Smaug-tpm

2022年4月11日 17:09

Modifying the Linux kernel

- **Modifying the Linux kernel device tree**

```
$> vim <Linux kernel installation directory>/linux-5.10.61/arch/arm/boot/dts/stm32mp15xx-dkx.dtsi
edit &spi5 as the following:
&spi5 {
        pinctrl-names = "default", "sleep";
        pinctrl-0 = <&spi5_pins_a>;
        pinctrl-1 = <&spi5_sleep_pins_a>;
        cs-gpios = <&gpiof 3 0>;
        status = "okay";

        slb9670: slb9670@0 {
                status="okay";
                compatible = "infineon,slb9670";
                reg = <0>;
                #address-cells = <1>;
                #size-cells = <0>;
                spi-max-frequency = <32000000>;
        };
};

$> cd <build directory>
$> source <STM32MP1 SDK PATH>/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
$> make ARCH=arm dtbs
$> sudo cp arch/arm/boot/dts/stm32mp157c-dk2.dtb /media/$USER/bootfs
```

- **Configure the Linux kernel Menuconfig**

```
$> cd <build directory>
$> make ARCH=arm menuconfig
Select in the Kernel Configuration:
Device Drivers  --->
   Character devices --->
   <*>   TPM Harware Support --->
         <*>   TPM Interface Specification 1.3 Interface / TPM 2.0 FIFO Interface - (SPI)

$> source <STM32MP1 SDK PATH>/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
$> make ARCH=arm uImage LOADADDR=0xC2000040
$> make ARCH=arm modules
$> make ARCH=arm INSTALL_MOD_PATH="$PWD/../build/install_artifact" modules_install
$> rm install_artifact/lib/modules/5.10.61/build install_artifact/lib/modules/5.10.61/source
$> find install_artifact/ -name "*.ko" | xargs $STRIP --strip-debug --remove-section=.comment --
remove-section=.note --preserve-dates
$> sudo cp arch/arm/boot/uImage /media/$USER/bootfs
$> sudo cp -r install_artifact/lib/modules/* /media/$USER/rootfs/lib/modules/
Board $> depmod -a
Board $> sync
Board $> reboot
```

Modifying the OP-TEE

- **添加系统调用**

  用户空间代码的修改

  1. 修改optee_os/lib/libutee/arch/arm/utee_syscalls_asm.S文件，添加如下内容：

```
// optee_os/lib/libutee/arch/arm/utee_syscalls_asm.S
UTEE_SYSCALL utee_tpm_get_version, TEE_SCN_TPM_GET_VERSION, 1
```

  2. 修改optee_os/lib/libutee/include/utee_syscalls.h文件，添加如下内容，申明上述函数接口，在TA的源代码中包含该头文件后就可调用该接口。

```
// optee_os/lib/libutee/include/utee_syscalls.h
TEE_Result utee_tpm_get_version(void *buf);
```

3. 修改optee_os/lib/libutee/include/tee_syscall_numbers.h文件，添加上述系统调用接口的索引值，并修改TEE_SCN_MAX的值，需要修改和添加的内容如下：

```
// optee_os/lib/libutee/include/tee_syscall_numbers.h
#define TEE_SCN_TPM_GET_VERSION            71
#define TEE_SCN_MAX                71
```

内核空间代码的修改

4. 修改optee_os/core/arch/arm/tee/arch_svc.c文件中系统调用数组变量tee_svc_syscall_table的内容，将上述系统调用对应的内核层接口添加到该数组中，并包含申明该接口的头文件，在该文件中添加的内容如下：

```
// optee_os/core/arch/arm/tee/arch_svc.c
#include <tee/tee_tpm.h>
static const struct syscall_entry tee_svc_syscall_table[] = {
    ......
    SYSCALL_ENTRY(syscall_tpm_get_version),
};
```

- **添加系统服务**

1. 在本示例中建立的系统服务的源代码为tee_tpm.c文件，需将该文件保存到optee_os/core/tee目录中。

```
// optee_os/core/tee/tee_tpm.c
#include <assert.h>
#include <string.h>
#include <optee_rpc_cmd.h>
#include <kernel/thread.h>
#include <kernel/msg_param.h>
#include <tee/tee_svc.h>
#include <mm/tee_mm.h>
#include <mm/mobj.h>
#include <tee/tee_tpm.h>
TEE_Result syscall_tpm_get_version(void *buf)
{
    uint8_t *ree_shm = NULL;
    struct mobj *mobj = NULL;
    TEE_Result res;
    struct thread_param params[2];
    memset(params, 0, sizeof(params));

    params[0].attr = THREAD_PARAM_ATTR_VALUE_IN;
    params[0].u.value.a = OPTEE_TPM_VERSION;

    // 分配共享内存
    mobj = thread_rpc_alloc_payload(4096);
    if (!mobj)
        return TEE_ERROR_OUT_OF_MEMORY;

    if (mobj->size < 4096) {
        res = TEE_ERROR_SHORT_BUFFER;
        goto exit;
    }
    // 获取分配的共享内存的虚拟地址被保存在ree_shm中
    ree_shm = mobj_get_va(mobj, 0);
    // 检查虚拟地址是否有效
    assert(ree_shm);
    params[1].attr = THREAD_PARAM_ATTR_MEMREF_OUT;
    params[1].u.memref.size = 4096;
    params[1].u.memref.offs = 0;
    params[1].u.memref.mobj = mobj;
    res = thread_rpc_cmd(OPTEE_MSG_RPC_CMD_TPM, 2, params);
    if (res != TEE_SUCCESS)
        goto exit;

    //tee_shm = malloc(params[1].u.memref.size);
    //memcpy(tee_shm, ree_shm, params[1].u.memref.size);
    //tee_svc_copy_to_user(buf, tee_shm, params[1].u.memref.size);
    //free(tee_shm);
    memcpy(buf, ree_shm, params[1].u.memref.size);
exit:
```

```
    thread_rpc_free_payload(mobj);
    return res;
}
```

2. 修改optee_os/core/tee目录下的sub.mk文件，将tee_tpm.c文件添加编译系统中。

```
// optee_os/core/tee/sub.mk
srcs-y += tee_tpm.c
```

3. 同时将tee_tpm.h文件保存到optee_os/core/include/tee目录中。

```
// optee_os/core/include/tee/tee_tpm.h
#ifndef TEE_TPM_H
#define TEE_TPM_H
#include <tee_api_types.h>
TEE_Result syscall_tpm_get_version(void *buf);
#endif /* TEE_TPM_H */
```

4. 修改optee_os/core/include/optee_rpc_cmd.h文件增加OPTEE_MSG_RPC_CMD_LOAD_TPM宏：

```
// optee_os/core/include/optee_msg_supplicant.h
/*
 * TPM
 */
#define OPTEE_MSG_RPC_CMD_TPM       60
#define OPTEE_TPM_VERSION           6
```

- **Updating the OP-TEE**

```
• Building the OP-TEE
  <OP-TEE installation directory>/README.HOW_TO.txt helper file tells the instructions.
  $> export FIP_DEPLOYDIR_ROOT=$PWD/../../FIP_artifacts
  $> source <STM32MP1 SDK PATH>/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
  $> make -f $PWD/../Makefile.sdk CFG_EMBED_DTB_SOURCE_FILE=stm32mp157c-dk2 all
  The generated FIP images are available in $FIP_DEPLOYDIR_ROOT/fip
• Updating the SDK
  $> cp -r $PWD/../build/stm32mp157c-dk2/export-ta_arm32/* <STM32MP1 SDK PATH>/sysroots/cortexa7t2hf-
  neon-vfpv4-ostl-linux-gnueabi/usr/include/optee/export-user_ta
• Deploying the OP-TEE
  Replace the fip-stm32mp157c-dk2-optee.bin and recreate the image.
  $> cp $FIP_DEPLOYDIR_ROOT/fip/fip-stm32mp157c-dk2-optee.bin <STM32MP1 IMAGE PATH>/stm32mp1/fip
• Create the Image
  $> cd stm32mp1-openstlinux-5.10-dunfell-mp1-21-11-17/images/stm32mp1/scripts/
  $> ./create_sdcard_from_flashlayout.sh ../flashlayout_st-image-
  weston/optee/FlashLayout_sdcard_stm32mp157c-dk2-optee.tsv
• Image flashing
  $> sudo dd if=../flashlayout_st-image-weston/extensible/../../FlashLayout_sdcard_stm32mp157c-dk2-
  optee.raw of=/dev/sdb bs=8M conv=fdatasync status=progress
```

Modifying the OPTEE-CLIENT

- **添加RPC调用**

1. 修改optee_client/tee-supplicant/src/optee_msg_supplicant.h文件增加OPTEE_MSG_RPC_CMD_LOAD_TPM宏：

```
// optee_client/tee-supplicant/src/optee_msg_supplicant.h
/*
 * TPM
 */
#define OPTEE_MSG_RPC_CMD_TPM       60
#define OPTEE_TPM_VERSION           6
```

2. 修改optee_client/tee-supplicant/src/tee_supplicant.c文件增加load_tpm函数：

```
// optee_client/tee-supplicant/src/tee_supplicant.c
#include <tee_tpm.h>
static uint32_t load_tpm(size_t num_params, struct tee_ioctl_param *params)
{
    struct param_value *val_cmd = NULL;
    TEEC_SharedMemory shm_ta;
    memset(&shm_ta, 0, sizeof(shm_ta));
```

```
    int size = 0;
    get_value(num_params, params, 0, &val_cmd);  //
    get_param(num_params, params, 1, &shm_ta);  //

    invoke_tpm(val_cmd->a, &size, (char *)shm_ta.buffer);
    MEMREF_SIZE(params + 1) = size;
    return TEEC_SUCCESS;
}

static bool process_one_request(struct thread_arg *arg)
{
    ......
    switch (func) {
    case OPTEE_MSG_RPC_CMD_TPM:
        ret = load_tpm(num_params, params);
        break;
    ......
    }
}
```

3. 添加optee_client/tee-supplicant/src/tee_tpm.c文件处理tpm相关功能：

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <fcntl.h>
#include <optee_msg_supplicant.h>
#include <tee_tpm.h>
#include <unistd.h>
#include <inttypes.h>
static int tpmtool_transmit(const uint8_t *buf, ssize_t length, uint8
_t *response, ssize_t *resp_length)
{
    // ---------- Transmit command given in buf to device with handle given in dev_tpm ----------
    int ret_val = EXIT_SUCCESS; // Return value.
    int dev_tpm = -1;       // TPM device handle.
    ssize_t transmit_size = 0;  // Amount of bytes sent to / received from the TPM.
    memset(response, 0, *resp_length);

    // ---------- Open TPM device ----------
    dev_tpm = open("/dev/tpm0", O_RDWR);
    /*
    if (-1 == dev_tpm)
    {
        ret_val = errno;
        fprintf(stderr, "Error opening the device.\n");
        break;
    }
    */
    // Send request data to TPM.
    transmit_size = write(dev_tpm, buf, length);
    /*
    if (transmit_size == ERR_COMMUNICATION || length != transmit_size)
    {
        ret_val = errno;
        fprintf(stderr, "Error sending request to TPM.\n");
        break;
    }
    */
    // Read the TPM response header.
    transmit_size = read(dev_tpm, response, TPM_RESP_MAX_SIZE);
    /*if (transmit_size == ERR_COMMUNICATION)
    {
        ret_val = errno;
        fprintf(stderr, "Error reading response from TPM.\n");
        break;
    }
    */

    // Update response buffer length with value of data length returned by TPM.
    *resp_length = transmit_size;
    // ---------- Close TPM device ----------
    if (-1 != dev_tpm)
    {
        // Close file handle.
        close(dev_tpm);
```

```
        // Invalidate file handle.
        dev_tpm = -1;
    }
    return ret_val;
}
static int buf_to_uint64(uint8_t *input_buffer, uint32_t offset, uint32_t length, uint64
_t *output_value)
{
    int ret_val = EXIT_SUCCESS; // Return value.
    uint32_t i = 0;         // Loop variable.
    uint64_t tmp = 0;       // Temporary variable for value calculation.
    *output_value = 0;
    for (i = 0; i < length; i++)
    {
        tmp = (tmp << 8) + input_buffer[offset + i];
    }
    *output_value = tmp;
    return ret_val;
}
static int print_capability_flags(uint8_t *response_buf, uint8_t cap_selector, int *n, char **outbuf)
{
    int ret_val = EXIT_SUCCESS; // Return value.
    uint64_t propertyValue = 0; // Value of the read property.
    uint64_t propertyKey = 0;   // Key of the property.
    int tmp = 0;            // Temporary buffer.
    do
    {
        if(cap_selector == PT_FIXED_SELECTOR)
        {
            (*n) += sprintf((*outbuf) + (*n), "\nTPM capability information of fixed properties:\n");
            (*n) += sprintf((*outbuf) + (*n), "=================================================
==\n");
            for(int x = 0x13; x<(TPM_RESP_MAX_SIZE-8); x+=8)
            {   //Iterate over each property key/value pair
                ret_val = buf_to_uint64(response_buf, x, 4, &propertyKey);
                ret_val = buf_to_uint64(response_buf, x+4, 4, &propertyValue);
                switch(propertyKey)
                {
                    case 0x100:
                        (*n) += sprintf((*outbuf) + (*n), "TPM_PT_FAMILY_INDICATOR:        %c%c%c%c
\n", response_buf[x+4], response_buf[x+5], response_buf[x+6], response_buf[x+7]);
                        break;
                    case 0x100+1:
                        (*n) += sprintf((*outbuf) + (*n), "TPM_PT_LEVEL:                   %" PRIu64 "
\n", propertyValue);
                        break;
                    case 0x100+2:
                        (*n) += sprintf((*outbuf) + (*n), "TPM_PT_REVISION:                %" PRIu64 "
\n", propertyValue);
                        break;
                    case 0x100+3:
                        (*n) += sprintf((*outbuf) + (*n), "TPM_PT_DAY_OF_YEAR:             %" PRIu64 "
\n", propertyValue);
                        break;
                    case 0x100+4:
                        (*n) += sprintf((*outbuf) + (*n), "TPM_PT_YEAR:                    %" PRIu64 "
\n", propertyValue);
                        break;
                    case 0x100+5:
                        (*n) += sprintf((*outbuf) + (*n), "TPM_PT_MANUFACTURER:            %c%c%c%c
\n", response_buf[x+4], response_buf[x+5], response_buf[x+6], response_buf[x+7]);
                        break;
                    case 0x100+6:
                        (*n) += sprintf((*outbuf) + (*n), "TPM_PT_VENDOR_STRING:           ");
                        (*n) += sprintf((*outbuf) + (*n), "%c%c%c%c",  response_buf[x+
4], response_buf[x+5], response_buf[x+6], response_buf[x+7]);
                        break;
                    case 0x100+7: // it is assumed that TPM_PT_VENDOR_STRING_2 follows _1
                        (*n) += sprintf((*outbuf) + (*n), "%c%c%c%c",  response_buf[x+
4], response_buf[x+5], response_buf[x+6], response_buf[x+7]);
                        break;
                    case 0x100+8:
                        (*n) += sprintf((*outbuf) + (*n), "%c%c%c%c",  response_buf[x+
4], response_buf[x+5], response_buf[x+6], response_buf[x+7]);
                        break;
                    case 0x100+9:
                        (*n) += sprintf((*outbuf) + (*n), "%c%c%c%c\n",  response_buf[x+
4], response_buf[x+5], response_buf[x+6], response_buf[x+7]);
```

```
                        break;
                    case 0x100+10:
                        (*n) += sprintf((*outbuf) + (*n), "TPM_PT_VENDOR_TPM_TYPE:         %" PRIu64 "
\n", propertyValue);
                        break;
                    case 0x100+11:
                        // special handling for firmware version XX.xx.xxxx.x
                        ret_val = buf_to_uint64(response_buf, x+4, 2, &propertyValue);
                        (*n) += sprintf((*outbuf) + (*n), "TPM_PT_FIRMWARE_VERSION:        %" PRIu64 "
", propertyValue);
                        ret_val = buf_to_uint64(response_buf, x+6, 2, &propertyValue);
                        (*n) += sprintf((*outbuf) + (*n), ".%" PRIu64 "", propertyValue);
                        break;
                    case 0x100+12:
                        // special handling for firmware version XX.xx.xxxx.x
                        ret_val = buf_to_uint64(response_buf, x+
4, 2, &propertyValue); // Check for output version.
                        if (2 <= propertyValue) // Infineon custom format
                        {
                            ret_val = buf_to_uint64(response_buf, x+5, 2, &propertyValue);
                            (*n) += sprintf((*outbuf) + (*n), ".%" PRIu64 "", propertyValue);
                            ret_val = buf_to_uint64(response_buf, x+7, 1, &propertyValue);
                            (*n) += sprintf((*outbuf) + (*n), ".%" PRIu64 "\n", propertyValue);
                        }
                        else
                        {
                            ret_val = buf_to_uint64(response_buf, x+4, 4, &propertyValue);
                            (*n) += sprintf((*outbuf) + (*n), ".%" PRIu64 "\n", propertyValue);
                        }
                        break;
                    case 0x100+24:
                        (*n) += sprintf((*outbuf) + (*n), "\nTPM_PT_MEMORY:\n");
                        (*n) += sprintf((*outbuf) + (*n), "==========================================
==============\n");
                        tmp = ((propertyValue & (1<<0)) == 0? 0:1); // Check bit 0 value.
                        (*n) += sprintf((*outbuf) + (*n), "Shared RAM:                     %i %
s", (tmp), ((tmp)? "SET\n" : "CLEAR\n"));
                        tmp = ((propertyValue & (1<<1)) == 0? 0:1); // Check bit 1 value.
                        (*n) += sprintf((*outbuf) + (*n), "Shared NV:                      %i %
s", (tmp), ((tmp)? "SET\n" : "CLEAR\n"));
                        tmp = ((propertyValue & (1<<2)) == 0? 0:1); // Check bit 2 value.
                        (*n) += sprintf((*outbuf) + (*n), "Object Copied To Ram:           %i %
s", (tmp), ((tmp)? "SET\n" : "CLEAR\n"));
                        //bit 31:3 = reserved
                        break;
                    case 0x200:
                        (*n) += sprintf((*outbuf) + (*n), "\nTPM_PT_PERMANENT:\n");
                        (*n) += sprintf((*outbuf) + (*n), "==========================================
==============\n");
                        tmp = ((propertyValue & (1<<0)) == 0? 0:1); // Check bit 0 value.
                        (*n) += sprintf((*outbuf) + (*n), "Owner Auth Set:                 %i %
s", (tmp), ((tmp)? "SET\n" : "CLEAR\n"));
                        tmp = ((propertyValue & (1<<1)) == 0? 0:1); // Check bit 1 value.
                        (*n) += sprintf((*outbuf) + (*n), "Sendorsement Auth Set:          %i %
s", (tmp), ((tmp)? "SET\n" : "CLEAR\n"));
                        tmp = ((propertyValue & (1<<2)) == 0? 0:1); // Check bit 2 value.
                        (*n) += sprintf((*outbuf) + (*n), "Lockout Auth Set:               %i %
s", (tmp), ((tmp)? "SET\n" : "CLEAR\n"));
                        //bit 7:3 = reserved
                        tmp = ((propertyValue & (1<<8)) == 0? 0:1); // Check bit 8 value.
                        (*n) += sprintf((*outbuf) + (*n), "Disable Clear:                  %i %
s", (tmp), ((tmp)? "SET\n" : "CLEAR\n"));
                        tmp = ((propertyValue & (1<<9)) == 0? 0:1); // Check bit 9 value.
                        (*n) += sprintf((*outbuf) + (*n), "In Lockout:                     %i %
s", (tmp), ((tmp)? "SET\n" : "CLEAR\n"));
                        tmp = ((propertyValue & (1<<10)) == 0? 0:1); // Check bit 10 value.
                        (*n) += sprintf((*outbuf) + (*n), "TPM Generated EPS:              %i %
s", (tmp), ((tmp)? "SET\n" : "CLEAR\n"));
                        //bit 31:11 = reserved
                        break;
                    default:
                        // Unknown attribute - ignore
                        break;
                }
            }
        }
        else if (cap_selector == PT_VAR_SELECTOR)
        {
```

```
            (*n) += sprintf((*outbuf) + (*n), "\nTPM capability information of variable properties:
\n");
            for(int x = 0x13; x<TPM_RESP_MAX_SIZE-8; x+=8)
            {   //Iterate over each property key/value pair
                ret_val = buf_to_uint64(response_buf, x, 4, &propertyKey);
                ret_val = buf_to_uint64(response_buf, x+4, 4, &propertyValue);
                switch(propertyKey)
                {
                    case 0x201:
                        (*n) += sprintf((*outbuf) + (*n), "\nTPM_PT_STARTUP_CLEAR:\n");
                        (*n) += sprintf((*outbuf) + (*n), "==========================================
==============\n");
                        tmp = ((propertyValue & (1<<0)) == 0? 0:1); // Check bit 0 value.
                        (*n) += sprintf((*outbuf) + (*n), "Ph Enable:                       %i %
s", (tmp), ((tmp)? "SET\n" : "CLEAR\n"));
                        tmp = ((propertyValue & (1<<1)) == 0? 0:1); // Check bit 1 value.
                        (*n) += sprintf((*outbuf) + (*n), "Sh Enable:                       %i %
s", (tmp), ((tmp)? "SET\n" : "CLEAR\n"));
                        tmp = ((propertyValue & (1<<2)) == 0? 0:1); // Check bit 2 value.
                        (*n) += sprintf((*outbuf) + (*n), "Eh Enable:                       %i %
s", (tmp), ((tmp)? "SET\n" : "CLEAR\n"));
                        //bit 30:3 = reserved
                        tmp = ((propertyValue & (1<<31)) == 0? 0:1); // Check bit 31 value.
                        (*n) += sprintf((*outbuf) + (*n), "Orderly:                        %i %
s", (tmp), ((tmp)? "SET\n" : "CLEAR\n"));
                        break;
                    default:
                        // Unknown attribute - ignore
                        break;
                }
            }
        }
    } while (0);
    return ret_val;
}
static int response_print(uint8_t *response_buf, int cmd, int *n, char **outbuf)
{
    int ret_val = EXIT_SUCCESS; // Return value.
    switch (cmd) {
    case OPTEE_TPM_VERSION: // Print the fixed capability flags.
        ret_val = print_capability_flags(response_buf, PT_FIXED_SELECTOR, n, outbuf);
        break;
    default:
        break;
    }
    return ret_val;
}
int invoke_tpm(int cmd, int *size, char *outbuf)
{
    int ret_val = EXIT_SUCCESS;     // Return value.
    uint8_t *tpm_response_buf = NULL;   // Buffer for TPM response.
    ssize_t tpm_response_buf_size = 0;  // Size of tpm_response_buf.
    int n = 0;  // control sprintf to move position of outbuf
    // ---------- Allocate memory for buffer containing TPM response ----------
    tpm_response_buf_size = TPM_RESP_MAX_SIZE;
    tpm_response_buf = malloc(tpm_response_buf_size);
    // MALLOC_ERROR_CHECK(tpm_response_buf);
    memset(tpm_response_buf, 0xFF, tpm_response_buf_size);
    switch (cmd) {
    case OPTEE_TPM_VERSION:
        ret_val = tpmtool_transmit(tpm2_getcapability_fixed, sizeof(tpm2
_getcapability_fixed), tpm_response_buf, &tpm_response_buf_size);
    default:
        break;
    }
    // Check for transmission errors.
    // Transmission has been successful, now get TPM return code from TPM response.

    // Print TPM response
    ret_val = response_print(tpm_response_buf, cmd, &n, &outbuf);
    // ---------- Cleanup ----------
    MEMSET_FREE(tpm_response_buf, tpm_response_buf_size);
    n += sprintf(outbuf + n, "\n");
    *size = n;
    *size += 1;
    return ret_val;
}
```

4. 同时添加optee_client/tee-supplicant/src/tee_tpm.h头文件：

```
#ifndef TEE_TPM_H
#define TEE_TPM_H
#define TPM_RESP_MAX_SIZE       4096    ///< This is the maximum possible TPM response size in bytes.
// TPM_PT constants
#define PT_FIXED_SELECTOR       1       ///< Fixed GetCapability Flags
#define PT_VAR_SELECTOR         2       ///< Variable GetCapability Flags
//-------------"Macros"-------------
#define MEMSET_FREE(x, y) if (NULL !
= x) { memset(x, 0, y); free(x); x = NULL; } ///< Sets memory to 0, frees memory and sets pointer to N
ULL.
typedef unsigned char   uint8_t;
static const uint8_t tpm2_getcapability_fixed[] ={
    0x80, 0x01,         // TPM_ST_NO_SESSIONS
    0x00, 0x00, 0x00, 0x16,     // commandSize
    0x00, 0x00, 0x01, 0x7A,     // TPM_CC_GetCapability
    0x00, 0x00, 0x00, 0x06,     // TPM_CAP_TPM_PROPERTIES (Property Type: TPM_PT)
    0x00, 0x00, 0x01, 0x00,     // Property: TPM_PT_FAMILY_INDICATOR: PT_GROUP * 1 + 0
    0x00, 0x00, 0x00, 0x66      // PropertyCount 102 (from 100 - 201)
};
int invoke_tpm(int cmd, int *size, char *outbuf);
#endif /* TEE_TPM_H */
```

5. 修改optee_client/tee-supplicant/Makefile文件增加需要编译的源文件：

```
TEES_SRCS   := tee_supplicant.c \
           teec_ta_load.c \
           tee_supp_fs.c \
           rpmb.c \
           handle.c \
           tee_tpm.c
```

- **Updating the OPTEE-CLIENT**

```
•Building the OPTEE-CLIENT
 $> source <STM32MP1 SDK PATH>/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
 $> make
•Deploying the OPTEE-CLIENT
 Replace the tee-supplicant in board.
 $> scp out/tee-supplicant/tee-supplicant root@<ip of board>:/usr/bin
```