Rockchip OTP 开发指南

文件标识：RK-KF-YF-147

发布版本：V1.3.0

日期：2022-01-14

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。

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

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

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

目录

---

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

OTP Layout

RK 平台 Non-Secure OTP Layout 结构基本相同，大小和偏移因芯片而异。

RV1126/RV1109

RV1126/RV1109 Non-Secure OTP 布局如表 1-1 所示：

Type	Range [bytes]	Description
SYSTEM	0x000 ~ 0x0FF	system info, read only
OEM	0x100 ~ 0x1EF	oem zone for customized
RESERVED	0x1F0 ~ 0x1F7	reserved
WP	0x1F8 ~ 0x1FF	write protection for oem zone

表 1-1 RV1126/RV1109 Non-Secure OTP Layout

OEM Zone

RK 平台 OTP 预留 OEM 区域，方便客户存储自定义数据，比如：序列号，MAC 地址，产品信息等。通过标准文件读写 API 对 OEM 区域进行读写。参考 OTP Layout 查询各芯片平台 OEM 支持情况。比如：RV1126的 OTP_OEM_OFFSET 为 0x100，RANGE 为 0x100 ~ 0x1EF，TOTAL SIZE 为 240 bytes。

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。

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。

支持平台

Platform	Protected OEM Zone Size	Support Write Lock
RV1126/RV1109	2048 Bytes	Not Support
RK3308/RK3326/RK3358	64 Bytes	Not Support
RK3566/RK3568	224 Bytes	Not Support
RK3588	1536 Bytes	Support

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)
    {
        EMSG("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)
    {
        EMSG("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)
    {
        EMSG("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)
    {
        EMSG("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)
    {
        EMSG("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)
    {
        EMSG("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)
    {
        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 = 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)
    {
        EMSG("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)
    {
        EMSG("TEE_InvokeTACommand returned 0x%x\n", res);
    }

    //disable Protected OEM Zone write from 1024 to 1535
    taParams[0].value.b = 2;
    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;

}
```

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) {
        EMSG("get_oem_otp_size failed with code 0x%x", res);
        return res;
    }
    IMSG("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) {
        EMSG("write_oem_otp failed with code 0x%x", res);
        return res;
    }
    IMSG("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) {
        EMSG("read_oem_otp failed with code 0x%x", res);
        return res;
    }
    IMSG("read_oem_otp succes with data: 0x%x, 0x%x", read_data[0],
read_data[1]);
    return res;
}
```

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。

支持平台

Platform	Non-Protected OEM Zone Size
RK3308/RK3326/RK3358	64 Bytes

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;
}
```

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，或者直接调用以下函数即可

```
#define STORAGE_CMD_READ_OEM_NS_OTP     13
/* byte_off 区间从 0 - (size - 1) */
static uint32_t read_oem_ns_otp(uint32_t byte_off, uint8_t *byte_buf, uint32_t
byte_len)
{
    TEEC_Result res = TEEC_SUCCESS;
    uint32_t error_origin = 0;
    TEEC_Context contex;
    TEEC_Session session;
    TEEC_Operation operation;
    const TEEC_UUID storage_uuid = { 0x2d26d8a8, 0x5134, 0x4dd8,
            { 0xb3, 0x2f, 0xb3, 0x4b, 0xce, 0xeb, 0xc4, 0x71 } };
    const TEEC_UUID *uuid = &storage_uuid;
```

```
    //[1] Connect to TEE
    res = TEEC_InitializeContext(NULL, &contex);
    if (res != TEEC_SUCCESS) {
        printf("TEEC_InitializeContext failed with code 0x%x\n", res);
        return res;
    }

    //[2] Open session with TEE application
    res = TEEC_OpenSession(&contex, &session, uuid,
                  TEEC_LOGIN_PUBLIC, NULL, NULL, &error_origin);
    if (res != TEEC_SUCCESS) {
        printf("TEEC_Opensession failed with code 0x%x origin 0x%x\n",
               res, error_origin);
        goto out;
    }

    //[3] Start invoke command to the TEE application.
    memset(&operation, 0, sizeof(TEEC_Operation));
    operation.paramTypes = TEEC_PARAM_TYPES(TEEC_VALUE_INPUT,
                        TEEC_MEMREF_TEMP_OUTPUT,
                        TEEC_NONE, TEEC_NONE);
    operation.params[0].value.a = byte_off;
    operation.params[1].tmpref.size = byte_len;
    operation.params[1].tmpref.buffer = (void *)byte_buf;

    res = TEEC_InvokeCommand(&session, STORAGE_CMD_READ_OEM_NS_OTP,
                &operation, &error_origin);
    if (res != TEEC_SUCCESS) {
        printf("InvokeCommand ERR! res= 0x%x\n", res);
        goto out1;
    }

    printf("Read OK.\n");
out1:
    TEEC_CloseSession(&session);
out:
    TEEC_FinalizeContext(&contex);
    return res;
}
```

```
#define STORAGE_CMD_WRITE_OEM_NS_OTP        12
/* byte_off 区间从 0 - (size - 1) */
static uint32_t write_oem_ns_otp(uint32_t byte_off, uint8_t *byte_buf, uint32_t
byte_len)
{
    TEEC_Result res = TEEC_SUCCESS;
    uint32_t error_origin = 0;
    TEEC_Context contex;
    TEEC_Session session;
    TEEC_Operation operation;
    const TEEC_UUID storage_uuid = { 0x2d26d8a8, 0x5134, 0x4dd8,
            { 0xb3, 0x2f, 0xb3, 0x4b, 0xce, 0xeb, 0xc4, 0x71 } };
    const TEEC_UUID *uuid = &storage_uuid;

    //[1] Connect to TEE
    res = TEEC_InitializeContext(NULL, &contex);
```

```
    if (res != TEEC_SUCCESS) {
        printf("TEEC_InitializeContext failed with code 0x%x\n", res);
        return res;
    }

    //[2] Open session with TEE application
    res = TEEC_OpenSession(&contex, &session, uuid,
                   TEEC_LOGIN_PUBLIC, NULL, NULL, &error_origin);
    if (res != TEEC_SUCCESS) {
        printf("TEEC_Opensession failed with code 0x%x origin 0x%x\n",
               res, error_origin);
        goto out;
    }

    //[3] Start invoke command to the TEE application.
    memset(&operation, 0, sizeof(TEEC_Operation));
    operation.paramTypes = TEEC_PARAM_TYPES(TEEC_VALUE_INPUT,
                         TEEC_MEMREF_TEMP_INPUT,
                         TEEC_NONE, TEEC_NONE);
    operation.params[0].value.a = byte_off;
    operation.params[1].tmpref.size = byte_len;
    operation.params[1].tmpref.buffer = (void *)byte_buf;

    res = TEEC_InvokeCommand(&session, STORAGE_CMD_WRITE_OEM_NS_OTP,
                 &operation, &error_origin);
    if (res != TEEC_SUCCESS) {
        printf("InvokeCommand ERR! res= 0x%x\n", res);
        goto out1;
    }

    printf("Write OK.\n");
out1:
    TEEC_CloseSession(&session);
out:
    TEEC_FinalizeContext(&contex);
    return res;
}
```

以下是UserSpace 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 = write_oem_ns_otp(write_offset, write_data, write_len);
    if (res != TEEC_SUCCESS) {
        printf("write_oem_ns_otp failed with code 0x%x", res);
        return res;
    }
    printf("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 = read_oem_ns_otp(read_offset, read_data, read_len);
    if (res != TEEC_SUCCESS) {
        printf("read_oem_ns_otp failed with code 0x%x", res);
        return res;
    }
    printf("read_oem_ns_otp succes with data: 0x%x, 0x%x", read_data[0],
read_data[1]);
    return res;
}
```

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。

支持平台

Platform	OEM Cipher Key Length	Is Support Hardware Read
RV1126/RV1109	RK_OEM_OTP_KEY0-3 (16 or 32 Bytes), RK_OEM_OTP_KEY_FW(16 Bytes)	Not Support
RK3566/RK3568	RK_OEM_OTP_KEY0-3 (16 or 24 or 32 Bytes)	Not Support
RK3588	RK_OEM_OTP_KEY0-3 (16 or 24 or 32 Bytes)	Support

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!");
    }
}
```

另外 RK3588 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;
}
```

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中调用以下函数即可

```
typedef struct {
    uint32_t    algo;
    uint32_t    mode;
    uint32_t    operation;
```

```
    uint8_t     key[64];
    uint32_t    key_len;
    uint8_t     iv[16];
    void        *reserved;
} rk_cipher_config;

/* Crypto algorithm */
enum RK_CRYPTO_ALGO {
    RK_ALGO_AES = 1,
    RK_ALGO_DES,
    RK_ALGO_TDES,
    RK_ALGO_SM4,
    RK_ALGO_ALGO_MAX
};

/* Crypto mode */
enum RK_CIPIHER_MODE {
    RK_CIPHER_MODE_ECB = 0,
    RK_CIPHER_MODE_CBC = 1,
    RK_CIPHER_MODE_CTS = 2,
    RK_CIPHER_MODE_CTR = 3,
    RK_CIPHER_MODE_CFB = 4,
    RK_CIPHER_MODE_OFB = 5,
    RK_CIPHER_MODE_XTS = 6,
    RK_CIPHER_MODE_CCM = 7,
    RK_CIPHER_MODE_GCM = 8,
    RK_CIPHER_MODE_CMAC = 9,
    RK_CIPHER_MODE_CBC_MAC = 10,
    RK_CIPHER_MODE_MAX
};

/* Algorithm operation */
#define RK_MODE_ENCRYPT         1
#define RK_MODE_DECRYPT         0

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
};

#define AES_BLOCK_SIZE          16
#define RK_CRYPTO_MAX_DATA_LEN      (1 * 1024 * 1024)

#define STORAGE_UUID { 0x2d26d8a8, 0x5134, 0x4dd8, \
        { 0xb3, 0x2f, 0xb3, 0x4b, 0xce, 0xeb, 0xc4, 0x71 } }
#define RK_CRYPTO_SERVICE_UUID  { 0x0cacdb5d, 0x4fea, 0x466c, \
        { 0x97, 0x16, 0x3d, 0x54, 0x16, 0x52, 0x83, 0x0f } }

#define STORAGE_CMD_WRITE_OEM_OTP_KEY           14
#define STORAGE_CMD_SET_OEM_HR_OTP_READ_LOCK        15
#define STORAGE_CMD_OEM_OTP_KEY_IS_WRITTEN      16
#define CRYPTO_SERVICE_CMD_OEM_OTP_KEY_CIPHER       0x00000001

TEEC_Result rk_write_oem_otp_key(enum RK_OEM_OTP_KEYID key_id, uint8_t *key,
```

```
                uint32_t key_len)
{
    TEEC_Result res;
    TEEC_Context contex;
    TEEC_Session session;
    TEEC_Operation operation;
    TEEC_UUID uuid = STORAGE_UUID;
    uint32_t error_origin = 0;

    if (key_id != RK_OEM_OTP_KEY0 &&
        key_id != RK_OEM_OTP_KEY1 &&
        key_id != RK_OEM_OTP_KEY2 &&
        key_id != RK_OEM_OTP_KEY3 &&
        key_id != RK_OEM_OTP_KEY_FW) {
        printf("key_id param error!");
        return TEEC_ERROR_BAD_PARAMETERS;
    }
    if (!key) {
        printf("error! key is null!");
        return TEEC_ERROR_BAD_PARAMETERS;
    }
    if (key_len != 16 &&
        key_len != 24 &&
        key_len != 32) {
        printf("key_len param error!");
        return TEEC_ERROR_BAD_PARAMETERS;
    }
    if (key_id == RK_OEM_OTP_KEY_FW &&
        key_len != 16) {
        printf("key_len param error!");
        return TEEC_ERROR_BAD_PARAMETERS;
    }

    res = TEEC_InitializeContext(NULL, &contex);
    if (res != TEEC_SUCCESS) {
        printf("TEEC_InitializeContext failed with code TEEC res= 0x%x", res);
        return res;
    }

    res = TEEC_OpenSession(&contex, &session, &uuid, TEEC_LOGIN_PUBLIC,
                   NULL, NULL, &error_origin);
    if (res != TEEC_SUCCESS) {
        printf("TEEC_Opensession failed with code TEEC res= 0x%x origin 0x%x",
            res, error_origin);
        goto out;
    }

    memset(&operation, 0, sizeof(TEEC_Operation));
    operation.params[0].value.a       = key_id;
    operation.params[1].tmpref.buffer = key;
    operation.params[1].tmpref.size   = key_len;
    operation.paramTypes = TEEC_PARAM_TYPES(TEEC_VALUE_INPUT,
                        TEEC_MEMREF_TEMP_INPUT,
                        TEEC_NONE,
                        TEEC_NONE);

    res = TEEC_InvokeCommand(&session, STORAGE_CMD_WRITE_OEM_OTP_KEY,
                 &operation, &error_origin);
```

```
    if (res != TEEC_SUCCESS) {
        printf("InvokeCommand ERR! TEEC res= 0x%x, error_origin= 0x%x",
            res, error_origin);
    }

    TEEC_CloseSession(&session);
out:
    TEEC_FinalizeContext(&contex);
    return res;
}

TEEC_Result rk_oem_otp_key_is_written(enum RK_OEM_OTP_KEYID key_id, uint8_t
*written_or_not)
{
    TEEC_Result res;
    TEEC_Context contex;
    TEEC_Session session;
    TEEC_Operation operation;
    TEEC_UUID uuid = STORAGE_UUID;
    uint32_t error_origin = 0;

    if (key_id != RK_OEM_OTP_KEY0 &&
        key_id != RK_OEM_OTP_KEY1 &&
        key_id != RK_OEM_OTP_KEY2 &&
        key_id != RK_OEM_OTP_KEY3 &&
        key_id != RK_OEM_OTP_KEY_FW) {
        printf("key_id param error!");
        return TEEC_ERROR_BAD_PARAMETERS;
    }
    if (!written_or_not) {
        printf("error! written_or_not is null!");
        return TEEC_ERROR_BAD_PARAMETERS;
    }

    res = TEEC_InitializeContext(NULL, &contex);
    if (res != TEEC_SUCCESS) {
        printf("TEEC_InitializeContext failed with code TEEC res= 0x%x", res);
        return res;
    }

    res = TEEC_OpenSession(&contex, &session, &uuid, TEEC_LOGIN_PUBLIC,
                    NULL, NULL, &error_origin);
    if (res != TEEC_SUCCESS) {
        printf("TEEC_Opensession failed with code TEEC res= 0x%x origin 0x%x",
            res, error_origin);
        goto out;
    }

    memset(&operation, 0, sizeof(TEEC_Operation));
    operation.params[0].value.a = key_id;
    operation.paramTypes = TEEC_PARAM_TYPES(TEEC_VALUE_INOUT,
                        TEEC_NONE,
                        TEEC_NONE,
                        TEEC_NONE);

    res = TEEC_InvokeCommand(&session, STORAGE_CMD_OEM_OTP_KEY_IS_WRITTEN,
                &operation, &error_origin);
    if (res != TEEC_SUCCESS) {
```

```
        printf("InvokeCommand ERR! TEEC res= 0x%x, error_origin= 0x%x",
            res, error_origin);
    } else {
        *written_or_not = operation.params[0].value.b;
    }

    TEEC_CloseSession(&session);

out:
    TEEC_FinalizeContext(&contex);
    return res;
}

TEEC_Result rk_set_oem_hr_otp_read_lock(enum RK_OEM_OTP_KEYID key_id)
{
    TEEC_Result res;
    TEEC_Context contex;
    TEEC_Session session;
    TEEC_Operation operation;
    TEEC_UUID uuid = STORAGE_UUID;
    uint32_t error_origin = 0;

    if (key_id != RK_OEM_OTP_KEY0 &&
        key_id != RK_OEM_OTP_KEY1 &&
        key_id != RK_OEM_OTP_KEY2 &&
        key_id != RK_OEM_OTP_KEY3) {
        printf("key_id param error!");
        return TEEC_ERROR_BAD_PARAMETERS;
    }

    res = TEEC_InitializeContext(NULL, &contex);
    if (res != TEEC_SUCCESS) {
        printf("TEEC_InitializeContext failed with code TEEC res= 0x%x", res);
        return res;
    }

    res = TEEC_OpenSession(&contex, &session, &uuid, TEEC_LOGIN_PUBLIC,
                    NULL, NULL, &error_origin);
    if (res != TEEC_SUCCESS) {
        printf("TEEC_Opensession failed with code TEEC res= 0x%x origin 0x%x",
            res, error_origin);
        goto out;
    }

    memset(&operation, 0, sizeof(TEEC_Operation));
    operation.params[0].value.a = key_id;
    operation.paramTypes = TEEC_PARAM_TYPES(TEEC_VALUE_INPUT,
                        TEEC_NONE,
                        TEEC_NONE,
                        TEEC_NONE);

    res = TEEC_InvokeCommand(&session, STORAGE_CMD_SET_OEM_HR_OTP_READ_LOCK,
                &operation, &error_origin);
    if (res != TEEC_SUCCESS) {
        printf("InvokeCommand ERR! TEEC res= 0x%x, error_origin= 0x%x",
            res, error_origin);
    }
```

```
    TEEC_CloseSession(&session);
out:
    TEEC_FinalizeContext(&contex);
    return res;
}

TEEC_Result rk_oem_otp_key_cipher(enum RK_OEM_OTP_KEYID key_id, rk_cipher_config
*config,
                  uint8_t *src, uint8_t *dst, uint32_t len)
{
    TEEC_Result res;
    TEEC_Context contex;
    TEEC_Session session;
    TEEC_Operation operation;
    TEEC_UUID uuid = RK_CRYPTO_SERVICE_UUID;
    uint32_t error_origin = 0;
    TEEC_SharedMemory sm;

    if (key_id != RK_OEM_OTP_KEY0 &&
        key_id != RK_OEM_OTP_KEY1 &&
        key_id != RK_OEM_OTP_KEY2 &&
        key_id != RK_OEM_OTP_KEY3 &&
        key_id != RK_OEM_OTP_KEY_FW) {
        printf("key_id param error!");
        return TEEC_ERROR_BAD_PARAMETERS;
    }
    if (!config  || !src || !dst) {
        printf("config or src or dst is null!");
        return TEEC_ERROR_BAD_PARAMETERS;
    }
    if (config->algo != RK_ALGO_AES &&
        config->algo != RK_ALGO_SM4) {
        printf("config->algo error!");
        return TEEC_ERROR_BAD_PARAMETERS;
    }
    if (config->mode >= RK_CIPHER_MODE_XTS) {
        printf("config->mode error!");
        return TEEC_ERROR_BAD_PARAMETERS;
    }
    if (config->operation != RK_MODE_ENCRYPT &&
        config->operation != RK_MODE_DECRYPT) {
        printf("config->operation error!");
        return TEEC_ERROR_BAD_PARAMETERS;
    }
    if (config->key_len != 16 &&
        config->key_len != 24 &&
        config->key_len != 32) {
        printf("config->key_len error!");
        return TEEC_ERROR_BAD_PARAMETERS;
    }
    if (key_id == RK_OEM_OTP_KEY_FW &&
        config->key_len != 16) {
        printf("config->key_len error!");
        return TEEC_ERROR_BAD_PARAMETERS;
    }
    if (len % AES_BLOCK_SIZE ||
        len > RK_CRYPTO_MAX_DATA_LEN ||
        len == 0) {
```

```
        printf("len error!");
        return TEEC_ERROR_BAD_PARAMETERS;
    }

    res = TEEC_InitializeContext(NULL, &contex);
    if (res != TEEC_SUCCESS) {
        printf("TEEC_InitializeContext failed with code TEEC res= 0x%x", res);
        return res;
    }

    res = TEEC_OpenSession(&contex, &session, &uuid, TEEC_LOGIN_PUBLIC,
                NULL, NULL, &error_origin);
    if (res != TEEC_SUCCESS) {
        printf("TEEC_Opensession failed with code TEEC res= 0x%x origin 0x%x",
            res, error_origin);
        goto out;
    }

    sm.size = len;
    sm.flags = TEEC_MEM_INPUT | TEEC_MEM_OUTPUT;
    res = TEEC_AllocateSharedMemory(&contex, &sm);
    if (res != TEEC_SUCCESS) {
        printf("AllocateSharedMemory ERR! TEEC res= 0x%x", res);
        goto out1;
    }

    memcpy(sm.buffer, src, len);

    memset(&operation, 0, sizeof(TEEC_Operation));
    operation.params[0].value.a       = key_id;
    operation.params[1].tmpref.buffer = config;
    operation.params[1].tmpref.size   = sizeof(rk_cipher_config);
    operation.params[2].memref.parent = &sm;
    operation.params[2].memref.offset = 0;
    operation.params[2].memref.size   = sm.size;

    operation.paramTypes = TEEC_PARAM_TYPES(TEEC_VALUE_INPUT,
                        TEEC_MEMREF_TEMP_INPUT,
                        TEEC_MEMREF_PARTIAL_INOUT,
                        TEEC_NONE);

    res = TEEC_InvokeCommand(&session, CRYPTO_SERVICE_CMD_OEM_OTP_KEY_CIPHER,
                &operation, &error_origin);
    if (res != TEEC_SUCCESS) {
        printf("InvokeCommand ERR! TEEC res= 0x%x, error_origin= 0x%x",
            res, error_origin);
    } else {
        memcpy(dst, sm.buffer, sm.size);
    }

    TEEC_ReleaseSharedMemory(&sm);

out1:
    TEEC_CloseSession(&session);

out:
    TEEC_FinalizeContext(&contex);
    return res;
```

```
}
```

以下是 UserSpace 烧写 OEM Cipher Key 参考 Demo：

```
uint32_t demo_for_rk_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 = rk_write_oem_otp_key(RK_OEM_OTP_KEY0, key, sizeof(key));
    if (res)
        printf("test rk_write_oem_otp_key fail! 0x%08x\n", res);
    else
        printf("test rk_write_oem_otp_key success.\n");
    return res;
}
```

以下是 UserSpace 判断是否已经烧写 OEM Cipher Key 参考 Demo：

```
void demo_for_rk_oem_otp_key_is_written(void)
{
    uint8_t value;
    uint32_t res = rk_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!");
    }
}
```

以下是 RK3588 平台 UserSpace 使用 Hardware Read 功能参考 Demo：

```
uint32_t demo_for_rk_set_oem_hr_otp_read_lock(void)
{
    uint32_t res;

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

以下是 UserSpace 使用 OEM Cipher Key 的参考 Demo:

```
uint32_t demo_for_rk_oem_otp_key_cipher(void)
{
    uint32_t res;
    rk_cipher_config config;
    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 input[16] = {
        0xc9, 0x07, 0x21, 0x05, 0x80, 0x1b, 0x00, 0x44,
        0xac, 0x13, 0xfb, 0x23, 0x93, 0x4a, 0x66, 0xe4,
    };
    uint8_t output[16];
    uint32_t data_len = sizeof(input);

    memset(output, 0, sizeof(output));

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

    res = rk_oem_otp_key_cipher(key_id, &config, input, output, data_len);
    if (res)
        printf("test rk_oem_otp_key_cipher fail! 0x%08x\n", res);
    else
        printf("test rk_oem_otp_key_cipher success.\n");

    return res;
}
```

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。

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。

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

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;

}
```