSIONED;
res = TEE_InvokeTACommand(sta_session, 0, CMD_SET_OTP_FLAGS, nParamTypes,
                           taParams, &origin);
if (res != TEE_SUCCESS)
{
    EMSG("TEE_InvokeTACommand returned 0x%x\n", res);
}

TEE_CloseTASession(sta_session);
sta_session = TEE_HANDLE_NULL;

return TEE_SUCCESS;
}

```