TinyTEE 使用文档

(M2351 平台)

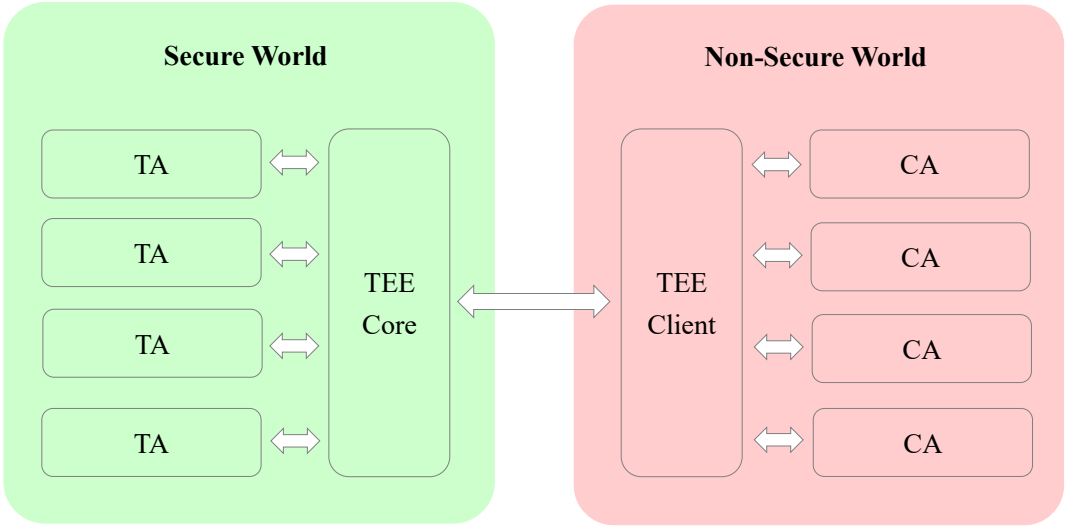
版本	编写/修订说明	修订人	修订日期	备注
1.0.0	初始版本	刘晨	2019-08-19	无

1. 简介

青莲云 TinyTEE 是基于 ARMv8-M 指令集及 TrustZone 技术搭建的可信执行环境 (Trusted Execution Environment)，充分利用硬件手段对 RAM、Flash 进行了隔离，以保障设备运行时的安全，适用于资源受限的 IoT 设备。

TinyTEE 可由用户自行开发 TA (Trusted Application)，也可提供定制化 TA 服务，利用通用 TA 和青莲云嵌入式通信 SDK，将本地安全与链路安全、云端安全相结合，可提供一个软、硬件结合，覆盖全链路的、端到端整体物联网安全解决方案。

TinyTEE 的总体框架如下：

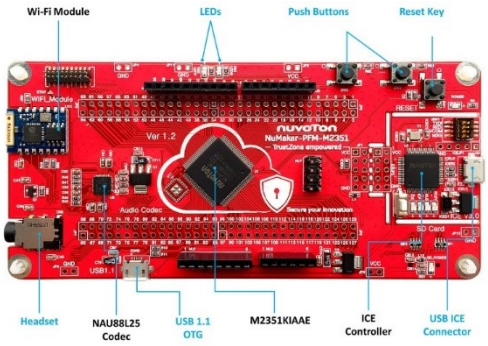


本文档介绍了 M2351 平台上 TinyTEE 的使用，包括软硬件环境的准备、Demo 程序说明与如何添加 TA 与 CA。

2. 准备工作

2.1. 硬件

TinyTEE M2351 平台 Demo 程序对硬件无特殊要求，程序仅通过 UART0 输出运行状态信息。推荐使用新唐的 NuMaker-PFM-M2351 开发板（相关信息可从新唐科技官网获取）。



2.2. 软件开发环境

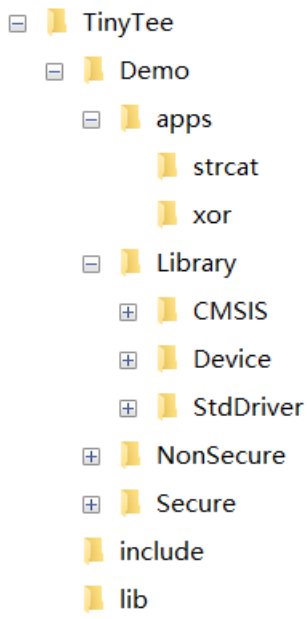
软件开发环境当前仅支持 Keil MDK。软件的下载、安装和注册过程请参考 M2351 系列 BSP 包中的 NuMaker-PFM-M2351 Board Quick Start Guide.pdf 文档。

BSP 包可以从新唐官网的“单片机 > Arm Cortex™-M23 单片机 > M2351 系列 > 软件”页面下载（编写本文档时为 M2351_BSP_v3.00.003）。

3. Demo 程序内容

3.1. 目录结构

TinyTEE (M2351 平台版本) 的目录结构如下：



lib 和 include 目录下为 SDK 的静态库文件和对应的头文件，Demo 下为示例工程。

lib 目录下的静态库文件如下表所示：

库文件	描述
tee_m2351_non_secure.lib	TEE (NonSecure)
tee_m2351_secure.lib	TEE (Secure)

include 目录下的头文件如下表所示：

头文件	库文件	描述
tee_client_api.h	tee_m2351_non_secure.lib	TEE Client API 头文件
tee_driver.h	tee_m2351_secure.lib	TEE Driver 头文件
tee_internal_api.h		TEE Internal Core API 头文件

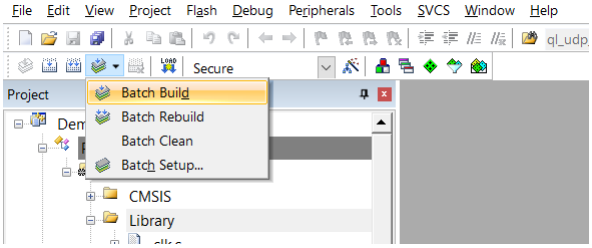
demo_m2351 目录下的主要目录和文件如下表所示：

文件或目录	描述
apps	示例 TA 和 CA，包含 xor 和 strcat 两个示例
Library	复制自 M2351 BSP 的 Demo 项目所需的 Library 文件
NonSecure	NonSecure 项目
Secure	Secure 项目
Demo.uvmpw	Demo 的 Multi-Project Workspace 文件

4. Demo 程序使用

4.1. 编译

在 Keil MDK 中打开 Demo\Demo.uvmpw 多项目工作空间，点击 Batch Build 按钮即可进行 Secure 和 NonSecure 项目的编译。



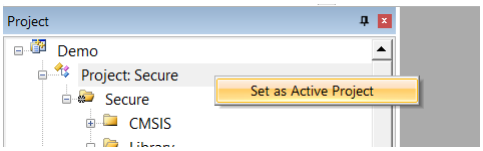
也可单独进行编译，但注意 NonSecure 项目要使用 Secure 项目编译产生的 nsclib_Secure.o 文件。若 Non-Secure callable 函数有变化，必须要先编译 Secure 项目。

4.2. 下载

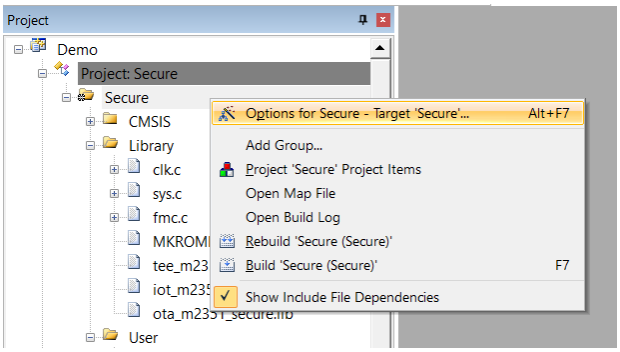
下载程序时，Secure 和 NonSecure 区域要单独下载。同时，如果 M2351 芯片的 Secure Setting 没有被设置过，还需要先设置 Secure Setting 中的 NonSecure 区 Base Address，否则默认值可能会与项目的 Secure, NonSecure 区域划分不同，导致引导到 NonSecure 区时失败。

4.2.1. 设置 Secure Setting

首先，右键点击“Project: Secure”，选择“Set as Active Project”，将 Secure 项目设为活动项目。

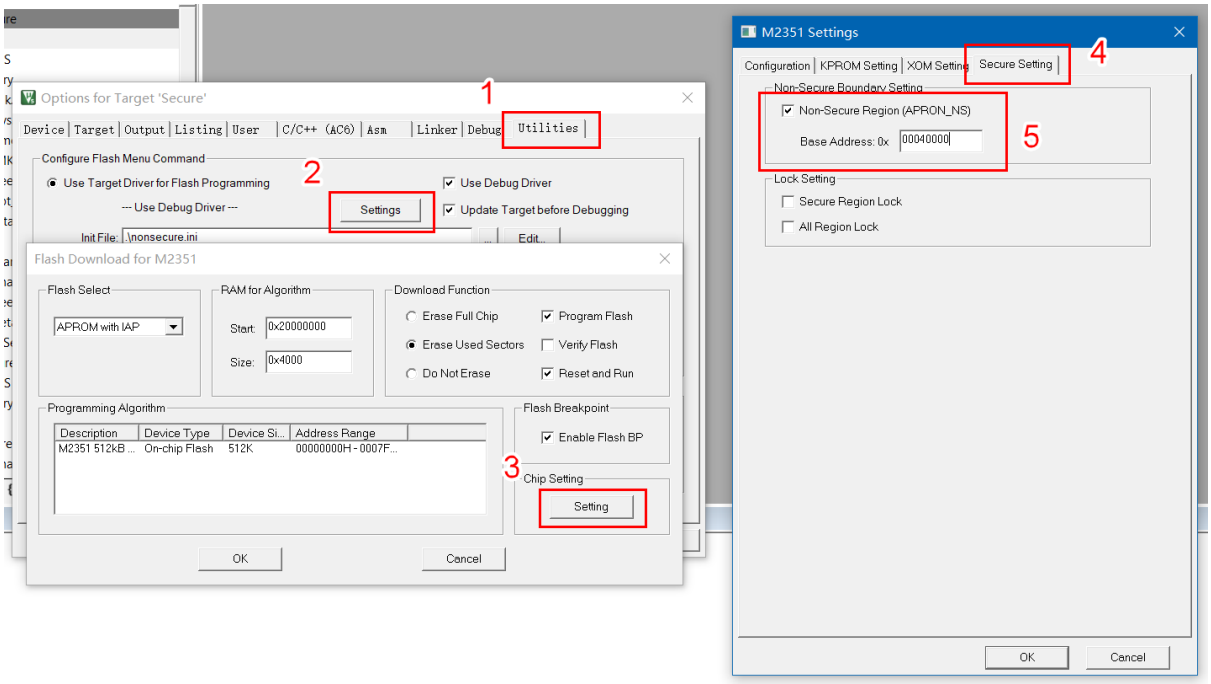


再右键点击“Project: Secure”下的“Secure”，选择“Options for Secure - Target 'Secure'...”。



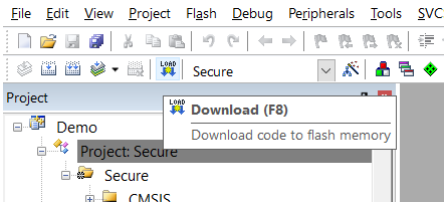
然后点击 Utilities 选项卡下的 Settings 按钮，进入 Flash 下载配置页面，点击 Chip Settings 内的 Settings 按钮，在弹出的 M2351 Settings 页面中选择 Secure Setting 选项卡，并勾选“Non-Secure Region (APROM_NS)”，将 Base Address 设为 0x00040000，点击“OK”。

此时可能会弹出“Are you sure you want to modify the chip settings?”和“It is necessary to erase whole chip before modifying Non-Secure Base Address. Continue?”的提示，均点击确定。此时 M2351 芯片的 Flash 被整体擦除，并且 Secure Setting 的值为刚才所配置的值。



4.2.2. 下载 Secure 区程序

将 Secure 项目设为活动项目后，点击工具栏的 Download 按钮下载 Secure 区程序。



4.2.3. 下载 NonSecure 区程序

同理，将 NonSecure 项目设为活动项目后，点击工具栏的 Download 按钮下载 NonSecure 区程序。

4.3. 运行

根据项目的默认配置，下载程序后会自动复位并运行。若未运行，则可手动按开发板上的 RESET 键进行复位。

设备上电复位后，UART0 会输出如下信息：

```
In Secure main()
Execute non-secure code ...
Entered Non-Secure main()

================ xor test ================

a = 1, b = 1, c = a ^ b = 0
a = 0, b = 1, c = a ^ b = 1

================ strcat test ================

Test 1: tmpref + tmpref -> tmpref
a = test_tmpref, b = ##ok_tmpref
out = test_tmpref##ok_tmpref (size = 22)
Test 2: whole + whole -> tmpref
a = test_memref, b = ##ok_memref
out = test_memref##ok_memref (size = 22)

Test 3: partial + partial -> tmpref
a = t_memref, b = ##ok
out = t_memref##ok (size = 12)
```

4.4. 添加新的 TA 和 CA

本节介绍如何参考 Demo\apps\xor 异或值计算示例程序，添加新的加法计算 TA 和 CA。

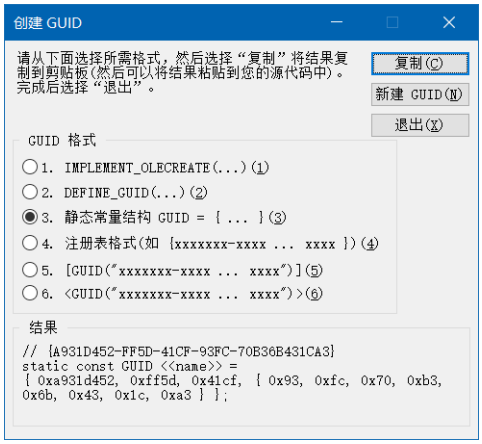
4.4.1. 编写头文件

TA 和 CA 头文件的编写可参考 Demo\apps\xor\xor.h，这里编写 Demo\apps\add\add.h 文件内容如下：

```
#ifndef __TA_ADD_H__
#define __TA_ADD_H__

// {A931D452-FF5D-41CF-93FC-70B36B431CA3}
#define ADD_TA_UUID            { 0xa931d452, 0xff5d, 0x41cf, { 0x93, 0xfc, 0x70, 0xb3,
0x6b, 0x43, 0x1c, 0xa3 } }
```

```
#define ADD_TA_CMD_DEFAULT      0x12340001

#if defined(__ARM_FEATURE_CMSE) && (__ARM_FEATURE_CMSE == 3)
    #include "tee_driver.h"
    extern TA_Reference g_ta_reference_add;
#else
    int add_test(void);
#endif

#endif /* __TA_ADD_H__ */
```

其中，TA 的 UUID 值可以通过 Visual Studio 的“创建 GUID”工具或其他类似工具生成。



4.4.2. 编写 TA

TA 的编写可参考 Demo\apps\xor\xor_ta.c，这里编写 Demo\apps\add\add_ta.c 文件。

首先，定义一个 UUID 的全局变量：

```
static TEE_UUID _add_ta_uuid = ADD_TA_UUID;
```

然后，实现如下 5 个入口函数：

```
TEE_Result _TA_CreateEntryPoint(void);
void _TA_DestroyEntryPoint(void);
TEE_Result _TA_OpenSessionEntryPoint(...);
void _TA_CloseSessionEntryPoint(...);
TEE_Result _TA_InvokeCommandEntryPoint(...);
```

入口函数通过在 TA_Reference 结构体中设置函数指针被调用，对其函数命名没有要求。入口函数的具体定义见 tee_internal_api.h 中的如下函数指针类型定义：

```
typedef TEE_Result (*TA_CreateEntryPoint_FuncPtr)(void);
typedef void (*TA_DestroyEntryPoint_FuncPtr)(void);
typedef TEE_Result (*TA_OpenSessionEntryPoint_FuncPtr)(uint32_t paramTypes, TEE_Param
params[4], void **sessionContext);
typedef void (*TA_CloseSessionEntryPoint_FuncPtr)(void *sessionContext);
typedef TEE_Result (*TA_InvokeCommandEntryPoint_FuncPtr)(void *sessionContext, uint32_t
commandID, uint32_t paramTypes, TEE_Param params[4]);
```

接着，定义一个 TA_Reference 结构体，并设置值为刚才定义的 UUID 和入口函数。

```
TA_Reference g_ta_reference_add = {
    .appId = &_add_ta_uuid,
```

```
    .entryPointTable = {
        .createEntryPoint = _TA_CreateEntryPoint,
        .destroyEntryPoint = _TA_DestroyEntryPoint,
        .openSessionEntryPoint = _TA_OpenSessionEntryPoint,
        .closeSessionEntryPoint = _TA_CloseSessionEntryPoint,
        .invokeCommandEntryPoint = _TA_InvokeCommandEntryPoint
    }
};
```

到此，TA 的源文件即编写完成。最后，在 Secure 项目的 main()函数中添加对该 TA 的注册。

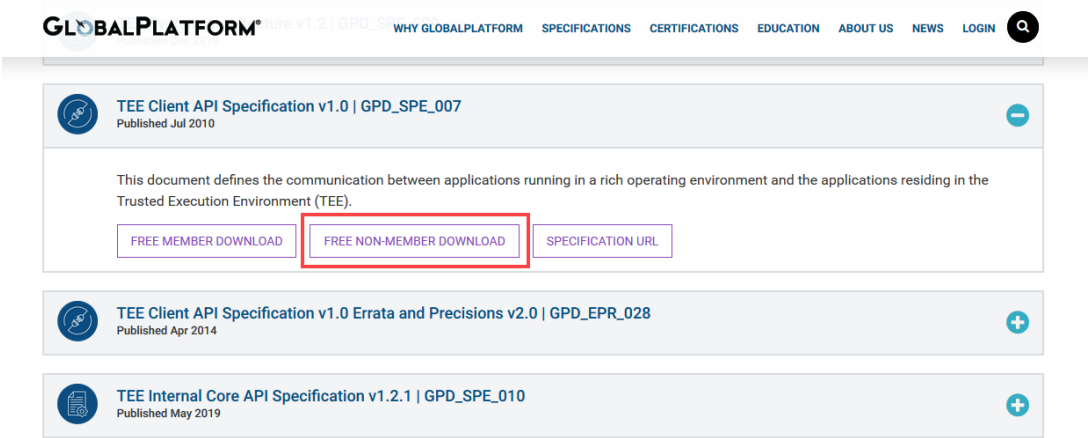
```
    // 注册 TA
    teeDriverTaRegister(&g_ta_reference_xor);
    teeDriverTaRegister(&g_ta_reference_strcat);
    teeDriverTaRegister(&g_ta_reference_add);
```

4.4.3. 编写 CA

CA 的编写请参考 Demo\apps\xor\xor_ca.c 文件，这里不再赘述。

5. TEE 参考资料

TEE 规范文档可从 GlobalPlatform 官方网站获取，下载地址为 https://globalplatform.org/specs-library/?filter-committee=tee。



点击“FREE NON-MEMBER DOWNLOAD”后按要求填入信息即可下载。

主要参考的文档有以下几个：

- TEE Client API Specification v1.0 | GPD_SPE_007
- TEE Client API Specification v1.0 Errata and Precisions v2.0 | GPD_EPR_028
- TEE Internal Core API Specification v1.2.1 | GPD_SPE_010