玄铁安全系统安全应用开发手册

文档编号 STG30002

版权声明

The purchased products, services and features are stipulated by the contract made between T-HEAD and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.
The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

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

平头哥半导体有限公司 T-HEAD Semiconductor Co.,LTD
地址:杭州市余杭区向往街1122号欧美金融城(EFC)英国中心西楼T6 43层 邮编: 311121
网址:www.T-head.cn

修订历史

日期	版本	说明	作者

20230109	v1.1	更新了工具链	柏强
20220327	v1.0	发布	柏强
20220303	v0.1	初版	柏强

目录

[TOC]

1.引言

1.1 面向对象

该文档的面向对象是安全应用的开发者。默认开发者都熟悉linux开发环境，了解GP API接口使用方法。

1.2 编写目的

安全应用SDK（以下简称Tsec_sdk）是支持GP接口标准的，提供给用户开发私有的安全应用（TA/CA）的工具包。本文档从环境搭建，开发方法以及测试验证步骤提供了详细的指导说明。开发者可以按照本文档介绍的方法流程，结合自己的需求开发验证自己的安全应用。

2. 环境搭建

2.1 硬件环境

安装有ubuntu/linux的服务器/操作系统

2.2 软件环境

2.2.1 工具链

编译安全应用的工具链版本需要确保和linux kernel的工具链一致，以免一些不必要的错误。

版本号： v2.2.4

下载链接：

https://occ-oss-prod.oss-cn-hangzhou.aliyuncs.com/resource//1663142462244/Xuantie-900-gcc-linux-5.10.4-glibc-i386-V2.6.1-20220906.tar.gz

2.2.2 依赖程序包安装

编译安全应用需要用到一些工具包，为了确保编译顺利通过，需要用户提前安装以下工具包，安装方法如下：

```bash
sudo apt-get install android-tools-adb android-tools-fastboot autoconf \
    automake bc bison build-essential ccache cscope curl device-tree-compiler \
    expect flex ftp-upload gdisk iasl libattr1-dev libcap-dev libepoxy-dev \
    libfdt-dev libftdi-dev libglib2.0-dev libgmp-dev libhidapi-dev \
    libmpc-dev libncurses5-dev libpixman-1-dev libssl-dev libtool make \
    mtools netcat ninja-build python-crypto python3-crypto python-pyelftools \
    python3-pycryptodome python3-pyelftools python-serial python3-serial \
    rsync unzip uuid-dev xdg-utils xterm xz-utils zlib1g-dev \
    libdaxctl-dev libpmem-dev \
```

3. SDK 环境介绍

```Bash
.
|-- export
|   |-- tee-client
|   |   |-- include
|   |   `-- lib
|   |       |-- android
|   |       `-- linux
|   `-- ta-rv64
`-- demo
    |-- trng
    |   |-- ta
    |   `-- host
    |-- secure_storage
    |   |-- ta
    |   `-- host
    |-- hello_world
    |   |-- ta
    |   `-- host
    |-- hash
    |   |-- ta
    |   `-- host
    |-- efuse
    |   `-- host
    |-- ecdsa
    |   |-- ta
    |   `-- host
    |-- ecdh
    |   |-- ta
    |   `-- host
    |-- dsa
    |   |-- ta
    |   `-- host
    |-- curve25519
    |   |-- ta
    |   `-- host
    |-- config.mk
    |-- cipher
    |   |-- ta
    |   `-- host
    |-- acipher
    |   |-- ta
    |   `-- host
    |-- Makefile
    `-- Android.mk

```

安全应用SDK，release的目录结构如上，其中：

- demo

  存放的是CA/TA的参考例子。每个demo下host 文件夹下存放的是参考CA，ta目录存放的是参考TA

  .demo/trng: 随机数调用示例
  .demo/secure_storage: 安全存储调用示例
  .demo/hello_world: 最简单的hello world示例
  .demo/trng: 随机数调用示例
  .demo/hash: 哈希算法调用示例
  .demo/efuse: efuse调用示例
  .demo/ecdsa: ecdsa算法调用示例
  .demo/ecdh: ecdh算法调用示例
  .demo/dsa: dsa算法调用示例
  .demo/curve25519: 25519算法调用示例
  .demo/cipher: 对称算法调用示例
  .demo/acipher: 非堆成算法调用示例

- export

  存放的是开发CA/TA所需要的一些依赖，如，头文件，库，以及编译脚本。其中：

  - ta-rv64 为编译TA所需要的依赖
  - tee-client/lib/linux 为编译linux CA所需要的依赖so
  - tee-client/lib/android 为编译android CA所需要的依赖so

4. 安全应用开发步骤

4.1 CA开发

CA 是运行在REE侧的应用程序。用户可以通过调用GP 定义client API接口调用TA。

4.1.1 代码开发

CA的代码与普通的linux应用程序的开发没什么特别大的差别，唯一的差别是在API接口上集成了GP的Client API。开发者可以按照自身的开发需求在需要的时候按照Client API的调用标准（详细说明请参考GP发布的TEE Client API Specification）去调用TA。如demo中的例子，TEEC_InitializeContext -> TEEC_OpenSession -> TEEC_InvokeCommand -> TEEC_CloseSession。

```
/* Initialize a context connecting us to the TEE */
res = TEEC_InitializeContext(NULL, &ctx);
if (res != TEEC_SUCCESS)
        errx(1, "TEEC_InitializeContext failed with code 0x%x", res);

/*
 * Open a session to the "hello world" TA, the TA will print "hello
 * world!" in the log when the session is created.
 */
res = TEEC_OpenSession(&ctx, &sess, &uuid,
                       TEEC_LOGIN_PUBLIC, NULL, NULL, &err_origin);
if (res != TEEC_SUCCESS)
        errx(1, "TEEC_Opensession failed with code 0x%x origin 0x%x",
                res, err_origin);

/*
 * Execute a function in the TA by invoking it, in this case
 * we're incrementing a number.
 *
 * The value of command ID part and how the parameters are
 * interpreted is part of the interface provided by the TA.
 */

/* Clear the TEEC_Operation struct */
memset(&op, 0, sizeof(op));

/*
 * Prepare the argument. Pass a value in the first parameter,
 * the remaining three parameters are unused.
 */
op.paramTypes = TEEC_PARAM_TYPES(TEEC_VALUE_INOUT, TEEC_NONE,
                                 TEEC_NONE, TEEC_NONE);
op.params[0].value.a = 42;

/*
 * TA_HELLO_WORLD_CMD_INC_VALUE is the actual function in the TA to be
 * called.
 */
printf("Invoking TA to increment %d\n", op.params[0].value.a);
res = TEEC_InvokeCommand(&sess, TA_HELLO_WORLD_CMD_INC_VALUE, &op,
                         &err_origin);
if (res != TEEC_SUCCESS)
        errx(1, "TEEC_InvokeCommand failed with code 0x%x origin 0x%x",
                res, err_origin);
printf("TA incremented value to %d\n", op.params[0].value.a);

/*
 * We're done with the TA, close the session and
 * destroy the context.
 *
 * The TA will print "Goodbye!" in the log when the
 * session is closed.
 */

TEEC_CloseSession(&sess);
```

4.1.2 编译选项修改

4.1.2.1 linux编译选项

修改交叉编译器路径，修改demo/config.mk中的CROSS_COMPILE变量

```
## Toolchain
## Install your compiler path here

CROSS_COMPILE := /home/xiaojt/toolchain_2.6.2/Xuantie-900-gcc-linux-5.10.4-glibc-x86_64-V2.6.2-light/bin/riscv64-unknown-linux-gnu-
```

修改其它编译选项，修改demo/【要修改的demo】/host/Makefile

```
-include ../../config.mk

## TEE export path
TEEC_EXPORT ?= ../../../export/tee-client-linux

CC      = $(CROSS_COMPILE)gcc
LD      = $(CROSS_COMPILE)ld
AR      = $(CROSS_COMPILE)ar
NM      = $(CROSS_COMPILE)nm
OBJCOPY = $(CROSS_COMPILE)objcopy
OBJDUMP = $(CROSS_COMPILE)objdump
READELF = $(CROSS_COMPILE)readelf

OBJS = main.o

CFLAGS += -Wall -I../ta/include -I$(TEEC_EXPORT)/include -I./include
#Add/link other required libraries here
LDADD += -lteec -L$(TEEC_EXPORT)/lib

BINARY = hello_world

.PHONY: all
all: $(BINARY)
    echo $(CROSS_COMPILE)

$(BINARY): $(OBJS)
    echo $(CROSS_COMPILE)
    $(CC) $(LDFLAGS) -o $@ $< $(LDADD)

.PHONY: clean
clean:
    rm -f $(OBJS) $(BINARY)

%.o: %.c
    $(CC) $(CFLAGS) -c $< -o $@
```

参考例子中Makefile如上图所示，一些关键变量说明如下：

- TEEC_EXPORT
  为上面SDK开发环境介绍中的export目录，如果目录结构有改动，需要调整这个路径
- OBJS

为需要编译对象的目标文件

- CFLAGS
  为编译参数
- LDADD
  为链接相关的编译参数
- BINARY
  为要生成的目标文件

4.1.2.2 android编译选项

修改demo/【要修改的demo】/host/Android.mk

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_SRC_FILES += main.c
LOCAL_C_INCLUDES := $(LOCAL_PATH)/../../../export/tee-client-android/include \
                    $(LOCAL_PATH)/../ta/include

LOCAL_LDFLAGS := $(LOCAL_PATH)/../../../export/tee-client-android/lib/libteec.so

LOCAL_MODULE := hello_world
LOCAL_MODULE_TAGS := optional
LOCAL_VENDOR_MODULE := true

include $(BUILD_EXECUTABLE)
```

参考例子中Android.mk如上图所示

- LOCAL_SRC_FILES
  定义了编译的源码文件列表
- LOCAL_C_INCLUDES
  定义了include目录
- LOCAL_LDFLAGS
  定义了引用的库
- LOCAL_MODULE
  编译出来的目标文件名称

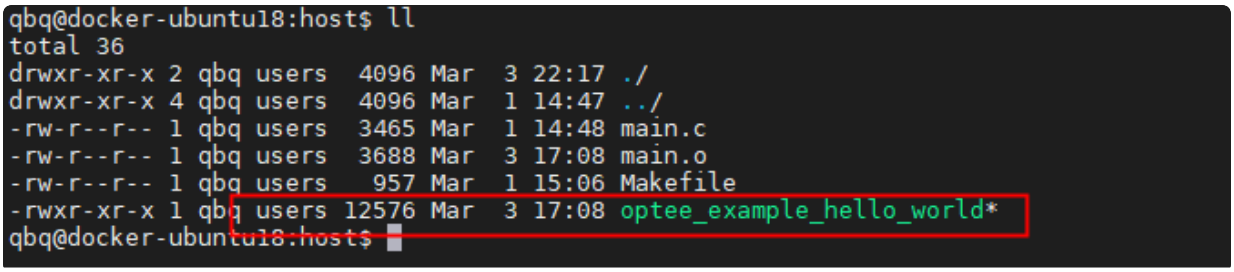
4.1.3 编译

4.1.3.1 linux编译

编译命令：

```bash
make
```

编译成功后，会生成目标二进制文件：



4.1.3.2 android 编译

- step1. 下载安卓源码（以仓库地址是https://yl-gerrit.eng.t-head.cn/thead-aosp/platform/manifest -b riscv64-android-12.0.0_dev为例，实际按照实际仓库地址进行修改），如果已经下载过就不需要再下载了，下载方法如下

```plain text
mkdir riscv-android-src && cd riscv-android-src
repo init -u https://yl-gerrit.eng.t-head.cn/thead-aosp/platform/manifest -b riscv64-android-12.0.0_dev
repo init -u ssh://【gerrit 用户名】@yl-gerrit.eng.t-head.cn:29518/thead-aosp/platform/manifest -b riscv64-android-12.0.0_dev
repo sync
```

- step2. 将Tsec_sdk目录拷贝到安卓源码vendor/thead/private路径下
- step3. 在安卓源码根目录下执行以下命令load编译环境

```bash
source ./build/envsetup.sh
lunch evb_light-userdebug
```

- step4. 执行编译ca

编译全部ca:

```Bash
cd vendor/thead/private/Tsec_sdk
mm
```

编译某个ca(比如hello_world):

```Bash
cd vendor/thead/private/Tsec_sdk/hello_wolrd
mm
```

- step5. 编译成功后可以在对应的host目录下找到生成的可执行文件，比如hello_world的生成文件位于out/target/product/evb_light/vendor/bin/hello_world

4.2 TA 开发

TA是运行TEE-OS的安全应用。在安全应用中，用户可以通过GP internal API接口调用TEE-OS的相关驱动或者其它TA。

4.2.1 TA所必须的文件

要编译TA以下文件必须要实现：

- Makefile -- 编译用的Makefile文件
- sub.mk -- 定义要编译的文件
- user_ta_header_defines.h -- 当前要编译的user ta头文件，里面定义了当前TA的属性
- 实现TA功能的.c 文件，该文件中必须要实现以下GP 定义的函数（具体函数定义可以参考GP internal API手册）
  - TA_CreateEntryPoint()
  - TA_DestroyEntryPoint()
  - TA_OpenSessionEntryPoint()
  - TA_CloseSessionEntryPoint()
  - TA_InvokeCommandEntryPoint()

可以参考demo中TA如下：

```Bash
.
├── Android.mk.DD
├── hello_world_ta.c
├── include
│   └── hello_world_ta.h
├── Makefile
├── sub.mk
└── user_ta_header_defines.h

```

4.2.2 TA 开发步骤

4.2.2.1 按照需求开发TA

用户在开发TA的时候，以下函数必须要实现：

```
TEE_Result TA_CreateEntryPoint(void)
{
/* Allocate some resources, init something, ... */
...
/* Return with a status */
return TEE_SUCCESS;
}
void TA_DestroyEntryPoint(void)
{
/* Release resources if required before TA destruction */
...
}
TEE_Result TA_OpenSessionEntryPoint(uint32_t ptype,
TEE_Param param[4],
void **session_id_ptr)
{
/* Check client identity, and alloc/init some session resources if any */
...
/* Return with a status */
return TEE_SUCCESS;
}
void TA_CloseSessionEntryPoint(void *sess_ptr)
{
/* check client and handle session resource release, if any */
...
}
TEE_Result TA_InvokeCommandEntryPoint(void *session_id,
uint32_t command_id,
uint32_t parameters_type,
TEE_Param parameters[4])
{
/* Decode the command and process execution of the target service */
...
/* Return with a status */
return TEE_SUCCESS;
}
```

4.2.2.2 定义TA属性

TA的属性定义在头文件：user_ta_header_defines.h中里面必须包含：

- TA_UUID

  可以通过以下脚本生成UUID：

```bash
python -c "import uuid; u=uuid.uuid4(); print(u); \
n = [', 0x'] * 11; \
n[::2] = ['{:12x}'.format(u.node)[i:i + 2] for i in range(0, 12, 2)]; \
print('\n' + '#define TA_UUID\n\t{ ' + \
'0x{:08x}'.format(u.time_low) + ', ' + \
'0x{:04x}'.format(u.time_mid) + ', ' + \
'0x{:04x}'.format(u.time_hi_version) + ', \\ \n\n\t\t{ ' + \
'0x{:02x}'.format(u.clock_seq_hi_variant) + ', ' + \
'0x{:02x}'.format(u.clock_seq_low) + ', ' + \
'0x' + ''.join(n) + '} }')"
```

运行该命令得到如下结果（因为uuid是随机的，所以每次运行的结果是不一样的，以下参数仅供参考）：

```bash
7e5fd739-7a76-4391-9b99-c43a10a817fb

#define TA_UUID
        { 0x7e5fd739, 0x7a76, 0x4391, \

                { 0x9b, 0x99, 0xc4, 0x3a, 0x10, 0xa8, 0x17, 0xfb} }
```

- TA_FLAGS
  定义TA的属性，如TA_FLAG_USER_MODE，TA_FLAG_EXEC_DDR，TA_FLAG_SINGLE_INSTANCE，具体的含义可参考GP internal API 手册
- TA_STACK_SIZE
  定义当前TA栈的大小，建议值2*1024
- TA_DATA_SIZE
  定义当前TA堆的大小，建议用32*1024

4.2.2.3 修改Makefile

```
CFG_TEE_TA_LOG_LEVEL ?= 4

CROSS_COMPILE=~/work/src/xuantie-tee/toolchains/riscv64/bin/riscv64-linux-gnu-
TA_DEV_KIT_DIR=../../export/ta-rv64
# The UUID for the Trusted Application
BINARY=8aaaf200-2450-11e4-abe2-0002a5d5c51b

-include $(TA_DEV_KIT_DIR)/mk/ta_dev_kit.mk

ifeq ($(wildcard $(TA_DEV_KIT_DIR)/mk/ta_dev_kit.mk), )
clean:
        @echo 'Note: $$(TA_DEV_KIT_DIR)/mk/ta_dev_kit.mk not found, cannot clean TA'
        @echo 'Note: TA_DEV_KIT_DIR=$(TA_DEV_KIT_DIR)'
endif
~
~
~
~
```

Makefile文件格式如上，其中：

- CFG_TEE_TA_LOG_LEVEL

  当前TA的log级别，参数越大，打印级别越高，如果配置成0，会关闭当前TA的log

- CROSS_COMPILE

  编译TA用到的工具链，开发者需要替换自己工作环境下工具链

- BINARY

  当前TA生成的目标文件。注意：这个参数必须和上面定义在TA_UUID保持一致，如

```bash
#define TA_UUID  { 0x7e5fd739, 0x7a76, 0x4391, \
                 { 0x9b, 0x99, 0xc4, 0x3a, 0x10, 0xa8, 0x17, 0xfb} }
```

那么对应的 BINARY就应该为：

```bash
BINARY=7e5fd739-7a76-4391-9b99-c43a10a817fb
```

4.2.3 编译TA

TA的编译方法为输入make命令，以demo中TA为例：

```Bash
qbq@docker-ubuntu18:ta$ make
  CC      hello_world_ta.o
  CC      user_ta_header.o
  CPP     ta.lds
  GEN     dyn_list
  LD      8aaaf200-2450-11e4-abe2-0002a5d5c51b.elf
  OBJDUMP 8aaaf200-2450-11e4-abe2-0002a5d5c51b.dmp
  OBJCOPY 8aaaf200-2450-11e4-abe2-0002a5d5c51b.stripped.elf
  SIGN    8aaaf200-2450-11e4-abe2-0002a5d5c51b.ta
```

其中：

- 8aaaf200-2450-11e4-abe2-0002a5d5c51b.elf -- 当前TA的elf文件
- 8aaaf200-2450-11e4-abe2-0002a5d5c51b.dmp -- 当前TA的dump文件
- 8aaaf200-2450-11e4-abe2-0002a5d5c51b.stripped.elf -- strip之后的当前TA的elf文件
- 8aaaf200-2450-11e4-abe2-0002a5d5c51b.ta -- 目标TA文件

4.2.4 编译选项

4.2.4.1 默认编译选项

用户可以通过编译命令（make V=1）查看，以demo中的hello_world_ta 为例：

```Bash
-march=rv64imacxtheadc -mabi=lp64 -std=gnu99 -fdiagnostics-show-option -Wall -Wcast-align -Werror-implicit-function-declaration -Wextra -Wfloat-equal -Wformat-nonliteral -Wformat-security -Wformat=2 -Winit-self -Wmissing-declarations -Wmissing-format-attribute -Wmissing-include-dirs -Wmissing-noreturn -Wmissing-prototypes -Wnested-externs -Wpointer-arith -Wshadow -Wstrict-prototypes -Wswitch-default -Wwrite-strings -Wno-missing-field-initializers -Wno-format-zero-length -Wredundant-decls -Wold-style-definition -Wstrict-aliasing=2 -Wundef -march=rv64imacxtheadc -mabi=lp64 -mcmodel=medany -O0 -g3 -fpic -mstrict-align -march=rv64imacxtheadc -mabi=lp64 -MD -MF ./.hello_world_ta.o.d -MT hello_world_ta.o -nostdinc -isystem /home/qbq/work/src/xuantie-tee/toolchains/riscv64/bin/../lib/gcc/riscv64-unknown-linux-gnu/10.2.0/include -I./include -I./. -DRISCV64=1 -D__LP64__=1 -DMBEDTLS_SELF_TEST -DTRACE_LEVEL=4 -I. -I/home/qbq/work/src/riscv_yocto/thead-build/light-fm/tmp-glibc/work/riscv64-oe-linux/op-tee/0.1-r0/git/export/ta-rv64/include -DCFG_TEE_TA_LOG_LEVEL=4 -DCFG_RISCV64_ta_riscv64=1 -DCFG_SYSTEM_PTA=1 -DCFG_UNWIND=1 -DCFG_TA_BGET_TEST=1 -DCFG_TA_MBEDTLS=1 -DCFG_TA_MBEDTLS_SELF_TEST=1 -DCFG_TA_MBEDTLS_MPI
```

4.2.4.2 增加编译选项

如果用户想要增加编译选项，可以在sub.mk 文件中通过以下格式增加：

```Bash
cflags-"file_name"-y += "option"
```

其中：

- file_name ： 是指要增加编译选项的文件名
- option： 是指要增加的编译选项

如，我们对hello_world_ta.c增加编译选项“-Wno-strict-prototypes”，可以在sub.mk中增加：

```Bash
cflags-hello_world_ta.c-y += -Wno-strict-prototypes
```

4.3 TA/CA部署

4.3.1 linux部署

编译好的TA/CA需要拷贝到linux的文件系统中，可以通过静态编译的或者动态拷贝的方式实现。

- 静态编译
  - 将生成的TA 文件拷贝到rootfs目录/lib/optee_armtz/
  - 将生成的CA文件拷贝到rootfs目录/usr/bin
- 动态拷贝（建议 ）

  在系统起来以后可以通过SCP和adb的方式将
  - TA文件拷贝到rootfs目录/lib/optee_armtz/
  - CA文件拷贝到rootfs目录/usr/bin

4.3.2 android部署

- step1. 板子上电启动到安卓kernel，pc上运行以下命令打开安卓文件系统写权限

```Plain Text
adb root
adb remount
```

- step2. 运行以下命令将需要部署的ta和ca部署到板子上(以hello_world举例)

```
adb push xxxxxxxxxxxx.ta /vendor/lib/optee_armtz
adb push hello_wolrd /vendor/bin
```

5. 验证方法

5.1 linux验证

在开发板上运行安全子系统，当系统进入kernel之后：

- 将上述生成的ta（8aaaf200-2450-11e4-abe2-0002a5d5c51b.ta）拷贝到/lib/optee_armtz/
- 将ca 拷贝到用户目录~/

运行后得到结果如下，说明ta运行正常（注意：如果当前没有运行过tee-supplicant 需要先执行tee-supplicant &）

```Bash
root@light-fm-linux-v0:~# ./optee_example_hello_world
D/TC:? 0 tee_ta_init_pseudo_ta_session:299 Lookup pseudo TA 8aaaf200-2450-11e4-abe2-0002a5d5c51b
D/TC:? 0 ldelf_load_ldelf:95 ldelf load address 0x40002000
D/LD:  ldelf:134 Loading TS 8aaaf200-2450-11e4-abe2-0002a5d5c51b
D/TC:? 0 ldelf_syscall_open_bin:147 Lookup user TA ELF 8aaaf200-2450-11e4-abe2-0002a5d5c51b (Secure Storage TA)
D/TC:? 0 ldelf_syscall_open_bin:151 res=0xffff0008
D/TC:? 0 ldelf_syscall_open_bin:147 Lookup user TA ELF 8aaaf200-2450-11e4-abe2-0002a5d5c51b (REE)
D/TC:? 0 ldelf_syscall_open_bin:151 res=0
D/LD:  ldelf:168 ELF (8aaaf200-2450-11e4-abe2-0002a5d5c51b) at 0x4007d000
D/TA:  TA_CreateEntryPoint:39 has been called
D/TA:  TA_OpenSessionEntryPoint:68 has been called
I/TA: Hello World!
D/TA:  inc_value:105 has been called
I/TA: Got value: 42 from NW
I/TA: Increase value to: 43
D/TC:? 0 tee_ta_close_session:516 csess 0xff0967a0 id 1
D/TC:? 0 tee_ta_close_session:535 Destroy session
I/TA: Goodbye!
D/TA:  TA_DestroyEntryPoint:50 has been called
D/TC:? 0 destroy_context:312 Destroy TA ctx (0xff096740)
Invoking TA to increment 42
TA incremented value to 43

```

5.2 android验证

- step1. 确保按照章节"**4.3.2 android部署**"已完成部署，然后pc 上运行 adb root开启root权限(tee-supplicant需要)

```Plain Text
adb root
```

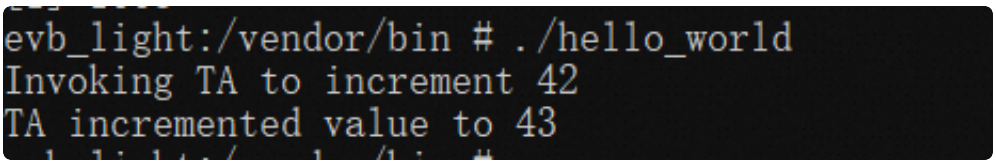
- step2. adb shell连接板子开启tee服务

```
adb shell
tee-supplicant &
```

- step3. 测试验证ca(以hello_world举例)，pc上adb shell连接执行命令

```
adb shell
cd /vendor/bin
./hello_world
```

如果运行成功，可以看到以下输出



6. 参考文档

- OPTEE用户手册：https://optee.readthedocs.io/en/latest/
- GP client API手册：https://globalplatform.org/specs-library/tee-client-api-specification/
- GP internal API手册：https://globalplatform.org/specs-library/tee-internal-core-api-specification/