Rockchip OTP Developer's Guide

ID：RK-KF-YF-147

Release Version：V1.5.0

Release Date：2023-04-13

Security Level: □Top-Secret □Secret □Internal ■Public

Rockchip Electronics Co., Ltd.

No.18 Building, A District, No.89, software Boulevard Fuzhou, Fujian,PRC

Website: www.rock-chips.com

Customer service Tel: +86-4007-700-590

Customer service Fax: +86-591-83951833

Customer service e-Mail: fae@rock-chips.com

Preface

Summary

This document mainly introduces the burning in Rockchip OTP OEM Zone.

Product Version

Chips	Kernel version
Chips for Rockchip	Linux 4.19
Chips for Rockchip	Linux 5.10

Readers

This document is mainly applicable to the following engineers:

Technical Support Engineer

Software Development Engineer

History

Revision	Author	Date	Description
V1.0.0	ZXG	2020-10-18	Original document
V1.0.1	ZXG	2021-02-08	Format revision
V1.1.0	hisping	2022-01-07	Add secure OTP OEM Zone description
V1.2.0	hisping	2022-01-14	Add the description to judge whether the OEM Cipher Key is written
V1.3.0	hisping	2022-01-14	Add the description for OTP Life cycle， Add the description for Protected OEM Zone Write lock
V1.4.0	hisping	2022-03-08	Modify Non-Protected OEM Zone support platform， Modify the description for UserSpace users to use OEM Cipher Keys
V1.5.0	hisping	2023-04-13	Modify Non-Secure OTP description, Add new Secure OTP support platform

Contents

1. Summary

OTP NVM (One Time Programmable Non-Volatile Memory)， non-volatile storage that can only be programmed once. In contrast, FLASH storage can be rewritten many times.

OTP divides the storage area into Secure OTP and Non-Secure OTP， Normal World（such as U-Boot, UserSpace）can directly read the data in the Non-Secure OTP, Normal World has no permission to directly read or write the data in Secure OTP， Generally, Sensitive data is stored in secure OTP， only the secure world（such as Miniloader/SPL， OP-TEE）can directly read and write secure OTP.

The concepts related to secure world and normal world involve TrustZone and TEE knowledge， For details, please refer to《Rockchip_Developer_Guide_TEE_SDK_EN》or ARM official document.

2. Non-Secure OTP

The RK platform Non-Secure OTP reserves an OEM Zone for customers to store customized data， For example: serial number, MAC address, product information, etc.

Customers can read and write OEM OTP through standard file read and write APIs.

2.1 Support platform

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

2.2 Usage

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, Before each OEM Write, enable write to avoid accidental writing.

```
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, The size and offset of the written data need to be aligned by 4 bytes. After the data is written, it will be marked as write protected. The written data write protection will take effect after the next restart.

```
/*
 * @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， Write 0~15 to the position of OEM Zone offset 0

```
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， View the results through the OEM Read or hexdump command， The following shows how to view OEM Zone data through the hexdump command

```
# 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

A variety of different OEM zones are reserved in Secure OTP to meet different user needs.

3.1 Protected OEM Zone

Protected OEM Zone is only used for legal Trust Application (TA application) calls running on OP-TEE OS, Normal world cannot directly read or write Protected OEM Zone， Protected OEM Zone is recommended for sensitive data that you do not want to expose to the normal world. The RK3588 platform also supports turning off the Protected OEM Zone burning function， Once the burn function is turned off, the Protected OEM Zone can no longer be burned.

3.1.1 Support platform

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
RK3528/RK3562	128 Bytes	Not Support

3.1.2 Usage

Users should first refer to 《Rockchip_Developer_Guide_TEE_SDK_EN》document，Compile and run CA TA application under rk_tee_user/ ，Please refer to rk_tee_user/v2/ta/rk_test/rktest_otp.c，You can directly call the following functions in TA if rktest_otp.c file does not exist.

Get 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;

}
```

Read Protected OEM Zone

```
/*
 * read_offset:  offset form 0 to (size - 1)
 * read_data:  please use variables defined in TA
 * read_data_size: read length in bytes
 */
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;

}
```

Write Protected OEM Zone

```
/*
 * write_offset:  offset form 0 to (size - 1)
 * write_data:  please use variables defined in TA
 * write_data_size: write length in bytes
 */
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;

}
```

Turn off the Protected OEM Zone burning function

```
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;

}
```

The following is the reference Demo for TA to use the Protected OEM Zone:

```
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;
}
```

3.2 Non-Protected OEM Zone

Non-Protected OEM Zone can be called by U-Boot and UserSpace, and the data will be exposed in the normal world memory.

Due to the limit size of the Non-Secure OTP and security factors, only some platforms have OEM Zone reserved for the Non-Secure OTP， For platforms which the Non-Secure OTP does not reserve an OEM Zone， Users also need to read and write OTP in U-Boot and UserSpace, so they can use the Non-Protected OEM Zone.

3.2.1 Support platform

Platform	Non-Protected OEM Zone Size
RK3308/RK3326/RK3358/RK3566/RK3568/RK3588	64 Bytes
RK3528/RK3562	32 Bytes

3.2.2 Usage

U-Boot read Non-Protected OEM Zone， Please call trusty_read_oem_ns_otp function in u-boot/lib/optee_clientApi/OpteeClientInterface.c

U-Boot write Non-Protected OEM Zone， Please call trusty_write_oem_ns_otp function in u-boot/lib/optee_clientApi/OpteeClientInterface.c

The following is the reference Demo for U-Boot using Non-Protected OEM Zone：

```
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 users should first refer to the《Rockchip_Developer_Guide_TEE_SDK_EN》document，Compile CA application under rk_tee_user/，Then refer to invoke_otp_ns_read and invoke_otp_ns_write in rk_tee_user/v2/host/rk_test/rktest.c，or directly call the following functions

```
#define STORAGE_CMD_READ_OEM_NS_OTP     13
/* byte_off form 0 to (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 from 0 to (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;
}
```

The following is the reference Demo for UserSpace use Non-Protected OEM Zone:

```
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;
}
```

3.3 OEM Cipher Key

OEM Cipher Key is used to store user keys, which cannot be changed once written， User can use the specified key for encryption and decryption after burning the key， the system only provides a burning interface to ensure that the key is not disclosed， The burning interface and algorithm interface can be called by U-Boot and UserSpace.

3.3.1 Support platform

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
RK3528/RK3562	RK_OEM_OTP_KEY0-3 (16 or 24 or 32 Bytes)	Support

3.3.2 Usage

U-Boot write OEM Cipher Key， please call trusty_write_oem_otp_key function in u-boot/lib/optee_clientApi/OpteeClientInterface.c

key_id structure in function uint32_t trusty_write_oem_otp_key(enum RK_OEM_OTP_KEYID key_id, uint8_t *byte_buf, uint32_t byte_len):

```
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
};
```

Platforms supports RK_OEM_OTP_KEY0、RK_OEM_OTP_KEY1、RK_OEM_OTP_KEY2、RK_OEM_OTP_KEY3；RV1126/RV1109 platform supports additional RK_OEM_OTP_KEY_FW, RK_OEM_OTP_KEY_FW used in BootROM for decrypting Loader firmware， Users can also use this key to process business data or decrypt the Kernel firmware.

The following is the reference Demo for U-Boot burning OEM Cipher Key:

```
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 check whether the OEM Cipher Key has been written， please call trusty_oem_otp_key_is_written function in u-boot/lib/optee_clientApi/OpteeClientInterface.c

The following is the reference Demo for U-Boot check whether the OEM Cipher Key has been written:

```
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!");
    }
}
```

In addition, Some platform also supports the Hardware Read function， Users can call trusty_set_oem_hr_otp_read_lock function in u-boot/lib/optee_clientApi/OpteeClientInterface.c. The CPU can not access to the key after calling this function， The key data does not appear in the memory ， so that the key is isolated from the CPU， The hardware can automatically read the key and send it to the crypto module for encryption and decryption. ***If RK3588 use RK_OEM_OTP_KEY0、RK_OEM_OTP_KEY1、RK_OEM_OTP_KEY2， The CPU's read/write permissions for other OTP data will be changed after calling this function， such as Secure Boot、Security Level and other data will lose the permission to burn. Therefore, the user needs to confirm that the OTP data will not be burned in the future before calling this function. If RK3588 use RK_OEM_OTP_KEY3， Other OTP data read/write permissions will not be affected after calling this function.**

The following is the reference Demo for the hardware read function of the RK3588 platform in U-Boot:

```
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 uses OEM Cipher Key for encryption and decryption， please call trusty_oem_otp_key_cipher function in u-boot/lib/optee_clientApi/OpteeClientInterface.c

The following is the reference Demo for U-Boot using OEM Cipher Key

```
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 burning and using OEM Cipher Key are similar to U-Boot， ***Please refer to the above U-Boot burning and OEM Cipher Key contents for usage note***

For UserSpace users to write and use OEM Cipher Keys，please refer to librkcrypto/demo/demo_otpkey.c, librkcrypto source code and documents《Rockchip_Developer_Guide_Crypto_HWRNG_CN.pdf》have integrated into SDK.

Android platform：librkcrypto source code is under hardware/rockchip/

Linux platform：librkcrypto source code is under external/

3.4 OTP Life Cycle

Some platforms support OTP Life Cycle， Its role is to control the access rights of OTP data in different life cycles.

3.4.1 Support platform

Platform	OTP Life Cycle Type	Description
RK3588	Blank/Tested/Provisioned/Missioned	Blank has the highest read and write permissions, Missioned has the lowest read/write permission, Read and write permissions decrease in sequence. You can choose to enter the low permission stage in the high permission stage, but you cannot enter the high permission stage in the low permission stage. The chip leaves the factory in Provisioned stage, OEM can choose to enter the Missioned stage, After the OEM enters the Missioned stage from the Provisioned stage, some OTP data read/write permissions will change.

3.4.2 permissions change

The following is the list of read and write permissions for RK3588 OTP in the Provisioned and Missioned stage， RW represents read-write and R represents read-only.

Data	Provisioned	Missioned	Description
Secure Boot Enable Flag	RW	R	If you need to use the Secure Boot, you need to enable the Secure Boot before changing the OTP Life Cycle， Secure Boot refer to 《Rockchip_Developer_Guide_Secure_Boot_Application_Note_EN.md》
RSA Public Hash	RW	R	The same as above
Security Level	RW	R	If you need to use the strong and weak security options, you need to select Security Level before changing the OTP Life Cycle， Security Level refer to 《Rockchip_Developer_Guide_TEE_SDK_EN》
OEM Cipher Key0-2	RW	None	See OEM Cipher Key chapter for details
FW encryption key	RW	None	It is used to encrypt Loader firmware， BootRom will use it to decrypt firmware

3.4.3 Usage

OTP Life Cycle can only be modified in the secure world， Please refer to 《Rockchip_Developer_Guide_TEE_SDK_EN》to change the OTP Life Cycle from Provisioned to Missioned， Compile and run CA TA application under rk_tee_user/ ， Then call the following functions in TA.

```
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;

}
```