SDK应用_Secure_Boot 模块配置及应用

版本

YT Config Tool Version: 2.4.0

YTM32BxSDK version: 1.2.0

Secure Boot Version: 1.0.0

1. 前言

芯片上电或复位后，默认从 Secure Boot 区域启动。Secure Boot 旨在保护用户的应用程序与数据，当用户程序与数据被篡改，或者受到物理损坏时，Secure Boot 可以拒绝跳转至程序区，从而实现保护应用程序与数据。

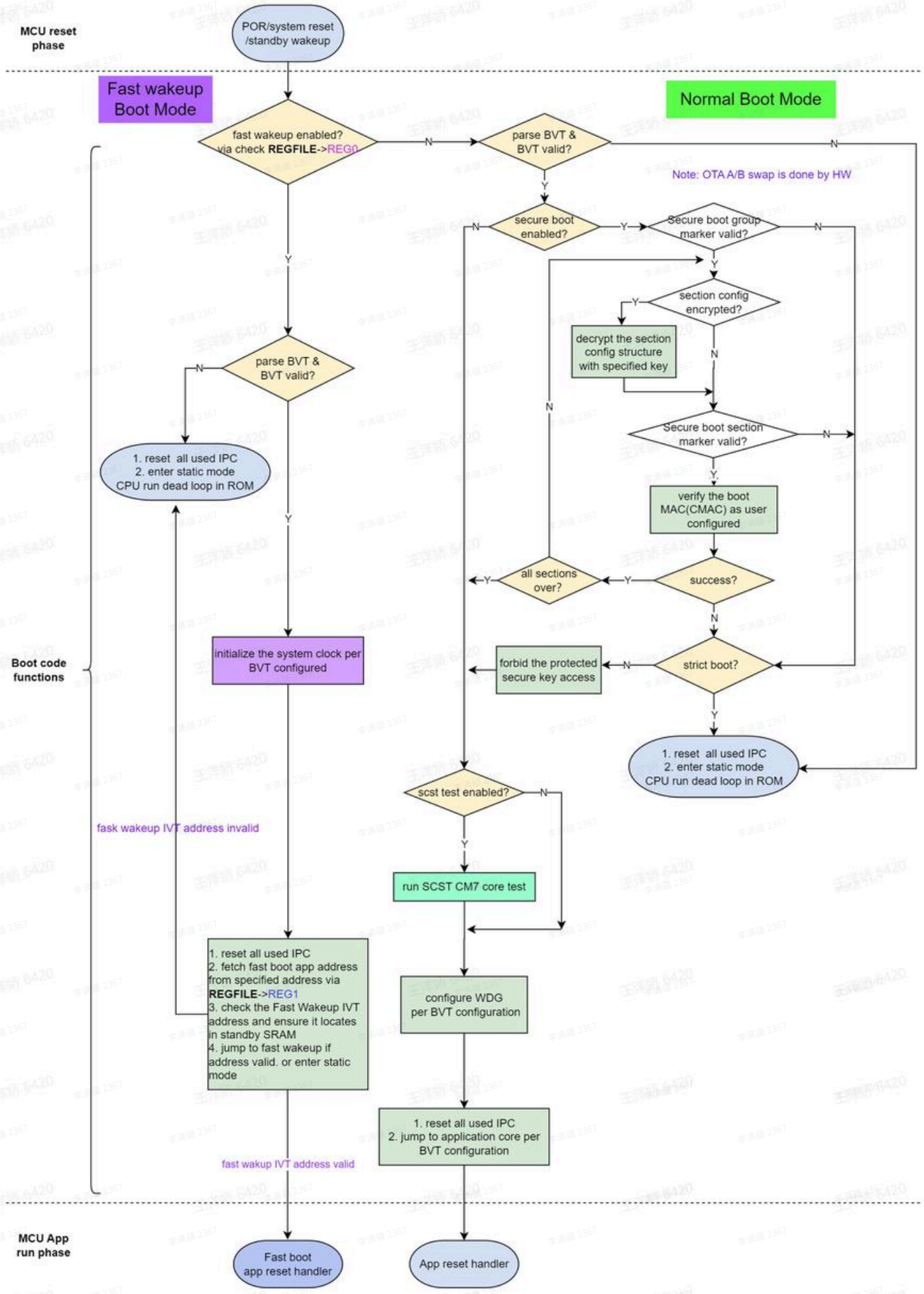
另外云途的 Secure Boot 可对时钟模块与看门狗模块进行预配置。时钟配置可以提速安全启动与用户的启动程序，看门狗配置可以防止在启动过程中出现死机。

截至当前版本，共有四款芯片支持 Secure Boot 功能配置，其中 YTM32B1ME05, YTM32B1MD14 为软件 Secure Boot，即在用户程序区分出一块区域用于放置 Secure Boot 固件；YTM32B1MC03, YTM32B1HA01 为硬件 Secure Boot，用户无需下载 Secure Boot 固件。

2. 原理

云途芯片的安全启动流程大致类似，各芯片细节或许有差异，详情可看对应芯片的 Reference Manual。本文主要介绍 **YTM32B1HA01** 的启动流程与配置。

2.1 安全启动流程



The ROM boot work flowchart

上图节选自 YTM32B1HA01 Reference Manual 4.2.2 The ROM Boot Work Flowchart 章节。

YTM32B1M32B1HA01 的安全启动有两种方式，**Normal Boot** 与 **Fast Wakeup Boot**。Normal Boot 为正常的，标准的安全启动流程。Fast Wakeup Boot 为快速启动流程，主要是针对 Power Down 功耗模式下唤醒后，立即跳转至客户应用程序。

2.1.1 Normal Boot

1. 读取 *REGFILE->REG0* 内容，确认 TAG 不一致，则进入 Normal Boot 模式
2. 解析 **BVT**(Boot Vector Table)，确认 BVT 信息是否有效（BVT 详细内容在下文会介绍），若 BVT 非法则程序停留在 ROM 内，*拒绝跳转至App*，否则进入下一步
3. 确认 Secure Boot 是否使能，若未使能，则不对用户程序区与数据区作验签，直接跳转至步骤9，否则进入下一步
4. 确认 Secure Boot Group 是否有效（Secure Boot Group 详细内容在下文会介绍），若非法则跳转至步骤8，否则进入下一步
5. 确认 Secure Boot Section 配置信息是否被加密（Secure Boot Section 详细内容在下文会介绍），若扇区配置信息被加密，则先对扇区信息进行解密（AES-ECB），随后进入下一步
6. 确认 Secure Boot Section 是否有效，若非法则跳转至步骤8，否则进入下一步
7. 根据扇区配置信息里的内容，对该扇区所保护的区域进行验签（AES-CMAC），若验签失败，则跳转至步骤8；若验签成功，则确认是否所有扇区都验签完成，若所有扇区均验签成功，则跳转至步骤9
8. 确认是否严格执行安全启动，若严格执行安全启动，则程序停留在 ROM 内，*拒绝跳转至App*；若非严格执行安全启动，则禁止用户程序加载 HCU 硬件密钥（HCU_NVR），随后跳转至步骤9
9. 确认是否需要进行内核自测试（Core Test），若需要则进行内核自测试，否则跳转至下一步
10. 根据 BVT 信息配置 WDG
11. 根据 BVT 信息配置获取应用程序的跳转地址
12. 跳转至用户的应用程序

2.1.2 Fast Wakeup Boot

1. 读取 *REGFILE->REG0* 内容，确认 TAG 一致，则进入 Fast Wakeup Boot 模式
2. 解析 **BVT**(Boot Vector Table)，确认 BVT 信息是否有效（BVT 详细内容在下文会介绍），若 BVT 非法则程序停留在 ROM 内，*拒绝跳转至App*，否则进入下一步
3. 根据 BVT 信息配置系统时钟
4. 读取 *REGFILE->REG1* 内容，获取快速启动的向量表
5. 确认该向量表在 **Retention RAM** 内，若不在 **Retention RAM** 则程序停留在 ROM 内，*拒绝跳转至App*。（Retention RAM 与 普通的 OCRAM 不同，Retention RAM 在 PowerDown 模式下，仍能保持当前内容；普通的 OCRAM 在 PowerDown 模式唤醒后，会变成随机值。）

6. 跳转至用户的快速启动应用程序

2.2 BVT 介绍

BVT（Boot Vector Table）是 Secure Boot 的配置信息区域，地址为 0x0200_0000 与 0x0210_0000。Secure Boot 程序会检测这两个位置是否有合法的 BVT 内容。

偏移地址	字节数	内容	描述
00h	4	Image Vector Table Marker	这是一个特殊的数字，用于确认 BVT 是否有效，这个值为 0xA55A_A55A
04h	4	Boot Configuration Word	配置字，允许用户选择可以引导设备的各种配置。细节将在下一节中提到。
08h	4	Secure Boot Group Config Address	内存安全启动组配置的起始地址，一个有效的配置组是由一个固定的标记标识，由若干安全引导段配置组成信息
0Ch	4	Reserved	-
10h	4	CM7_0_main application Start Address	用户程序的起始地址，必须遵循内核 VTOR 的对齐限制。
14h	20	Reserved	-
28h	4	APP_WDG_TIMEOUT	用于设置应用程序看门狗超时时间，看门狗时钟选择 SIRC/4 (3MHz)，若该值设为 3000，则看门狗超时时间为1ms。
2Ch	20	Reserved	-
40h-0x3FF	TBD	reserved for secure boot group and section config as well as CMAC result storge	该扇区的剩余地址空间(2KB)可用于存储安全引导配置组信息/结构和安全引导段配置结构以及每个安全引导段的CMAC结果(需要16字节对齐)。

2.3 BCW 介绍

BCW（Boot Config Word）是 BVT 内的一个 Word，其中每个比特都有其独特意义。

比特域	描述	
31-14	Reserved for future extension	
13-10	CPDIVS[3:0]	Core Platform Clock 分频配置: 0000b - Divide by 1 0001b - Divide by 2 ... 1111b - Divide by 16 该配置可用于提升 Secure Boot 运算速度，以及应用程序启动时间(Startup). Tips: 芯片默认使用48MHz系统时钟，该配置提升系统时钟至96MHz
9	ROM_BOOT_SCST_EN	控制是否需要完成内核自测试（Core Test） 0 - Disable 1 - Enable 注意：如果希望在应用程序中也调用 ROM 中的 Core Test 库，则不能破坏 OCRAM 的最后 4KB 区域 (0x0x2005_F000 to 0x2005_FFFF)
8	APP_WDG_EN	控制应用程序的看门狗是否使能。 0 - Disable 1 - Enable 当该比特置起时，Boot中可以在进入应用程序前初始化看门狗。应用程序中应当及时喂狗来防止看门狗复位。具体的超时时间可在 *BVT -> APP_WDG_TIMEOUT(0x28h)* 中配置
7	BOOT_SEQ_STRICT	控制是否使能严格安全启动流程 0 - Disable 1 - Enable
6	BOOT_SEQ	安全启动模式配置: 0 - 禁止串行安全启动模式 1 - 使能串行安全启动模式
5-1	Reserved	-
0	CM7_0_M_EN	是否使能跳转至应用程序 0 - Disable 1 - Enable 注意：为保证能正常执行用户的应用程序，该位必须置1

2.4 Secure Boot Group

Secure Boot 配置通过 Secure Boot 分组配置与Secure Boot 扇区配置来实现，最多可支持8 块 Secure Boot 扇区配置。

指向Secure Boot 分组配置信息的地址存储于 *BVT->Secure_Boot_Group_Config_Address(0x08h)* 中，细节如下所示：

1. 分组信息标志: 0xACBEEFDD (4B)
2. Secure Boot 扇区个数(SBSN)，最多支持8组 (1B)
3. 是否加密扇区配置信息 (1B)
4. AES Key 的类别 (1B)
5. 用于解密扇区配置信息的密钥索引 (1B)
6. 指向Secure Boot 扇区配置信息的地址 (4B*SBSN)

地址偏移	字节数	内容	描述
0x00	4	Config Group Makrer	固定为 0xACBEEFDD，用来判断 Secure Boot 分组信息是否合法。
0x04	1	Secure Boot Section Number (SBSN)	Secure Boot 扇区个数，最多可支持8组扇区配置
0x05	1	Encrypt Flag	指示是否需要解密扇区配置信息 注意: 加密扇区配置信息需要开发人员事先完成
0x06	1	AES Key Type/Size	解密扇区配置信息的密钥种类，包括 AES-128/192/256
0x07	1	Key Slot	用于解密扇区配置信息的密钥索引
0x08	4*SBSN	Secure Boot Section Config Pointer	指向 Secure Boot 扇区配置信息的地址(32-bit)数组，有几个扇区配置信息，就有几个地址

2.5 Secure Boot Section

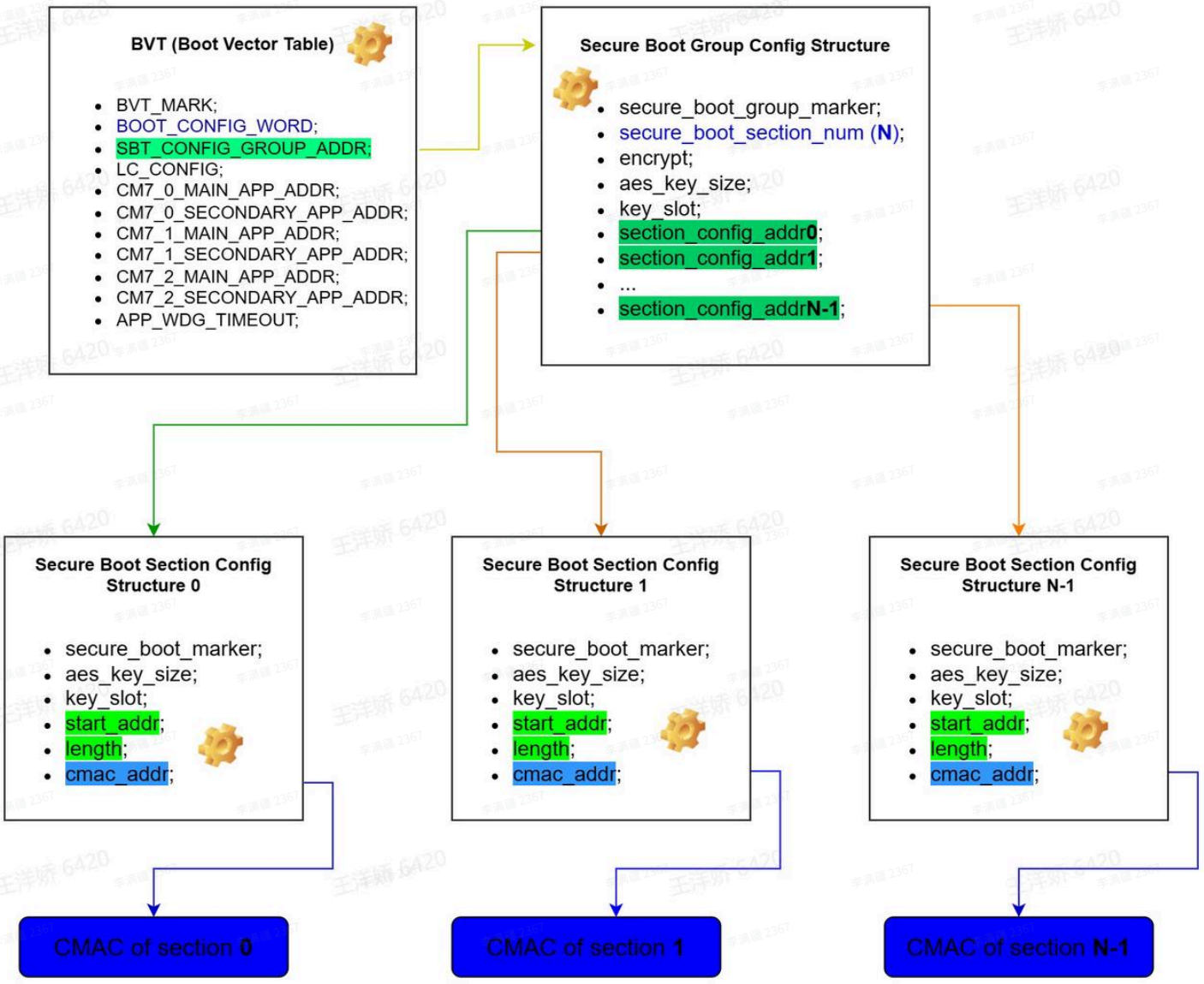
Secure Boot 的扇区配置信息需要16 字节对齐，具体信息如下所示：

1. 扇区信息标志：0x5AA5 (2B)
2. AES-128/192/256 密钥类型选择 (1B)
3. 用于校验CMAC 的密钥索引 (1B)
4. 需要进行验签的Flash 起始地址 (4B)
5. 需要进行验签的数据长度 (4B)
6. 验签结果的存储地址 (4B)

地址偏移	字节数	内容	描述
0x00	2	Config Section Marker	固定为 0x5AA5，用来判断 Secure Boot 扇区信息是否合理
0x02	1	AES Key Type/Size	CMAC验签的密钥种类，包括AES-128/192/256
0x03	1	Key Slot	用于校验 CMAC 的密钥索引
0x04	4	Section Start Address	需要进行验签的 Flash 起始地址
0x08	4	Length	需要进行验签的数据长度
0x0C	4	CMAC address	验签结果的存储地址

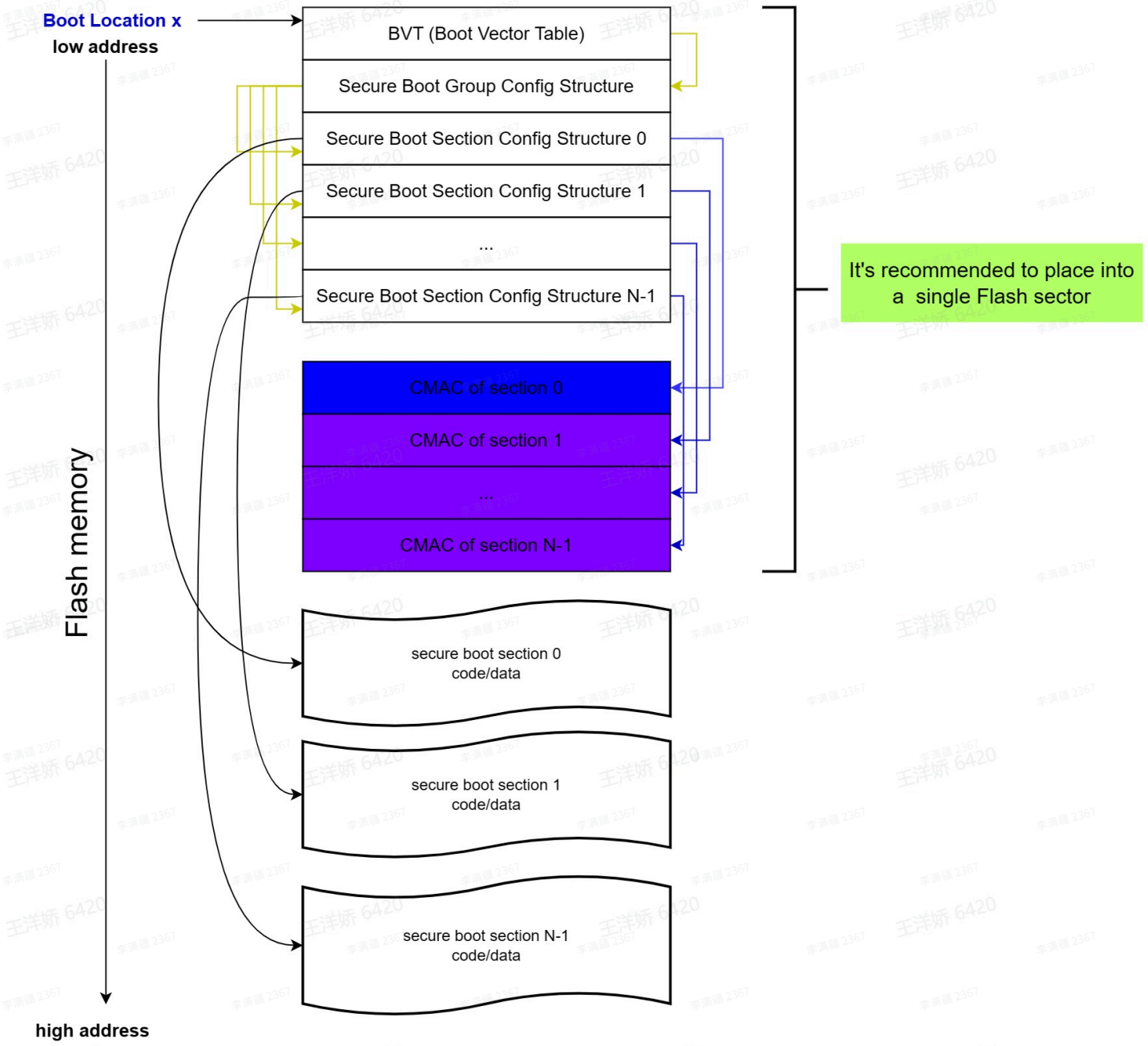
2.6 各配置信息关系图

BVT 中包含 Secure Boot Group 的物理地址，Secure Boot Group 内有 Secure Boot Section 的物理地址，Secure Boot Section 有 CMAC 验签结果的物理地址，这些配置信息如下图所示。



secure boot group/section configuration relationship with BVT

此外，下图还展示了与用户应用程序的关系图，供参考理解。



User Application Memory Map

2.7 HCU 密钥

在 YTM32B1HA01 的 Secure Boot 中，主要用到了 **AES-ECB** 用于解密 Secure Boot Section 配置信息，以及 **AES-CMAC** 用于对用户程序数据进行验签。其中使用的密钥均存储在 **HCU_NVR** 中，该区域无法被任何用户读取，只能通过 HCU 模块进行加载密钥，且用户无法在 HCU 模块中读取该密钥。

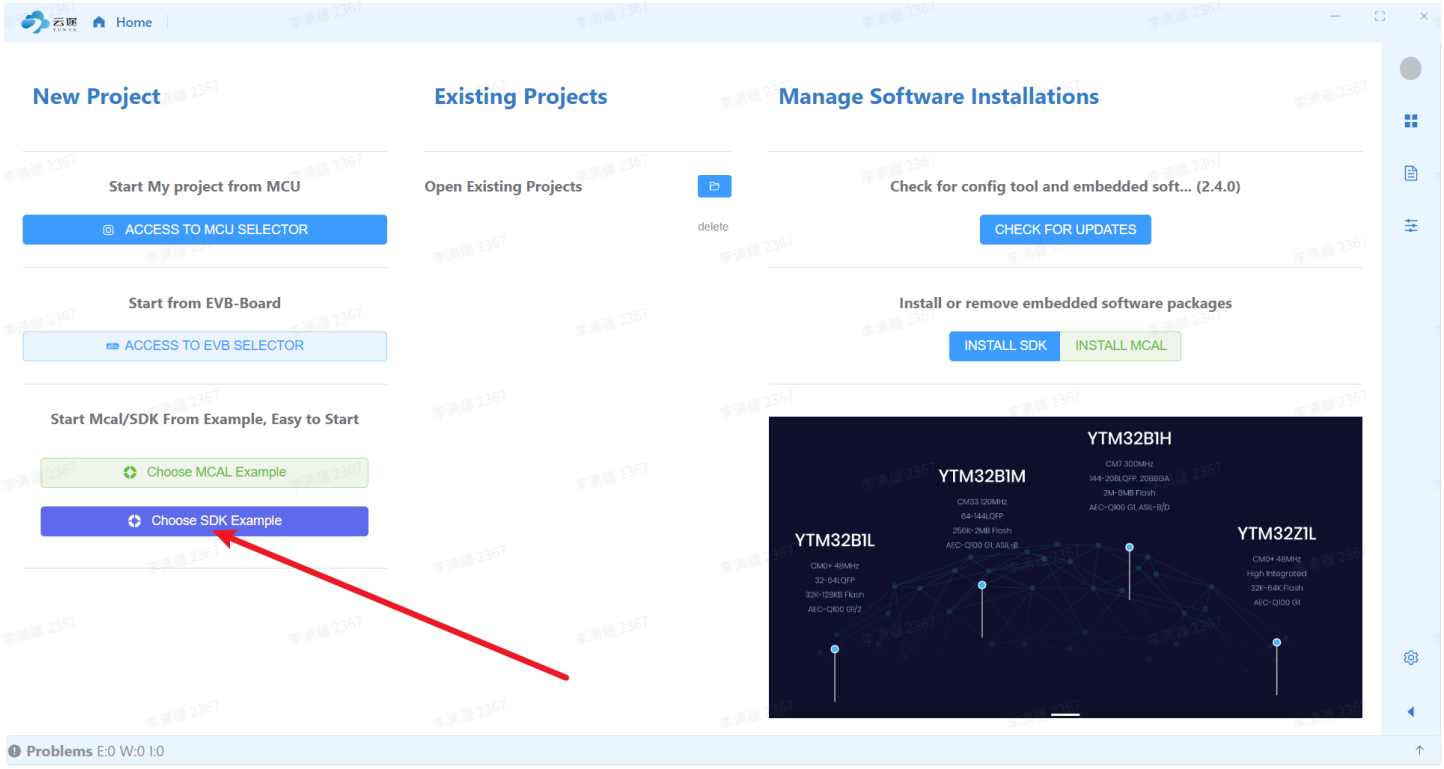
3. 应用

要使用 Secure Boot 功能，通常情况下需要禁止 Secure Boot 的 ByPass 模式，并且将 Secure Boot 过程中使用的密钥提前编程至 HCU_NVR 中。所以 YT_Config_Tool 提供了两个 Demo，**Secure_Boot_Prepare** 与 **Secure_Boot_Demo**。

仍以 YTM32B1HA01 为例进行 Demo 展示。

3.1 Secure Boot Prepare

用户可以直接点开 SDK Example 选择 YTM32B1HA01 的 Secure Boot Prepare demo

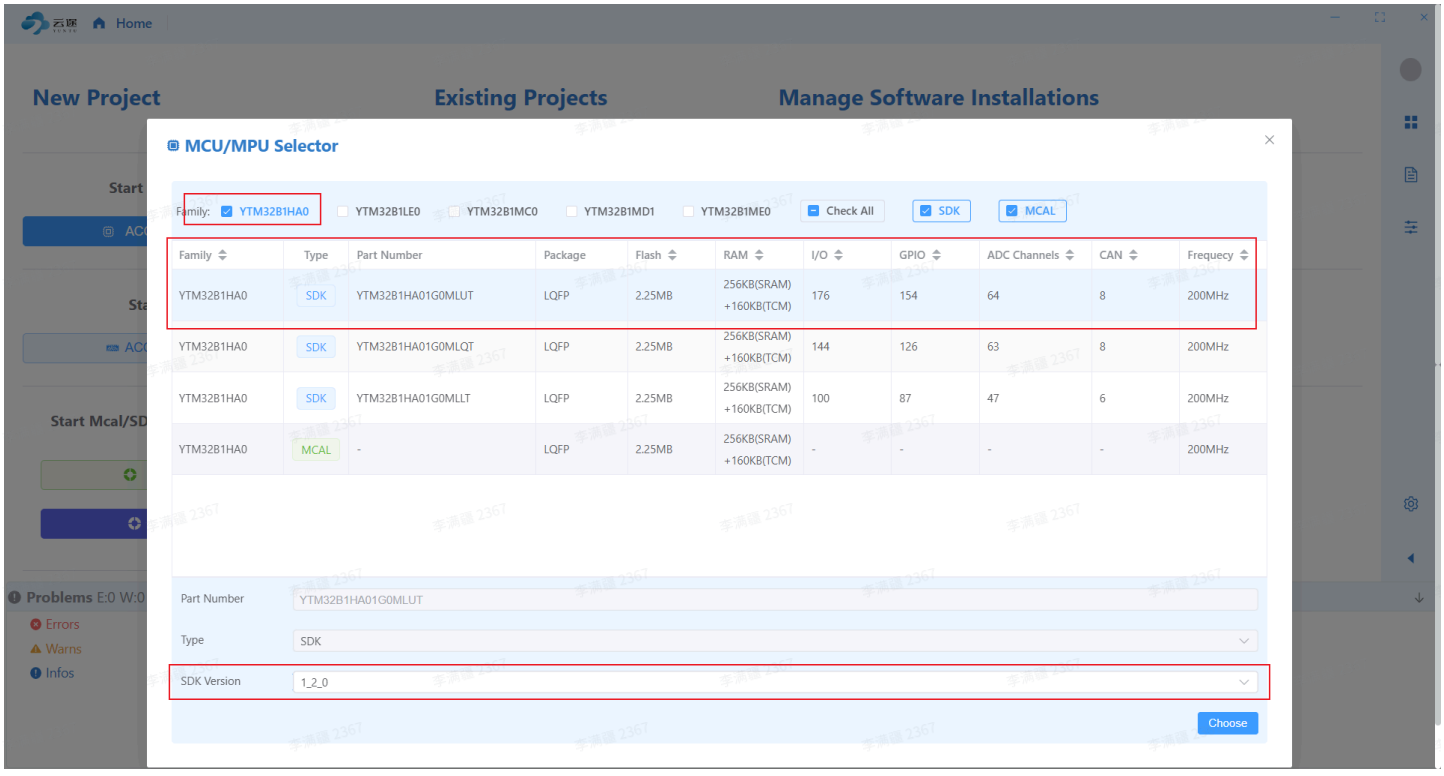


或者跟随下文一步步重新创建工程。

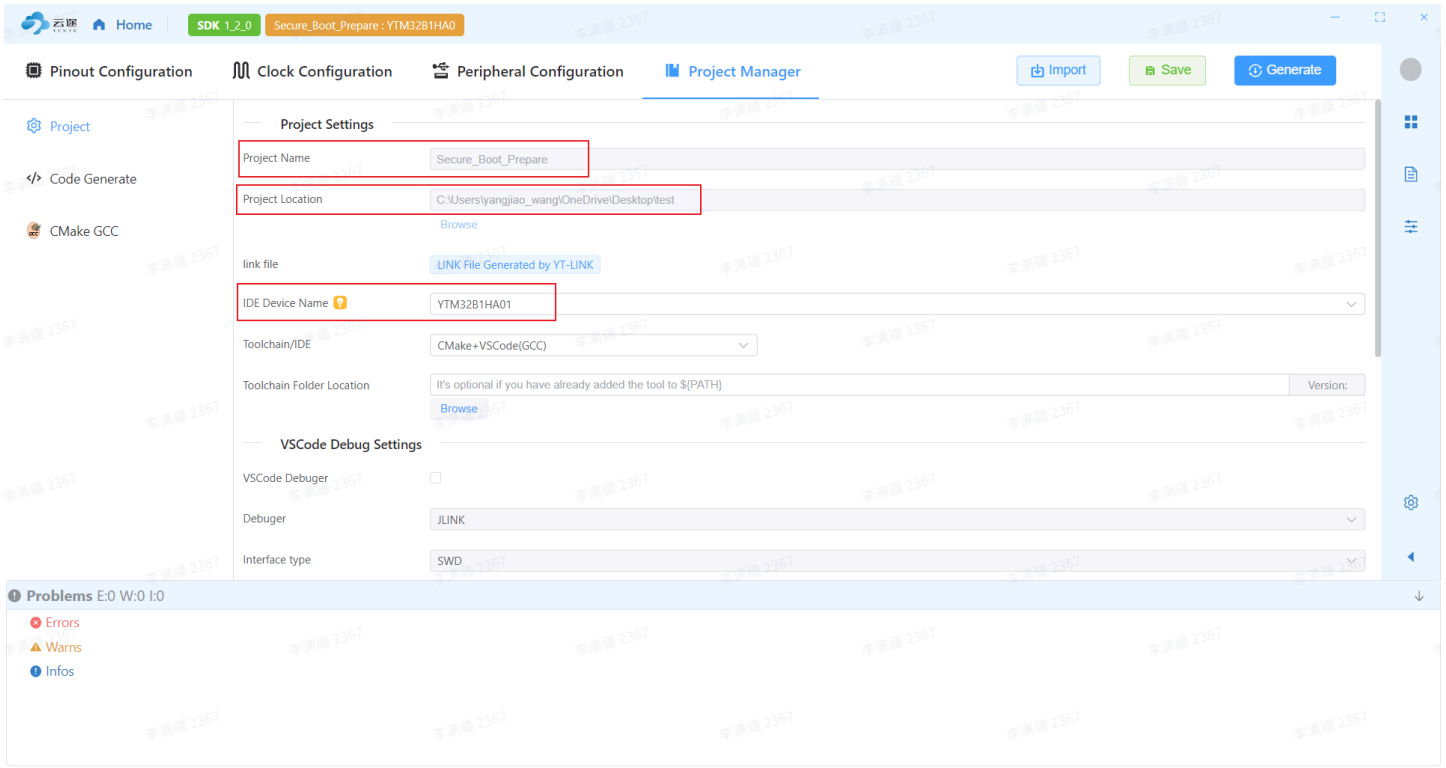
3.1.1 基础配置

3.1.1.1 创建工程

选择 YTM32B1HA01，选择当前芯片的具体型号，选择 SDK 版本号（≥1.2.0）

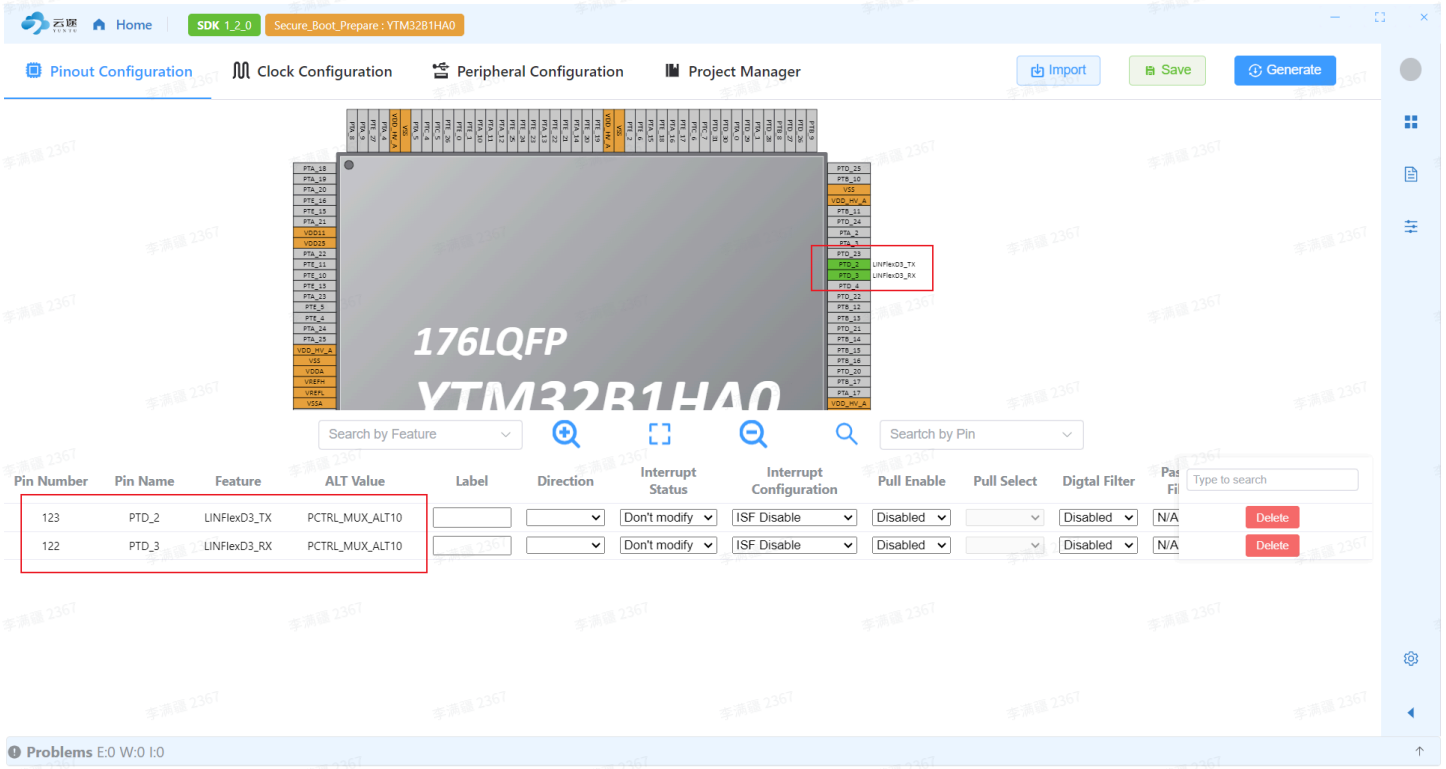


设置项目名字，工程路径，以及 IDE Device Name



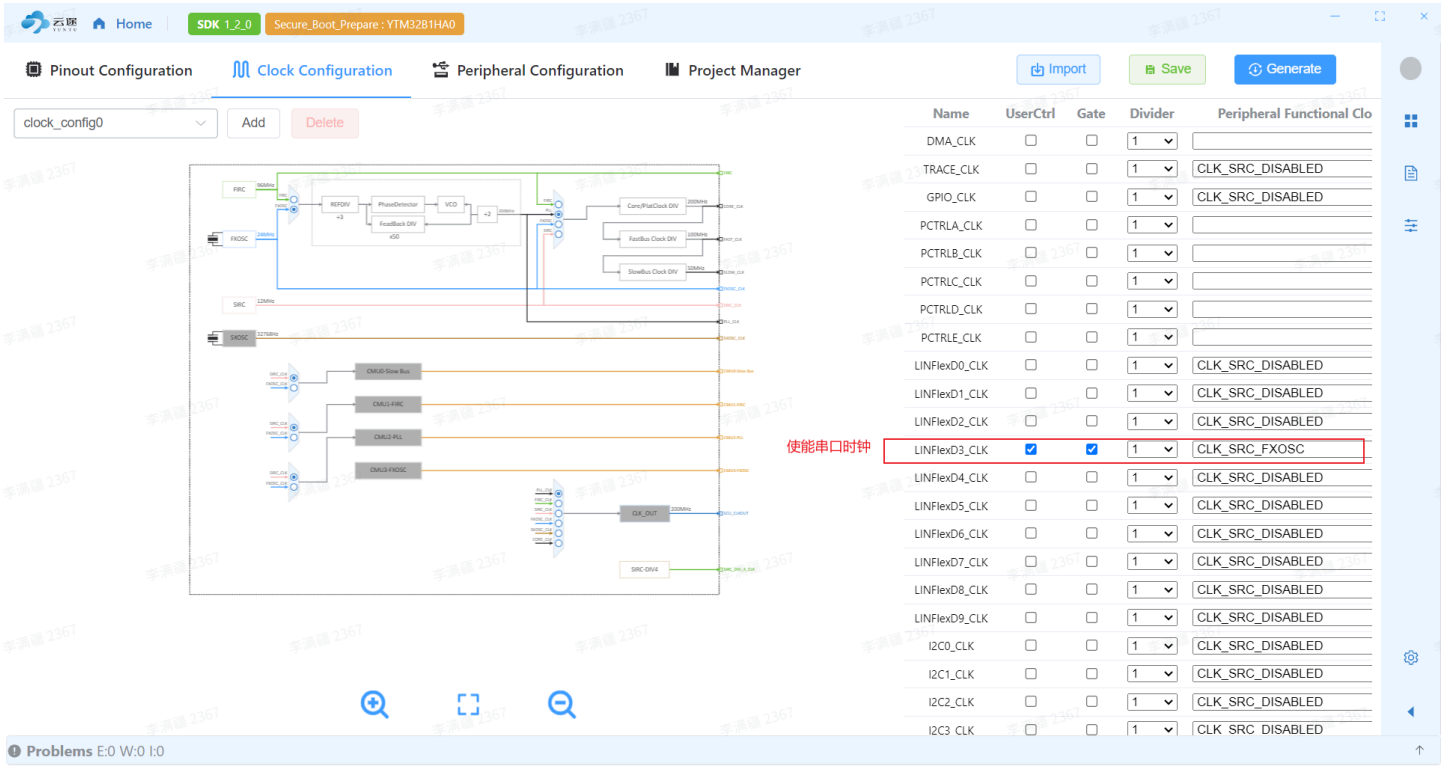
3.1.1.2 引脚配置

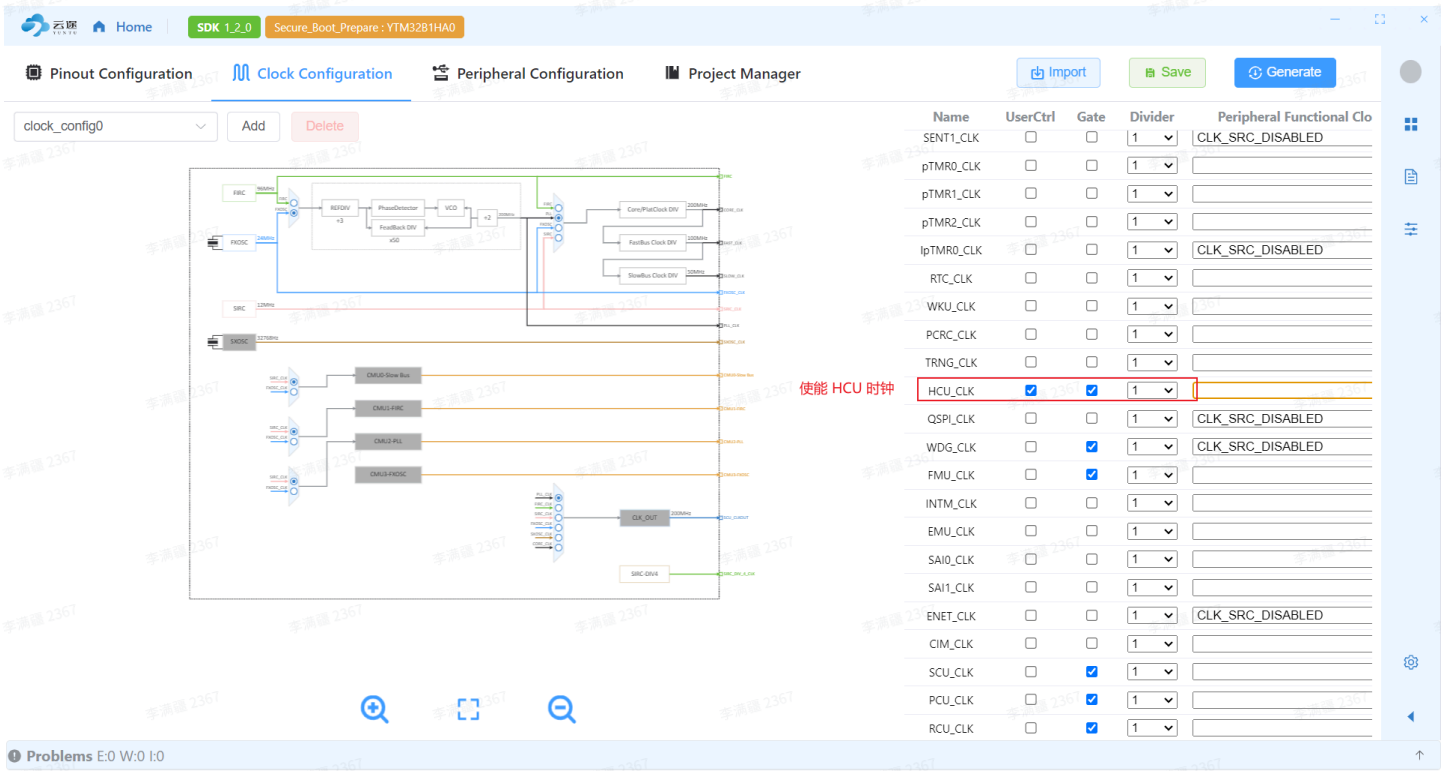
配置 PTD_2 与 PTD_3 作串口的 TX 与 RX 引脚



3.1.1.3 时钟配置

开发板可选择如下时钟配置

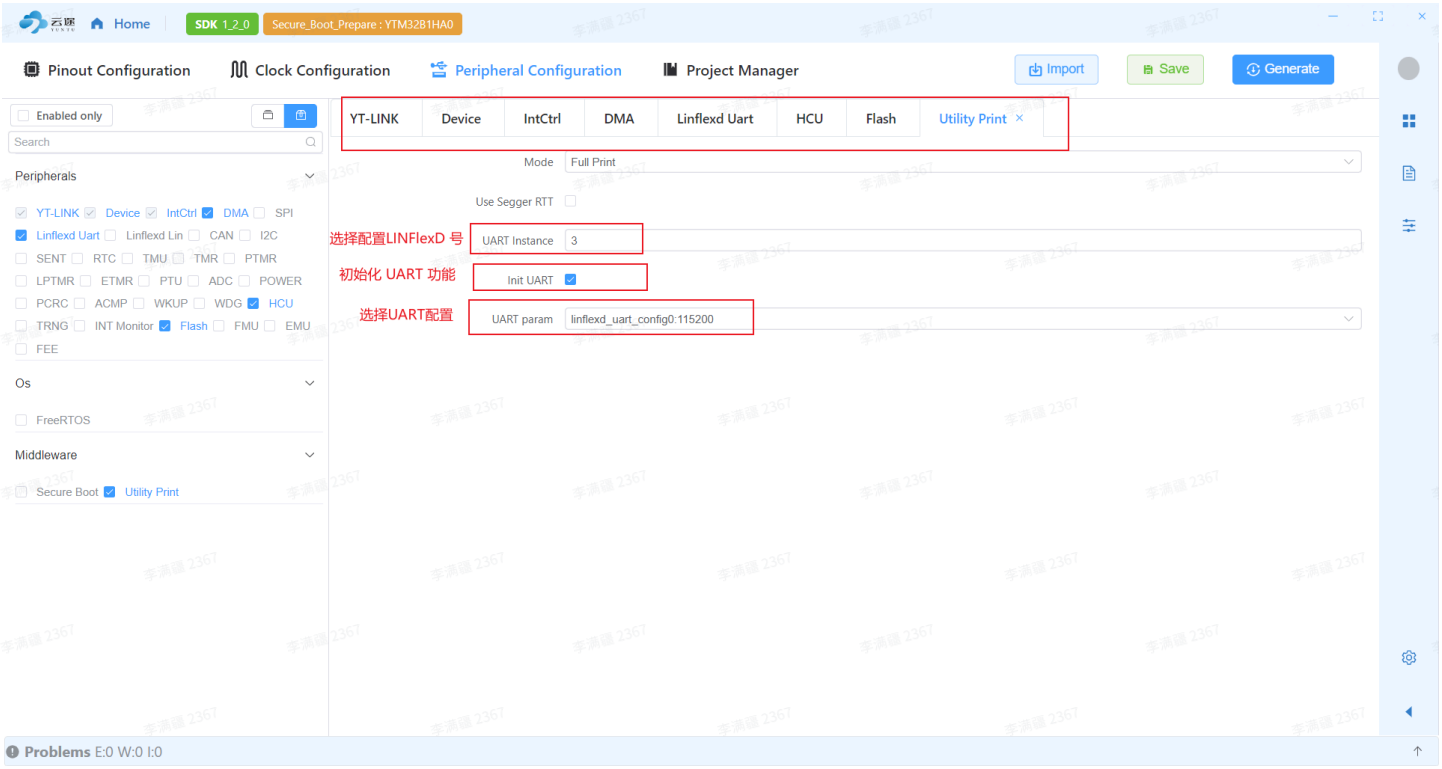




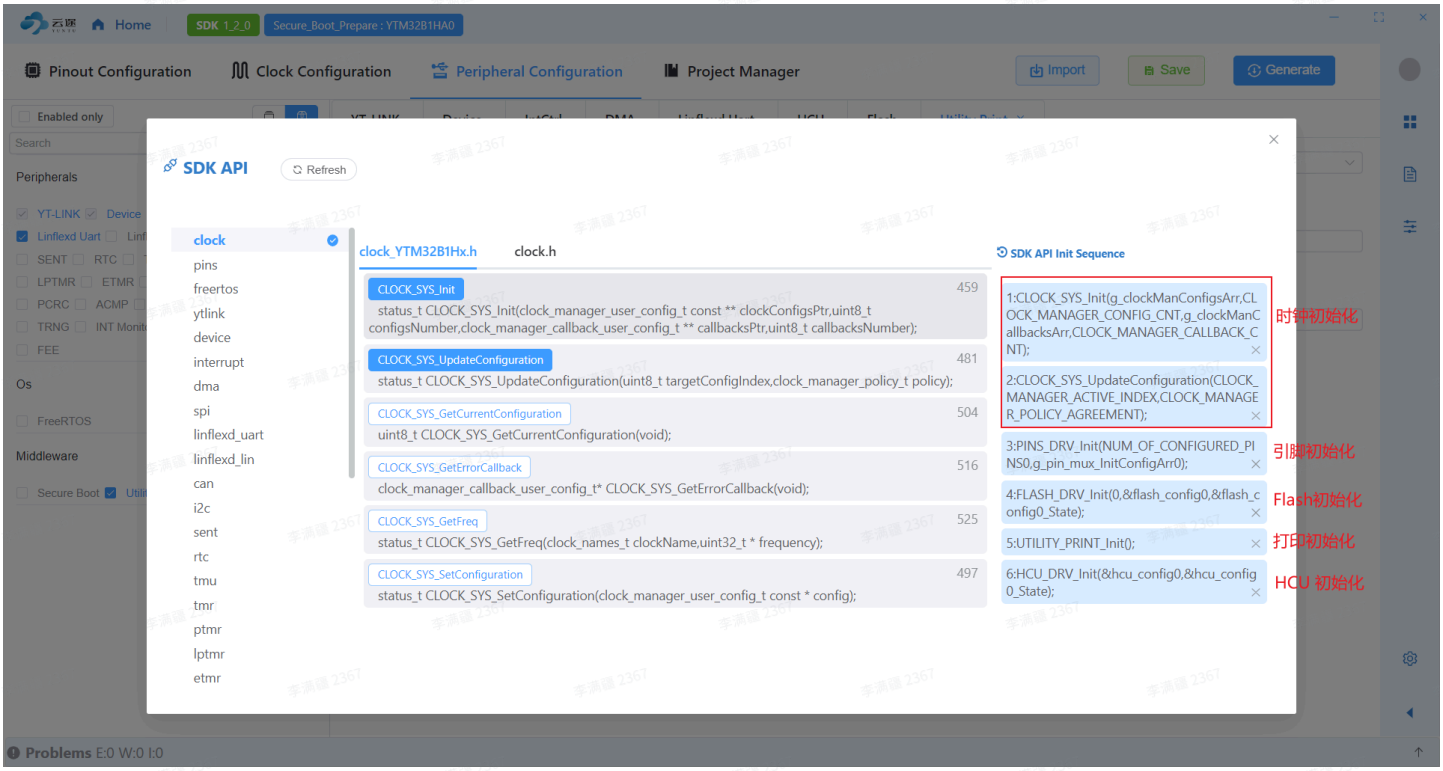
3.1.1.4 模块配置

依次使能 YT-LINK，Device，IntCtrl，DMA，Linflexd Uart，HCU，Flash，选择默认配置即可。

使能 Utility Print，配置如下

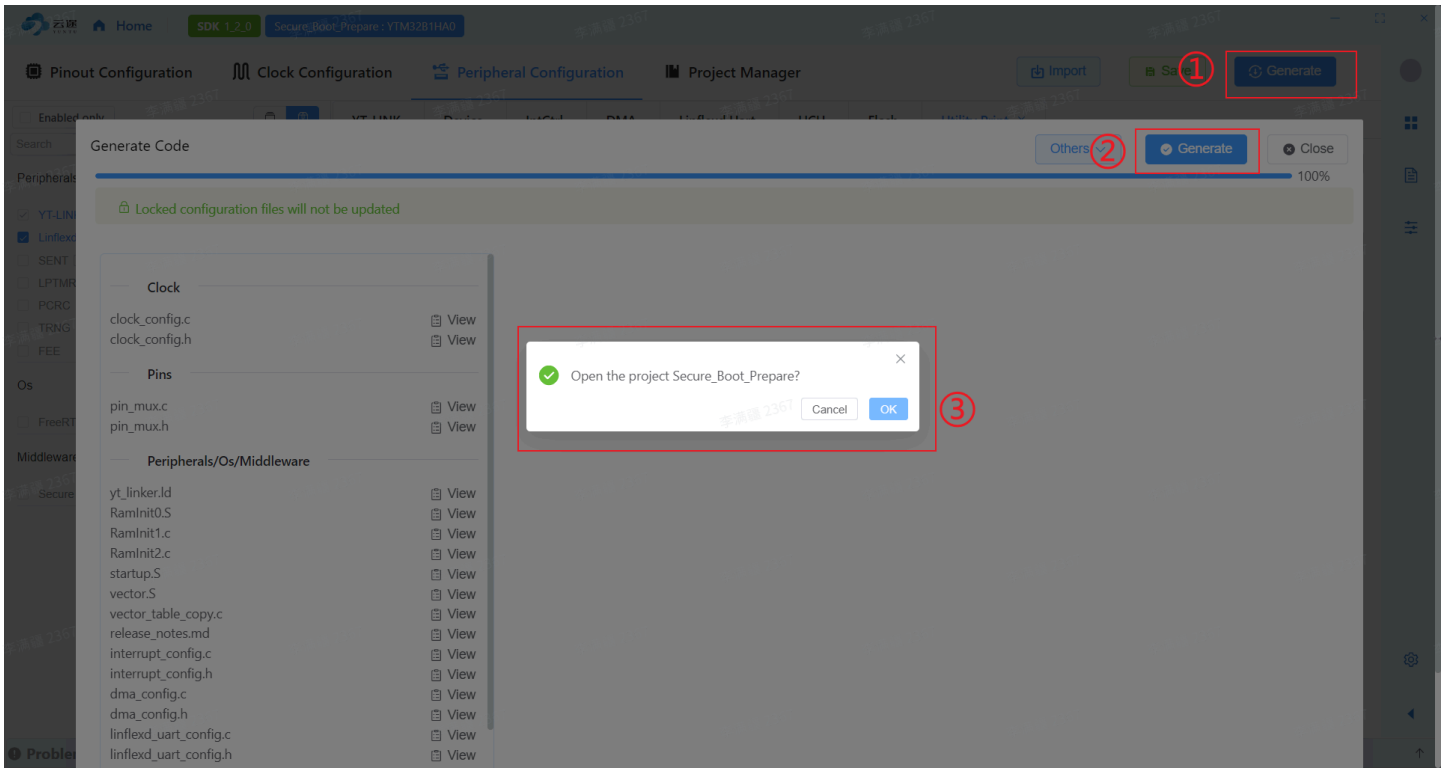


3.1.1.5 API 配置



3.1.1.6 工程生成

依次点击 Generate 生成工程与代码。



3.1.2 生成代码

3.1.2.1 代码块

Demo 中主要有如下几块内容

```
34
35  /* Private define ------------------------------------------------------------*/
36  /* USER CODE BEGIN PD */
37  #define HCU_NVR_START       (0x10000800)   HCU_NVR 的起始地址
38  #define HCU_NVR_SECTOR_SIZE (0x800)
39  /* USER CODE END PD */
40
41  /* Private macro -------------------------------------------------------------*/
42  /* USER CODE BEGIN PM */
43  /* USER CODE END PM */
44
45  /* Private variables ---------------------------------------------------------*/
46  /* USER CODE BEGIN PV */
47  /* HCU key used for secure boot */
48  const uint32_t keyNvr[48] = { 0x2b7e1516, 0x28aed2a6, 0xabf71588, 0x09cf4f3c, 0xffffffff, 0xffffffff, 0xffffffff,
49                                0xffffffff, 0x03020100, 0x07060504, 0x0b0a0908, 0x0f0e0d0c, 0xffffffff, 0xffffffff,
50                                0xffffffff, 0xffffffff, 0x33221100, 0x77665544, 0xbbaa9988, 0xffeeddcc, 0xffffffff,
51                                0xffffffff, 0xffffffff, 0xffffffff, 0x76543210, 0xfedcba98, 0x01234567, 0x89abcdef,
52                                0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff, 0xade12372, 0xfed23123, 0xaf231f41,
53                                0xcdf21098, 0x2b7e1516, 0x0f0e0d0c, 0xffffffff, 0xffffffff, 0xaefc2334, 0x34fe2124,
54                                0x213805fe, 0x23efb2c2, 0xfed23123, 0x77665544, 0xaefc2334, 0x89abcdef };
55  /* AES-ECB plain data */
56  const uint32_t plainText[16] = { 0x6bc1bee2, 0x2e409f96, 0xe93d7e11, 0x7393172a, 0xae2d8a57, 0x1e03ac9c,
57                                   0x9eb76fac, 0x45af8e51, 0x30c81c46, 0xa35ce411, 0xe5fbc119, 0x1a0a52ef,
58                                   0xf69f2445, 0xdf4f9b17, 0xad2b417b, 0xe66c3710 };
59  /* AES-ECB cipher data */
60  const uint32_t cipherText[16] = { 0xC48D5784, 0xF6BB1688, 0xA1F6AC18, 0xEEDAB413, 0x39CBB8F3, 0x7FAB8C25,
61                                    0x3B1C1E50, 0x8C53E9D0, 0xDF273015, 0x3603F913, 0xD4298D27, 0xBB91F6FA,
62                                    0xF95B31D0, 0x9053DFF1, 0x214A42EF, 0xF22A080D };
63  uint32_t sw_encrypt_result[16];
64  uint32_t hw_encrypt_result[16];
65  /* USER CODE END PV */
66
67  /* Private function declare --------------------------------------------------*/
68  /* USER CODE BEGIN PFDC */
69  /* USER CODE END PFDC */
70  static void Board_Init(void);
71
```

Secure Boot 过程中使用到的密钥

AES-ECB 明文，用于测试

AES-ECB 密文，用于测试

```c
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
/* Program HCU key */
static status_t Program_HcuKey(const void *key, uint32_t length)
{
    status_t status = STATUS_SUCCESS;
    /* Erase HCU NVR, user should be careful of it */
    EFM->CUS_KEY = 0x4dff32;
    status |= FLASH_DRV_EraseSector(0, HCU_NVR_START, HCU_NVR_SECTOR_SIZE);
    /* Program HCU key */
    status |= FLASH_DRV_Program(0, HCU_NVR_START, length, key);
    return status;
}

/* Check HCU key load success */
static status_t Check_HcuKeyLoad(void)
{
    status_t status = STATUS_SUCCESS;

    /* Load hardware key */
    INT_SYS_DisableIRQGlobal();
    status |= FLASH_DRV_LoadAESKey(0, HCU_NVR_START);
    INT_SYS_EnableIRQGlobal();
    HCU_SetKeySize(KEY_SIZE_128_BITS);
    /* HCU ECB encrypt with hardware key */
    status |= HCU_DRV_EncryptECB(plainText, 64, hw_encrypt_result);

    /* Load software key */
    status |= HCU_DRV_LoadUserKey(keyNvr, KEY_SIZE_128_BITS);
    /* Start AES-ECB software encrypt */
    status |= HCU_DRV_EncryptECB(plainText, 64, sw_encrypt_result);
    /* Check result */
    for (uint32_t i = 0; i < 16; i++)
    {
        if (sw_encrypt_result[i] != cipherText[i])
        {
            PRINTF("Software key mismatch!\n");
            return STATUS_ERROR;
        }
        if (hw_encrypt_result[i] != cipherText[i])
        {
            PRINTF("Hardware key mismatch!\n");
            return STATUS_ERROR;
        }
    }
    PRINTF("HCU key load success!\n");
    return status;
}

/* USER CODE END 0 */
```

对HCU_NVR进行编程，密钥存储于此

加载硬件密钥

使用硬件密钥对明文加密

加载软件密钥

使用软件密钥对明文加密

检查使用软件密钥加密后的密文与正确值是否一致

检查使用硬件密钥加密后的密文与正确值是否一致

```c
/**
 * @brief  The application entry point.
 * @retval int
 */
int main(void)
{
    /* USER CODE BEGIN 1 */
    status_t status = STATUS_SUCCESS;
    /* USER CODE END 1 */
    Board_Init();
    /* USER CODE BEGIN 2 */
    PRINTF("Secure boot prepare demo!\n");
    Program_HcuKey(keyNvr, sizeof(keyNvr));
    Check_HcuKeyLoad();
    /* USER CODE END 2 */

    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1)
    {
        if (status != STATUS_SUCCESS)
        {
            break;
        }
        /* USER CODE END WHILE */
        /* USER CODE BEGIN 3 */
    }
    /* USER CODE END 3 */
}

static void Board_Init(void)
{
    CLOCK_SYS_Init(g_clockManConfigsArr,CLOCK_MANAGER_CONFIG_CNT,g_clockManCallbacksArr,CLOCK_MANAGER_CALLBACK_CNT);
    CLOCK_SYS_UpdateConfiguration(CLOCK_MANAGER_ACTIVE_INDEX,CLOCK_MANAGER_POLICY_AGREEMENT);
    PINS_DRV_Init(NUM_OF_CONFIGURED_PINS0,g_pin_mux_InitConfigArr0);
    FLASH_DRV_Init(0,&flash_config0,&flash_config0_State);
    UTILITY_PRINT_Init();
    HCU_DRV_Init(&hcu_config0,&hcu_config0_State);
}

/* USER CODE BEGIN 4 */
/* USER CODE END 4 */
```

编程 HCU_NVR，将用户密钥编入

检查硬件密钥是否编程成功

模块初始化，由 YT_Config_Tool 自动生成

3.1.2.2 注意事项

1. 不同芯片的 HCU_NVR 的起始地址与大小都不同，需要单独配置，即代码块的 line 37~38
2. 不同芯片对 HCU_NVR 的处理方式不同，例如 YTM32B1MD14 **不可擦除** HCU_NVR，其余几款芯片可以擦除 HCU_NVR，但是需要 **Customer Key**，Customers Key 默认值可在 Reference Manual 中查看，用户可自行修改 Customers Key，只有用户自己知道。HCU_NVR 编程没有限制，只需要保证要写入的 HCU_NVR 为空即可。详情可查看 **AN_0062_EFM_ApplicationNote.pdf**
3. 不同芯片的 HCU 密钥长度不一样，例如 YTM32B1HA01 最高支持 256-bit 密钥，所以使用 ***FLASH_DRV_LoadAESKey()*** 时，HCU 默认加载的是 256-bit 密钥，但当前希望使用的是 128-bit 密钥，所以需要 ***HCU_SetKeySize(KEY_SIZE_128_BITS)*** 来修改 HCU 加载的硬件密钥长度。详情可查看 **AN_0063_HCU_ApplicationNote.pdf**

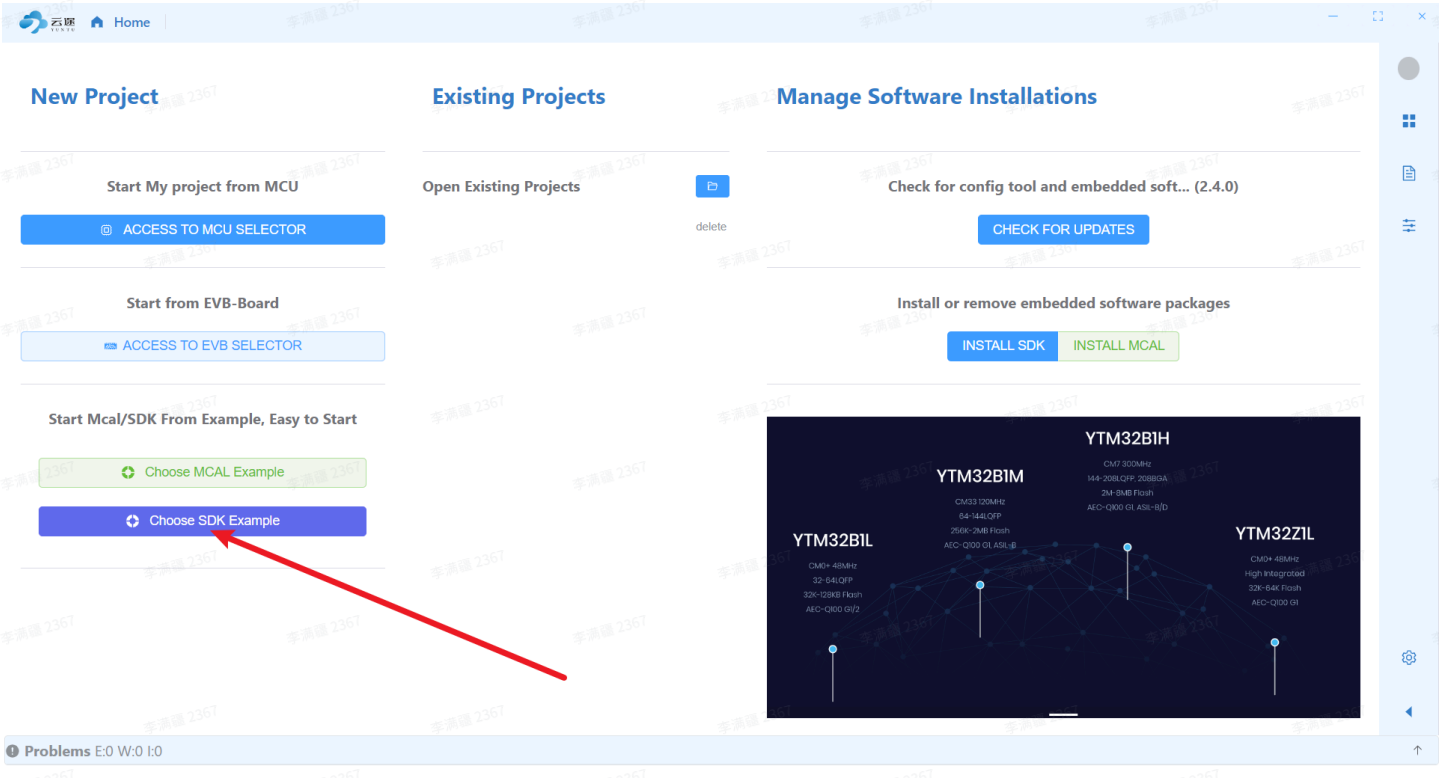
3.1.2.3 测试结果

编译完成后，将程序下载至芯片中，复位或重新上电，看到串口打印如下内容，代表密钥编程成功。



3.2 Secure Boot Demo

用户可以直接点开 SDK Example 选择 YTM32B1HA01 的 Secure Boot demo

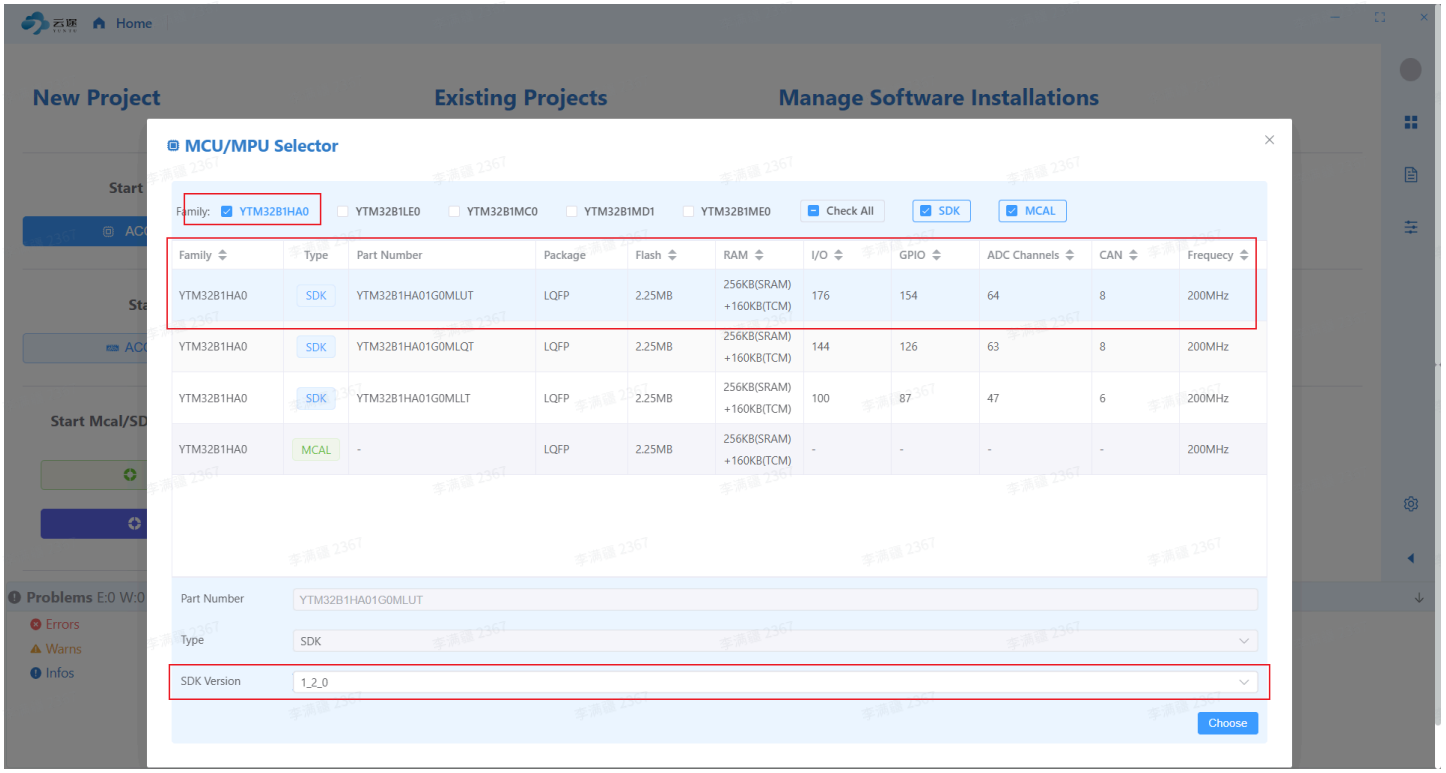


或者跟随下文一步步重新创建工程。

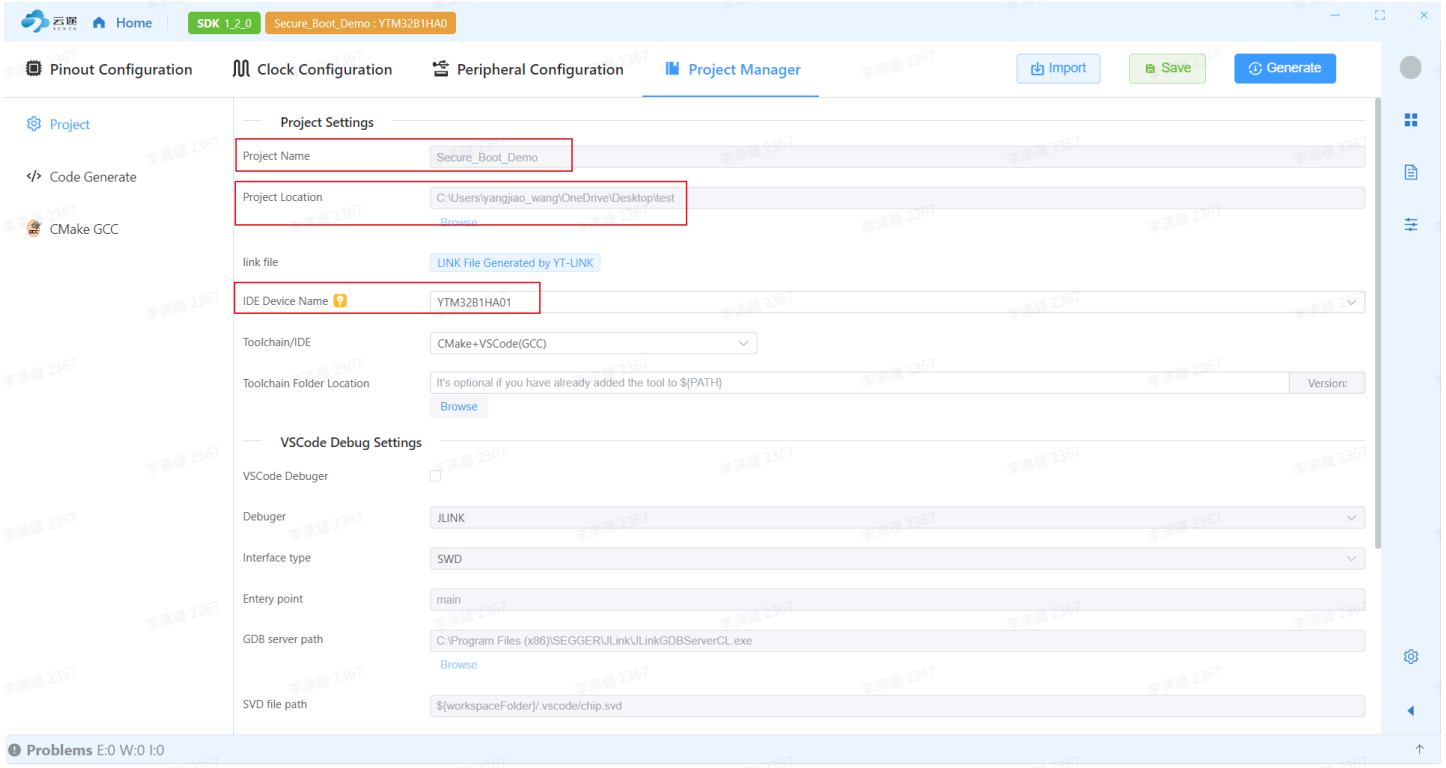
3.2.1 基础配置

3.2.1.1 创建工程

选择 YTM32B1HA01，选择当前芯片的具体型号，选择 SDK 版本号（≥1.2.0）

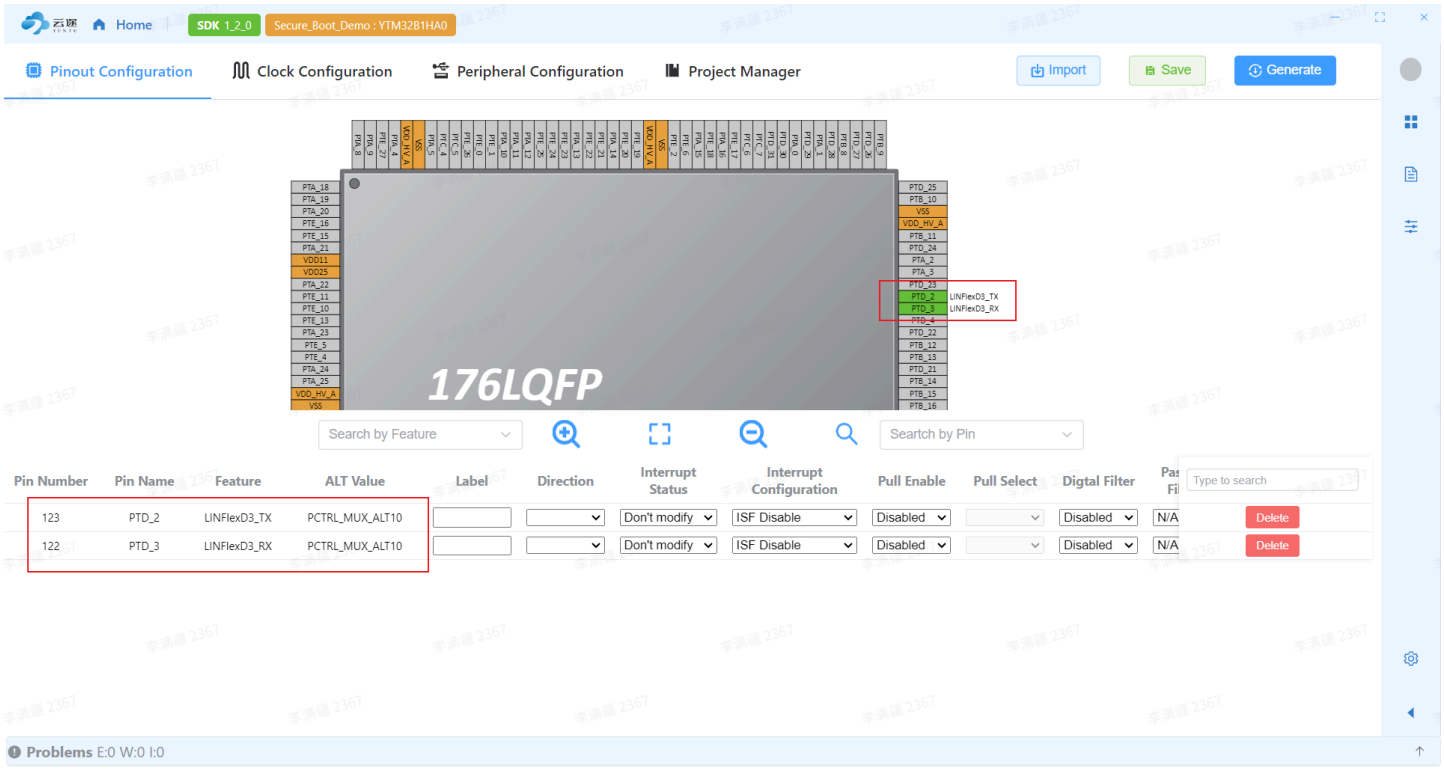


设置项目名字，工程路径，以及 IDE Device Name



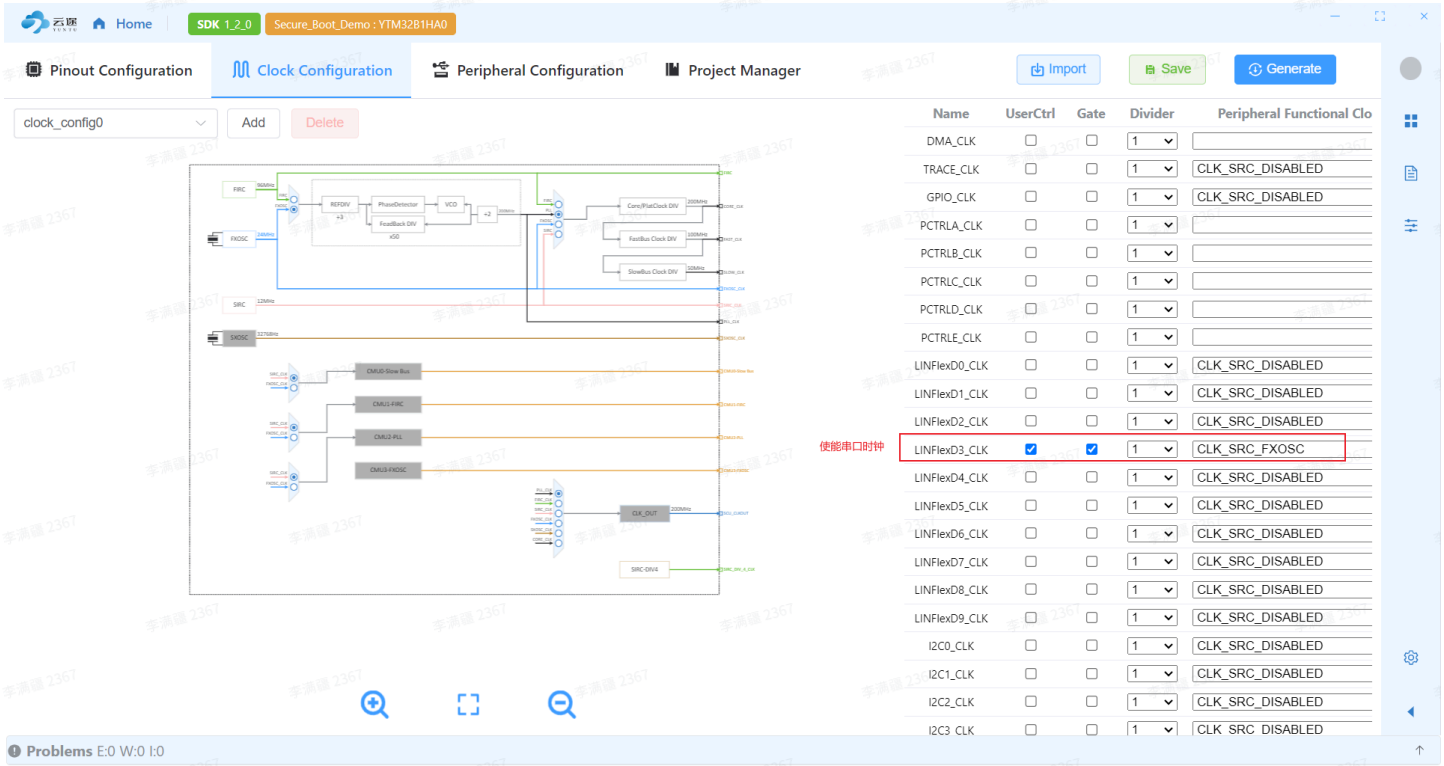
3.2.1.2 引脚配置

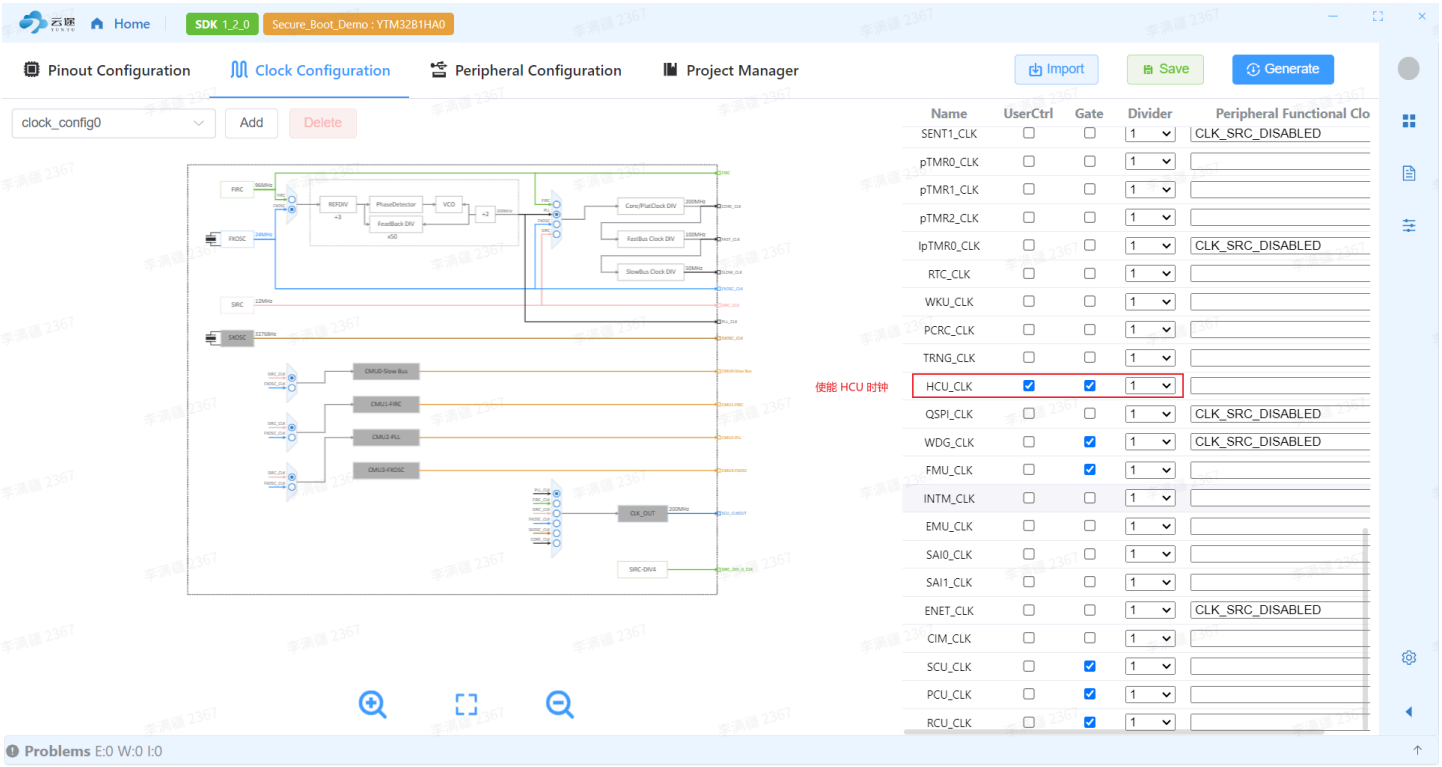
配置 PTD_2 与 PTD_3 作串口的 TX 与 RX 引脚



3.2.1.3 时钟配置

开发板可选择如下时钟配置

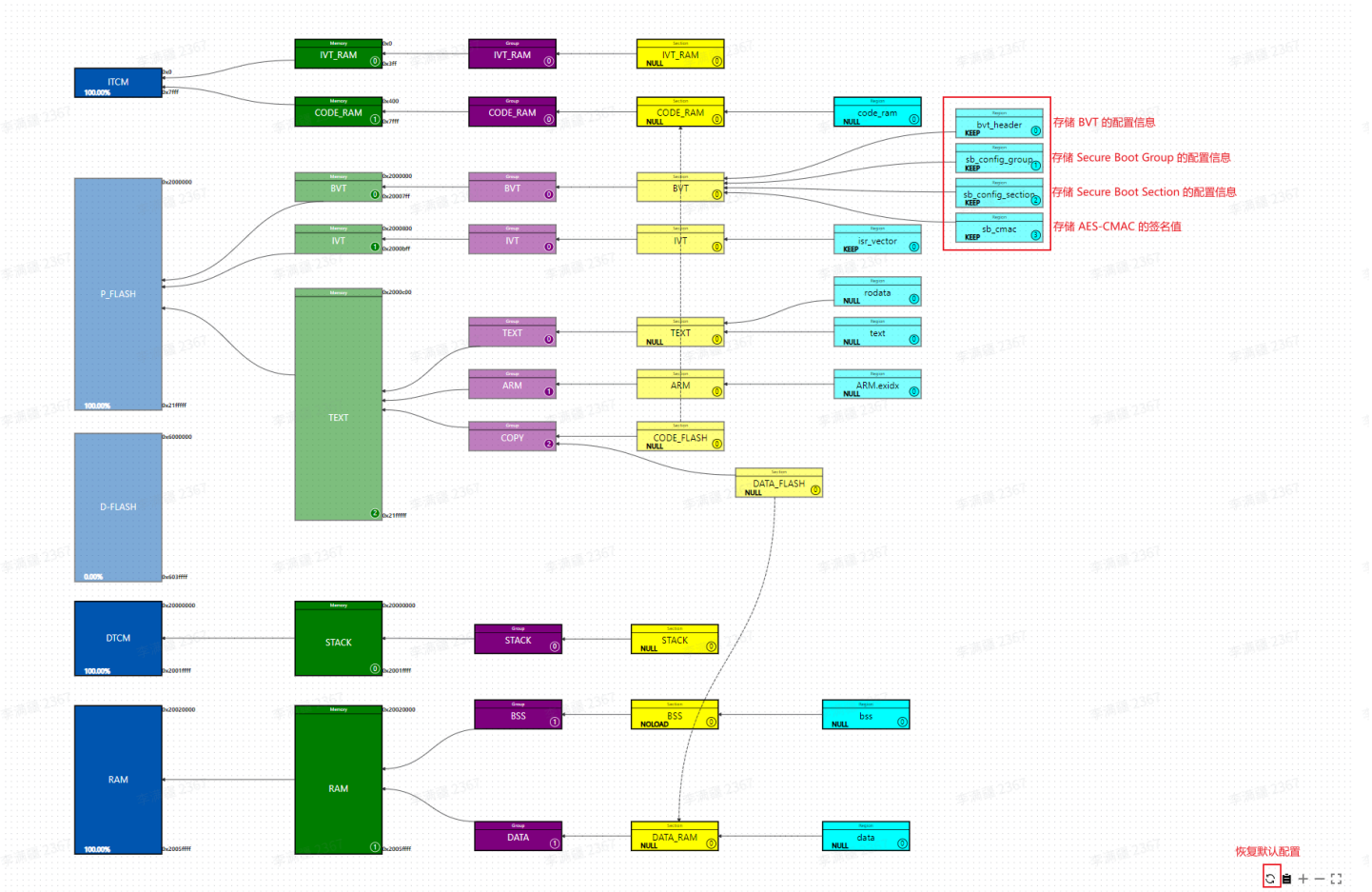




3.2.1.4 YT-LINK 配置

Secure Boot 由于需要 BVT，Secure Boot Group，Secure Boot Section 与 CMAC，**Link File** 需要作修改。现通过 YT-LINK 增加四个 Region 用于存储 Secure Boot 相关配置信息。YT-LINK 的使用可参考 YT-LINK User Manual。

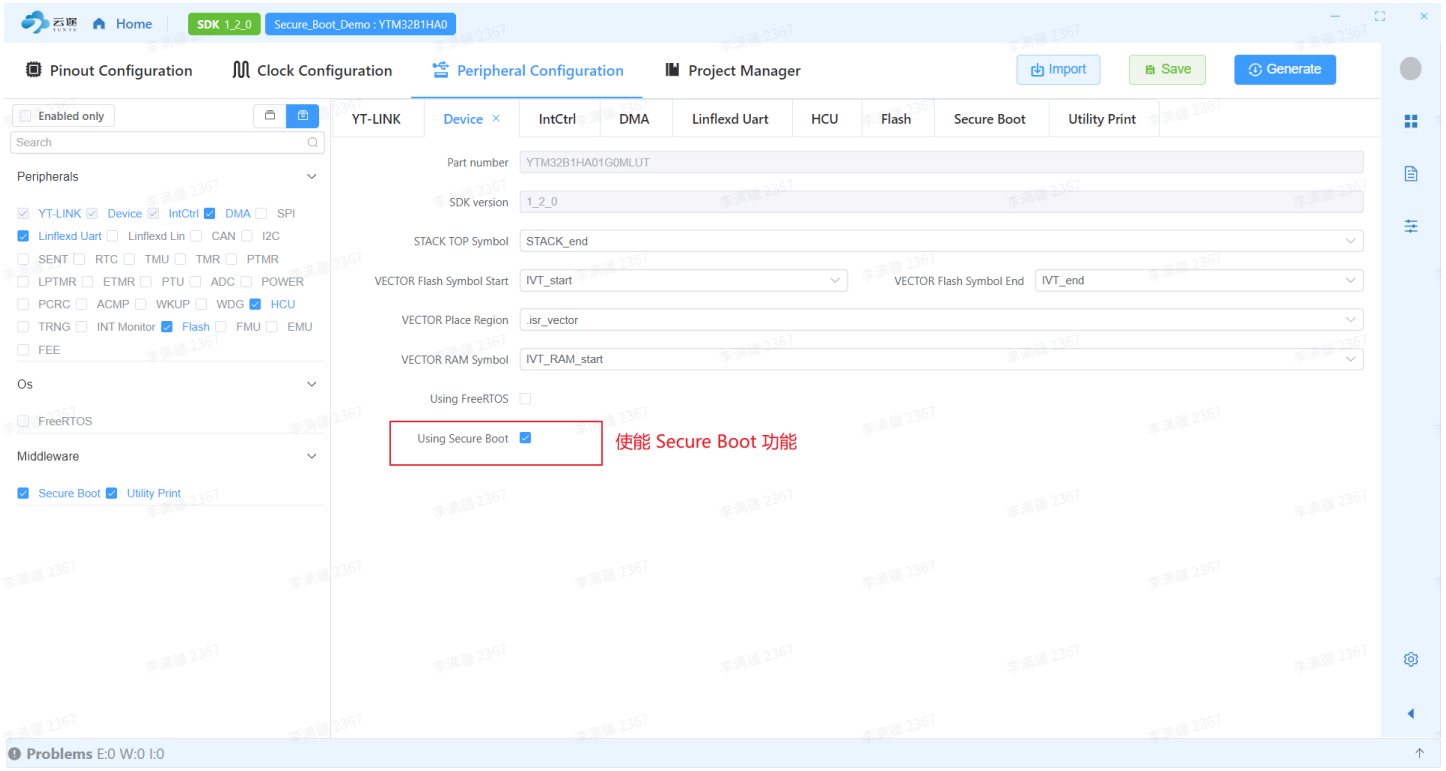
注意 BVT 放置的位置必须符合规定，参考 2.2 章节。



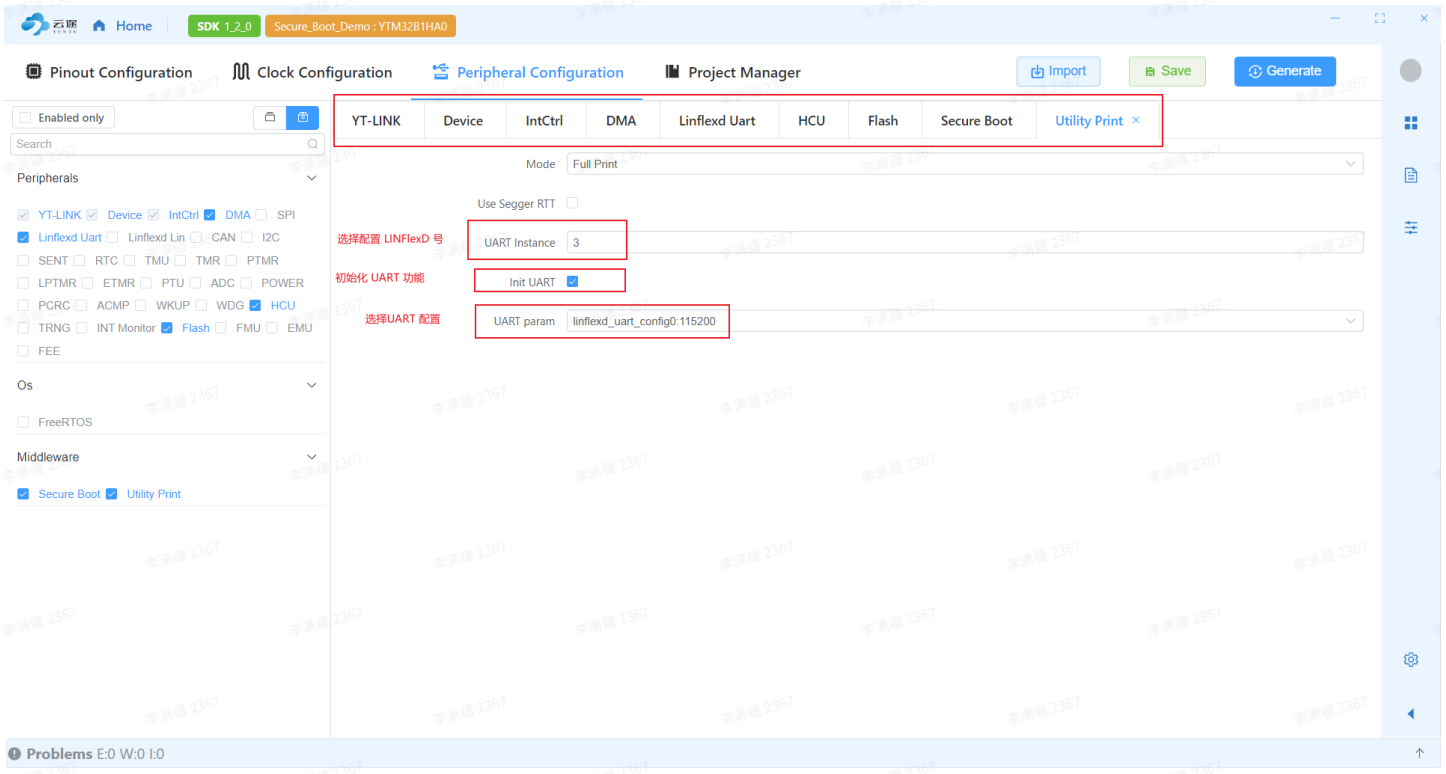
3.2.1.5 模块配置

依次使能 IntCtrl，DMA，Linflexd Uart，HCU，Flash，选择默认配置即可。

配置 Device 模块



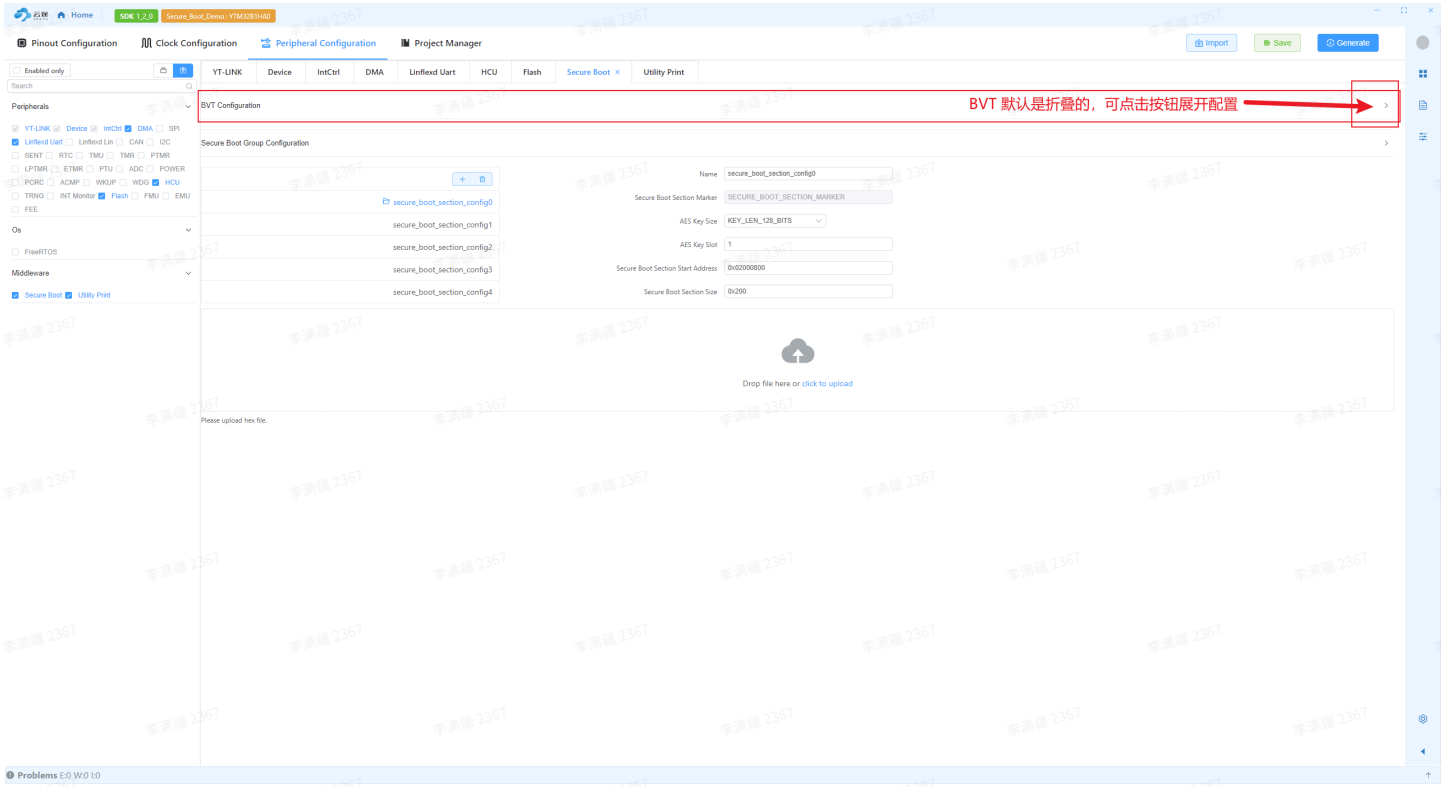
使能 Utility Print，配置如下

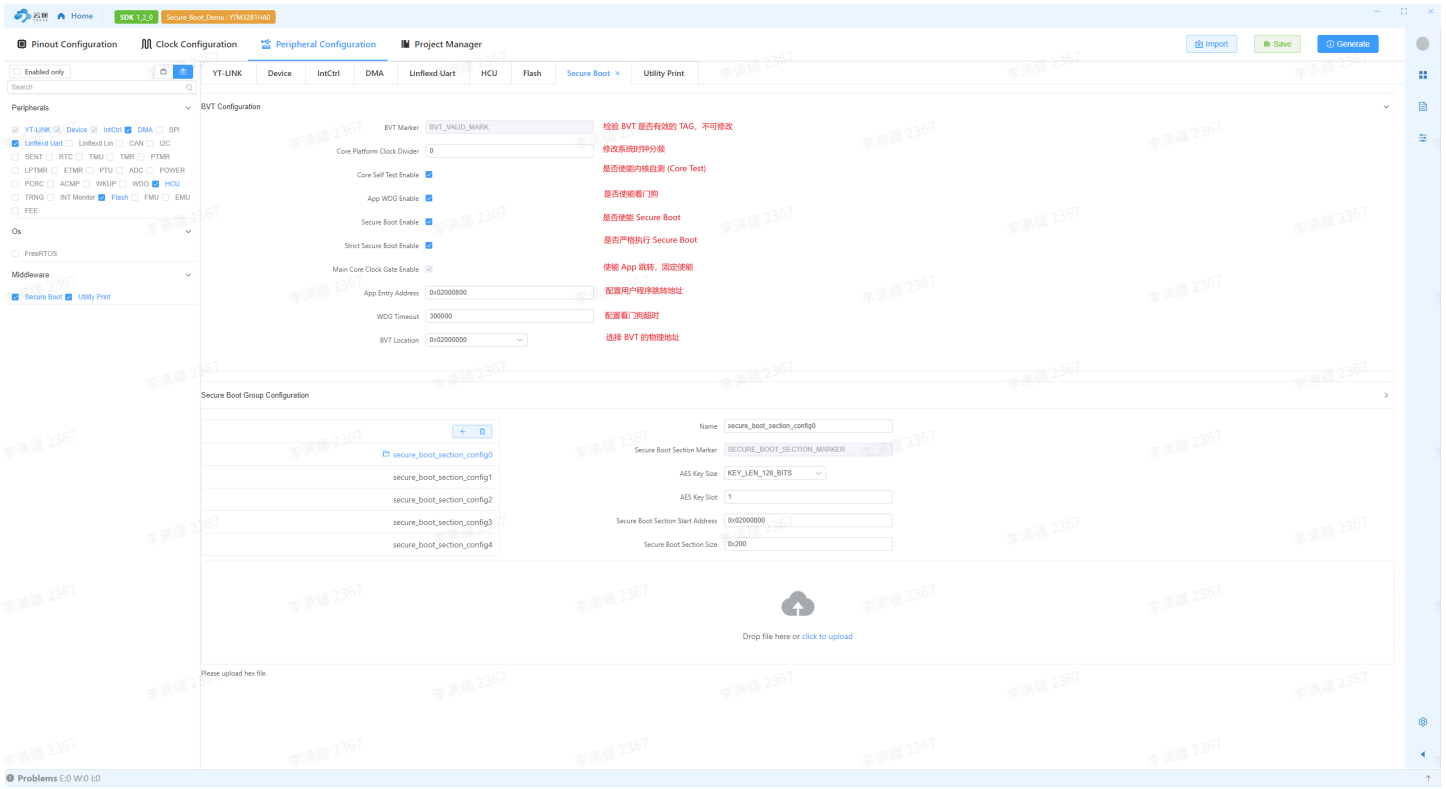


3.2.1.6 Secure Boot 配置

Secure Boot 需要单独配置 BVT，Secure Boot Group 与 Secure Boot Section，具体可参考如下配置

3.2.1.6.1 BVT 配置



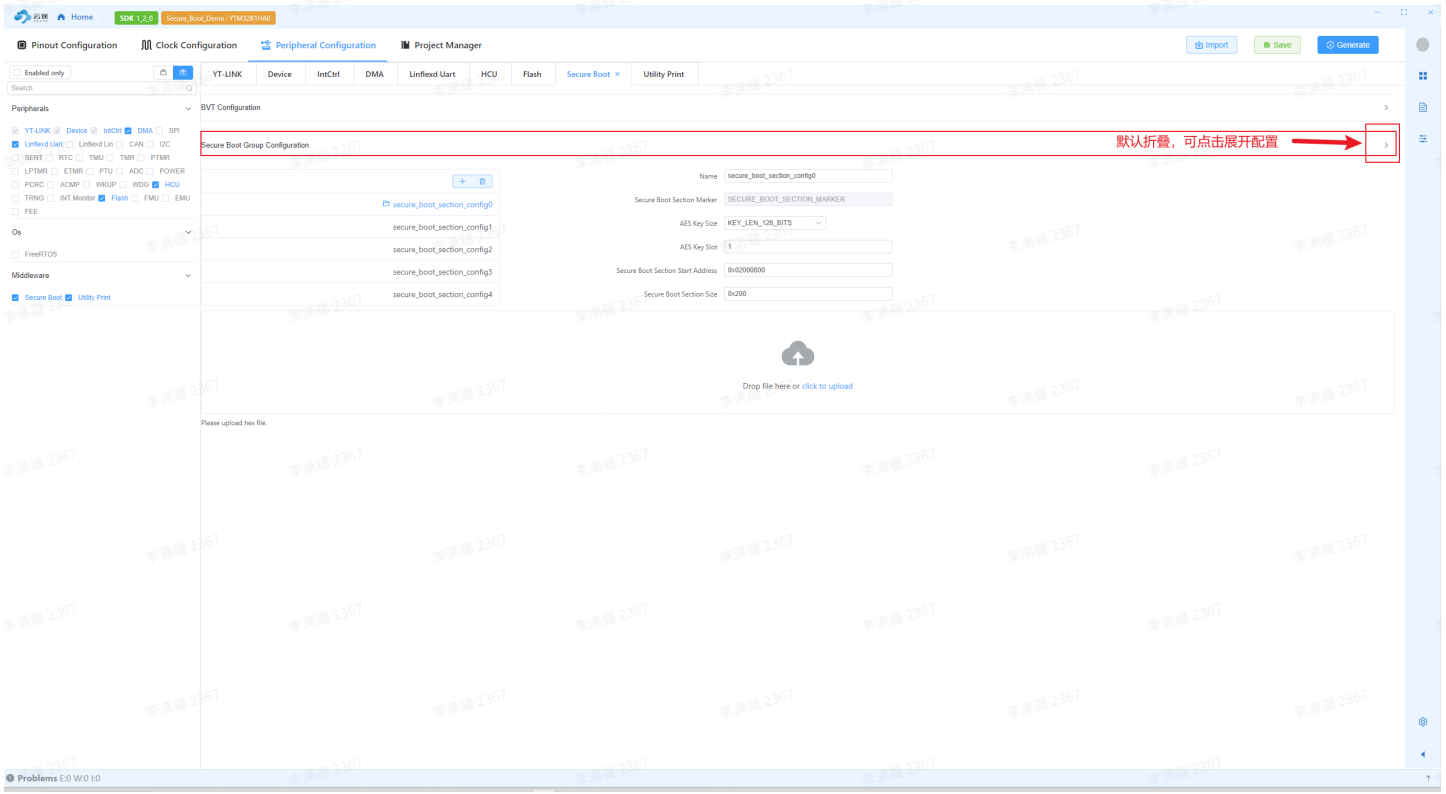


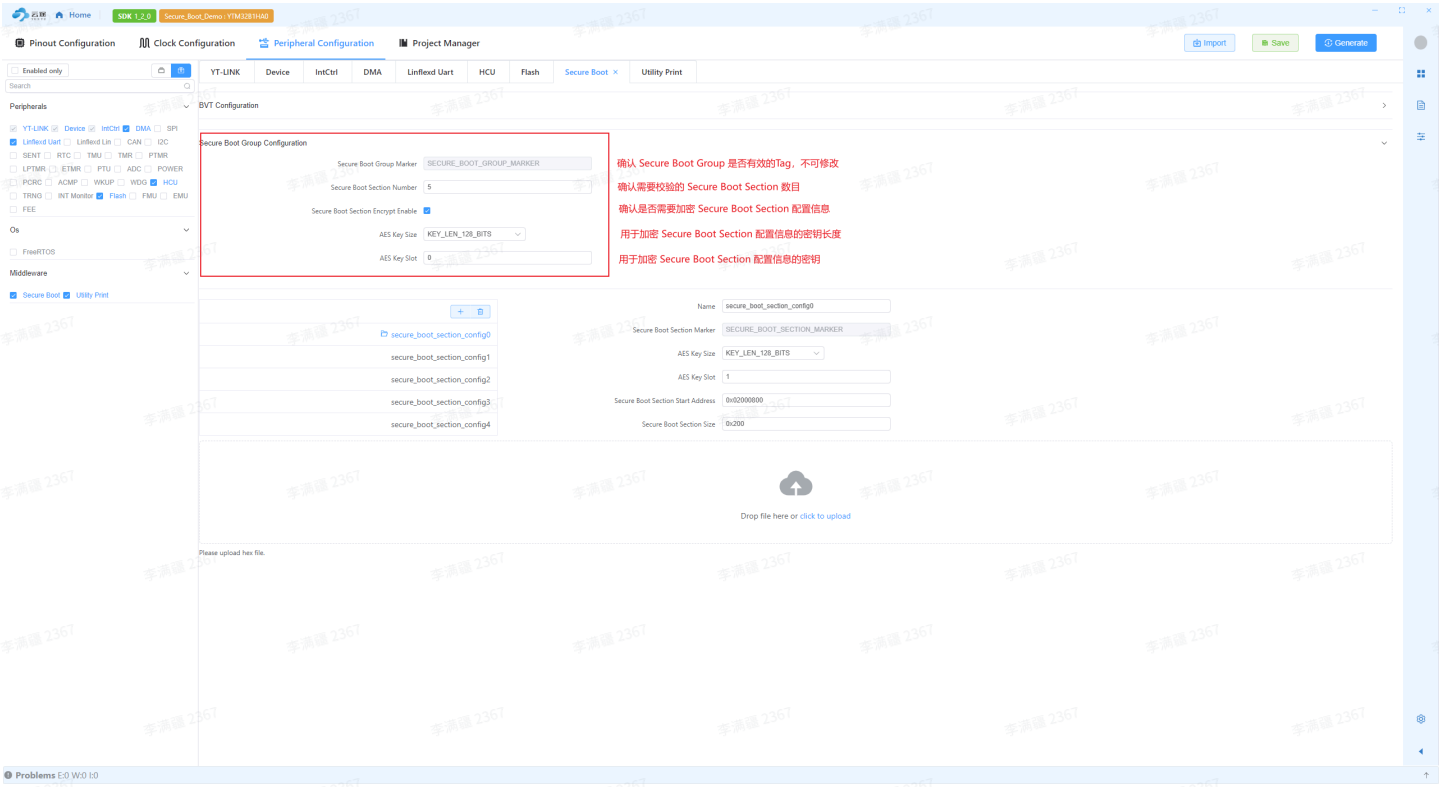
注意：不同芯片的 BVT 配置不一样，下面统一进行汇总。

- BVT Marker：确认 BVT 是否有效的 Tag，为固定值，不可修改
- PLL set as Core Clock：确认是否将 PLL 作系统时钟以加速启动流程
- Core Platform Clock Divider：系统时钟分频
- Core Self Test Enable：确认是否使能内核自测试（Core Test）
- WDG Re-config Enable：确认是否重新配置 WDG。该款芯片默认上电后就使能 WDG，并有默认配置。使能该位可在 Secure Boot 期间重新配置。

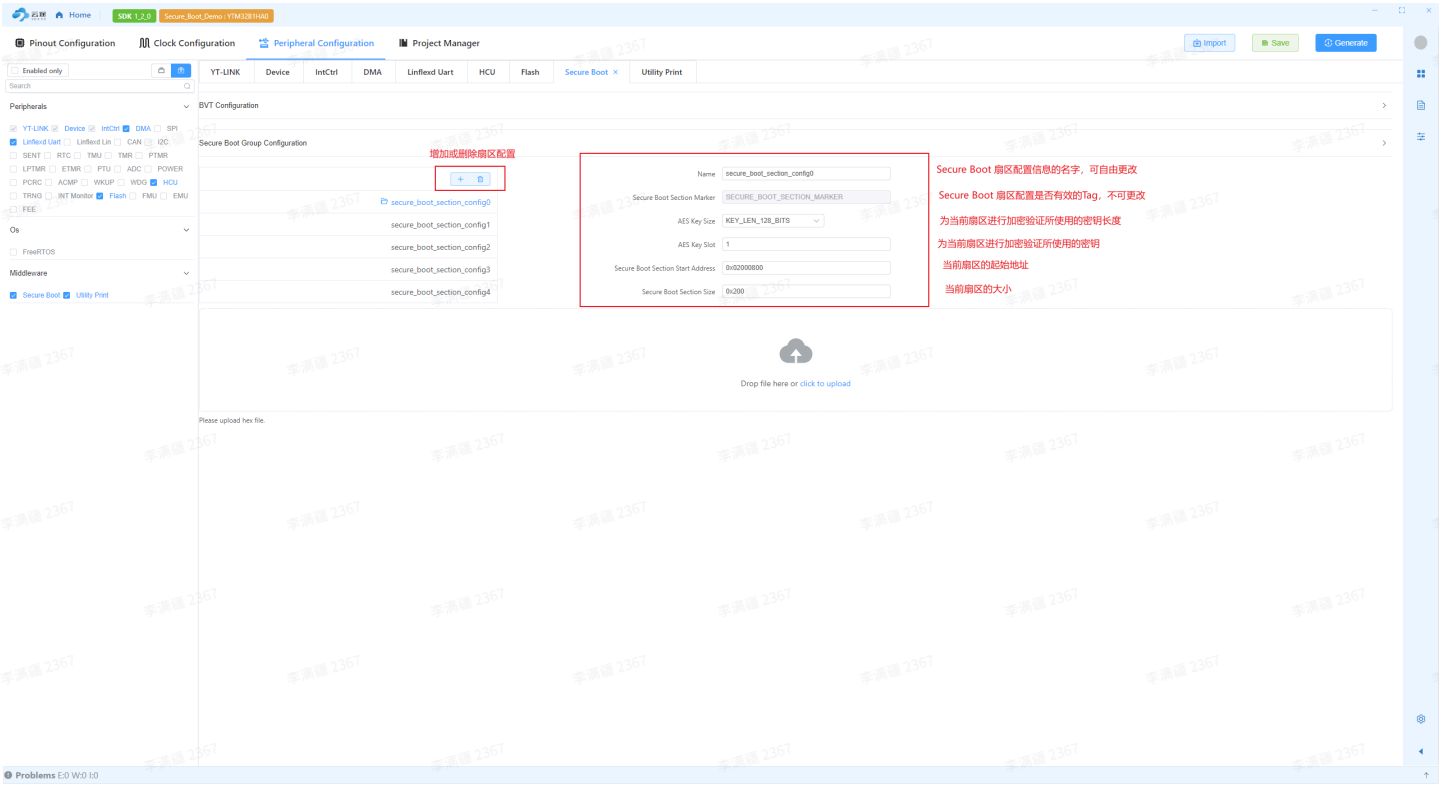
- WDG Clock Source：选择看门狗的计数时钟
- App WDG Enable：确认是否使能看门狗
- Secure Boot Enable：确认是否使能安全启动
- Strict Secure Boot Enable：确认是否严格执行安全启动
- Main Core Clock Gate Enable：使能 App 跳转，固定使能，不可修改
- App Entry Address：应用程序跳转地址
- WDG Timeout：看门狗超时配置
- BVT Location：BVT 实际存在的物理地址。(必须与 YT-LINK 中设置的位置一致)

3.2.1.6.2 Secure Boot Group 配置



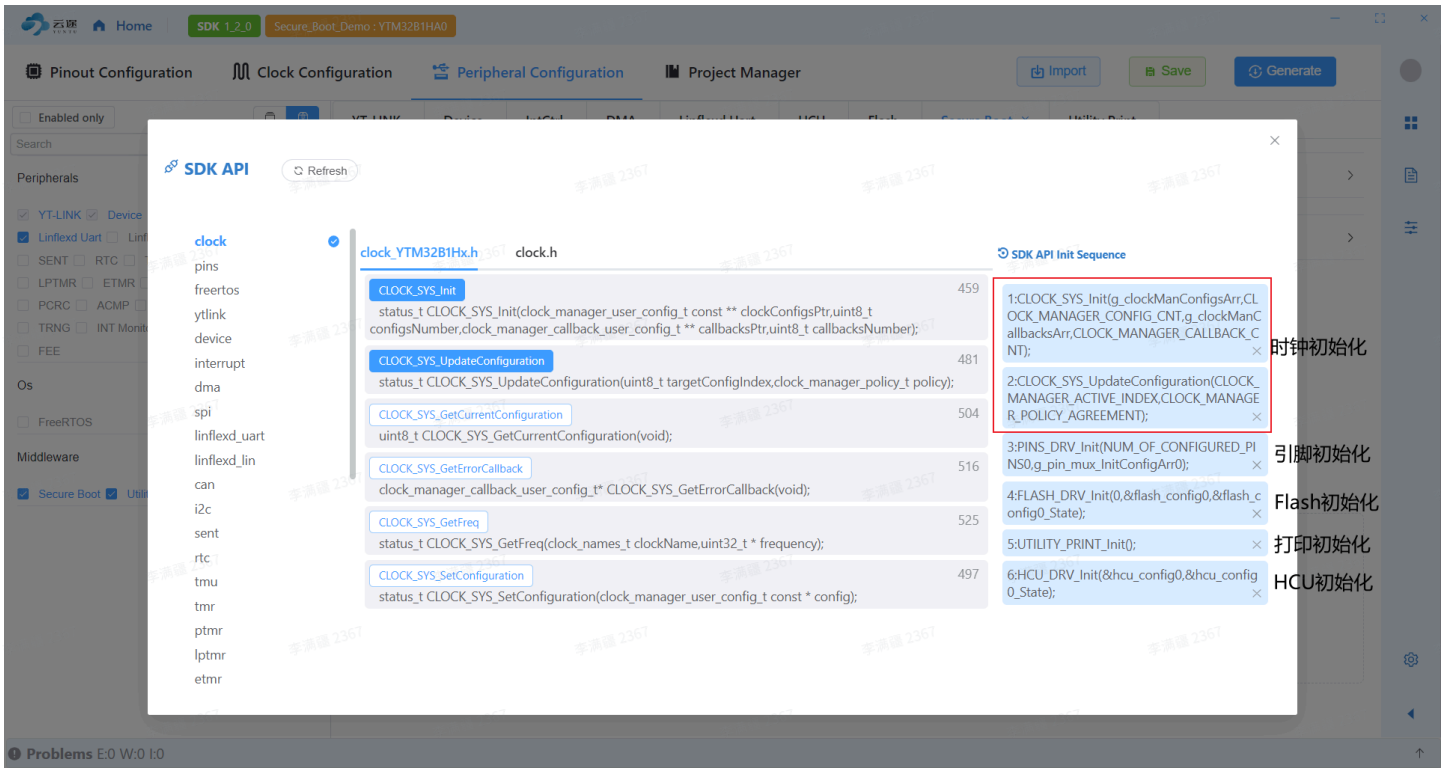


3.2.1.6.3 Secure Boot Section 配置

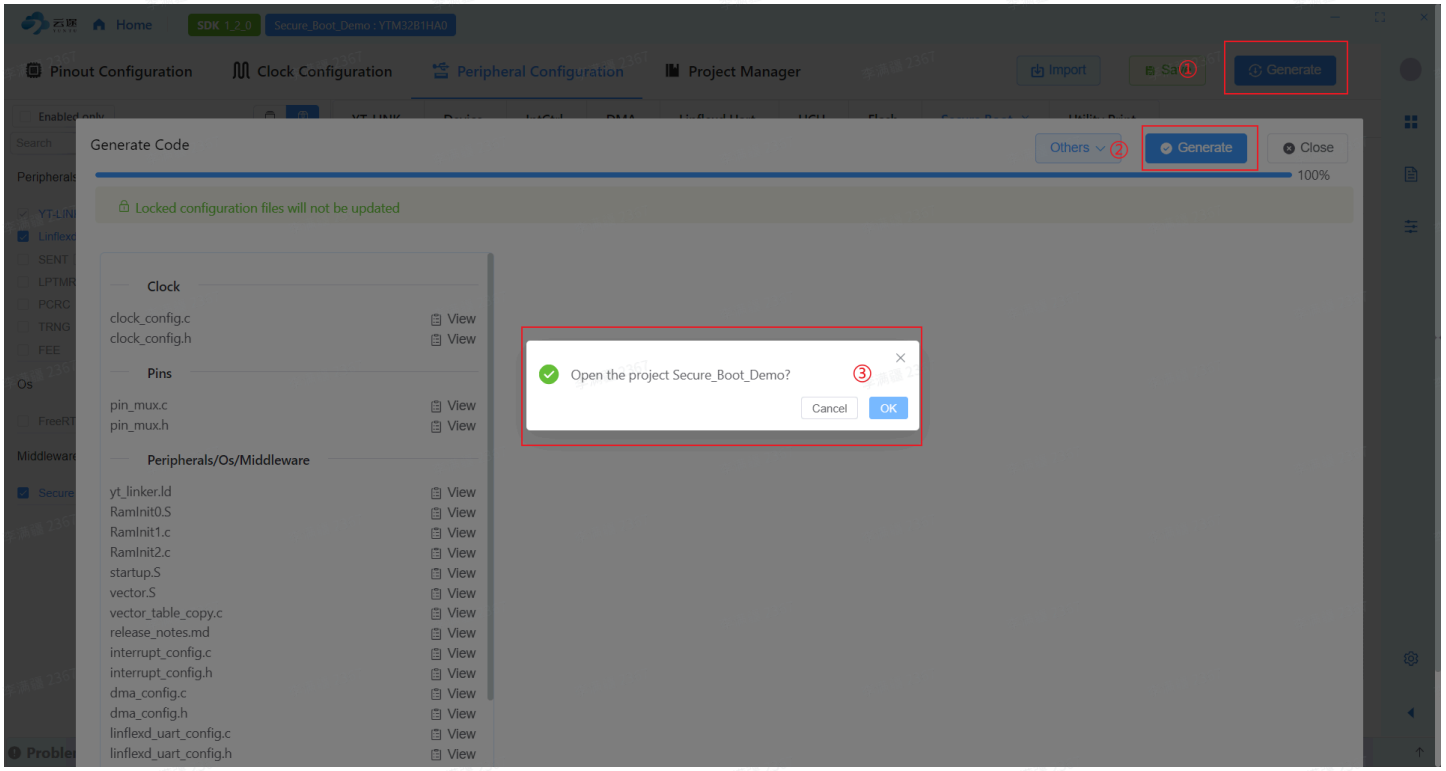


注意：所保护的扇区不可包含可变区域。因为若该区域内容发生变化，则产生的签名势必不一致，导致验签失败。

3.2.1.7 API 配置



3.2.1.8 工程生成



3.2.2 生成代码

代码内容与 Secure Boot Prepare Demo 并无太大区别，只是**删除了密钥编程**的步骤，故不再赘述。该代码只是用来验证 Secure Boot 是否验签成功并跳转至应用程序。

3.2.2.1 代码块

```c
/* Private variables ---------------------------------------------------------*/
/* USER CODE BEGIN PV */
/* HCU key used for secure boot */
const uint32_t keyNvr[48] = { 0x2b7e1516, 0x28aed2a6, 0xabf71588, 0x09cf4f3c, 0xffffffff, 0xffffffff, 0xffffffff,
                              0xffffffff, 0x03020100, 0x07060504, 0x0b0a0908, 0x0f0e0d0c, 0xffffffff, 0xffffffff,
                              0xffffffff, 0xffffffff, 0x33221100, 0x77665544, 0xbbaa9988, 0xffeeddcc, 0xffffffff,
                              0xffffffff, 0xffffffff, 0xffffffff, 0x76543210, 0xfedcba98, 0x01234567, 0x89abcdef,
                              0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff, 0xade12372, 0xfed23123, 0xaf231f41,
                              0xcdf21098, 0x2b7e1516, 0x0f0e0d0c, 0xffffffff, 0xffffffff, 0xaefc2334, 0x34fe2124,
                              0x213805fe, 0x23efb2c2, 0xfed23123, 0x77665544, 0xaefc2334, 0x89abcdef };
/* AES-ECB plain data */
const uint32_t plainText[16] = { 0x6bc1bee2, 0x2e409f96, 0xe93d7e11, 0x7393172a, 0xae2d8a57, 0x1e03ac9c,
                                 0x9eb76fac, 0x45af8e51, 0x30c81c46, 0xa35ce411, 0xe5fbc119, 0x1a0a52ef,
                                 0xf69f2445, 0xdf4f9b17, 0xad2b417b, 0xe66c3710 };
/* AES-ECB cipher data */
const uint32_t cipherText[16] = { 0xC48D5784, 0xF6BB1688, 0xA1F6AC18, 0xEEDAB413, 0x39CBB8F3, 0x7FAB8C25,
                                  0x3B1C1E50, 0x8C53E9D0, 0xDF273015, 0x3603F913, 0xD4298D27, 0xBB91F6FA,
                                  0xF95B31D0, 0x9053DFF1, 0x214A42EF, 0xF22A080D };
uint32_t sw_encrypt_result[16];
uint32_t hw_encrypt_result[16];
/* USER CODE END PV */
```

```c
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* Check HCU key load success */
static status_t Check_HcuKeyLoad(void)
{
    status_t status = STATUS_SUCCESS;

    /* Load hardware key */
    INT_SYS_DisableIRQGlobal();
    status |= FLASH_DRV_LoadAESKey(0, HCU_NVR_START);
    INT_SYS_EnableIRQGlobal();
    HCU_SetKeySize(KEY_SIZE_128_BITS);
    /* HCU ECB encrypt with hardware key */
    status |= HCU_DRV_EncryptECB(plainText, 64, hw_encrypt_result);

    /* Load software key */
    status |= HCU_DRV_LoadUserKey(keyNvr, KEY_SIZE_128_BITS);
    /* Start AES-ECB software encrypt */
    status |= HCU_DRV_EncryptECB(plainText, 64, sw_encrypt_result);
    /* Check result */
    for (uint32_t i = 0; i < 16; i++)
    {
        if (sw_encrypt_result[i] != cipherText[i])
        {
            PRINTF("Software key mismatch!\n");
            return STATUS_ERROR;
        }
        if (hw_encrypt_result[i] != cipherText[i])
        {
            PRINTF("Hardware key mismatch!\n");
            return STATUS_ERROR;
        }
    }
    PRINTF("HCU key load success!\n");
    return status;
}

/* USER CODE END 0 */
```

```c
/**
 * @brief  The application entry point.
 * @retval int
 */
int main(void)
{
    /* USER CODE BEGIN 1 */
    status_t status = STATUS_SUCCESS;
    /* USER CODE END 1 */
    Board_Init();
    /* USER CODE BEGIN 2 */
    PRINTF("Secure boot prepare demo!\n");
    Check_HcuKeyLoad();
    /* USER CODE END 2 */

    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    uint32_t counter = 0;
    while (1)
    {
        if (status != STATUS_SUCCESS)
        {
            break;
        }
        PRINTF("Current counter is %d\n", counter++);
        OSIF_TimeDelay(1000);
        /* USER CODE END WHILE */
        /* USER CODE BEGIN 3 */
    }
    /* USER CODE END 3 */
}

static void Board_Init(void)
{
    CLOCK_SYS_Init(g_clockManConfigsArr,CLOCK_MANAGER_CONFIG_CNT,g_clockManCallbacksArr,CLOCK_MANAGER_CALLBACK_CNT);
    CLOCK_SYS_UpdateConfiguration(CLOCK_MANAGER_ACTIVE_INDEX,CLOCK_MANAGER_POLICY_AGREEMENT);
    PINS_DRV_Init(NUM_OF_CONFIGURED_PINS0,g_pin_mux_InitConfigArr0);
    FLASH_DRV_Init(0,&flash_config0,&flash_config0_State);
    UTILITY_PRINT_Init();
    HCU_DRV_Init(&hcu_config0,&hcu_config0_State);
}

/* USER CODE BEGIN 4 */
/* USER CODE END 4 */
```

3.2.2.2 测试结果（FAIL）

编译完成后，若直接下载至芯片。随后重新上电或复位，无法看见正常打印的内容。这是由于使能了严格安全启动流程，但并未对程序进行签名，故在 Secure Boot 期间验签失败，拒绝跳转至应用程序。

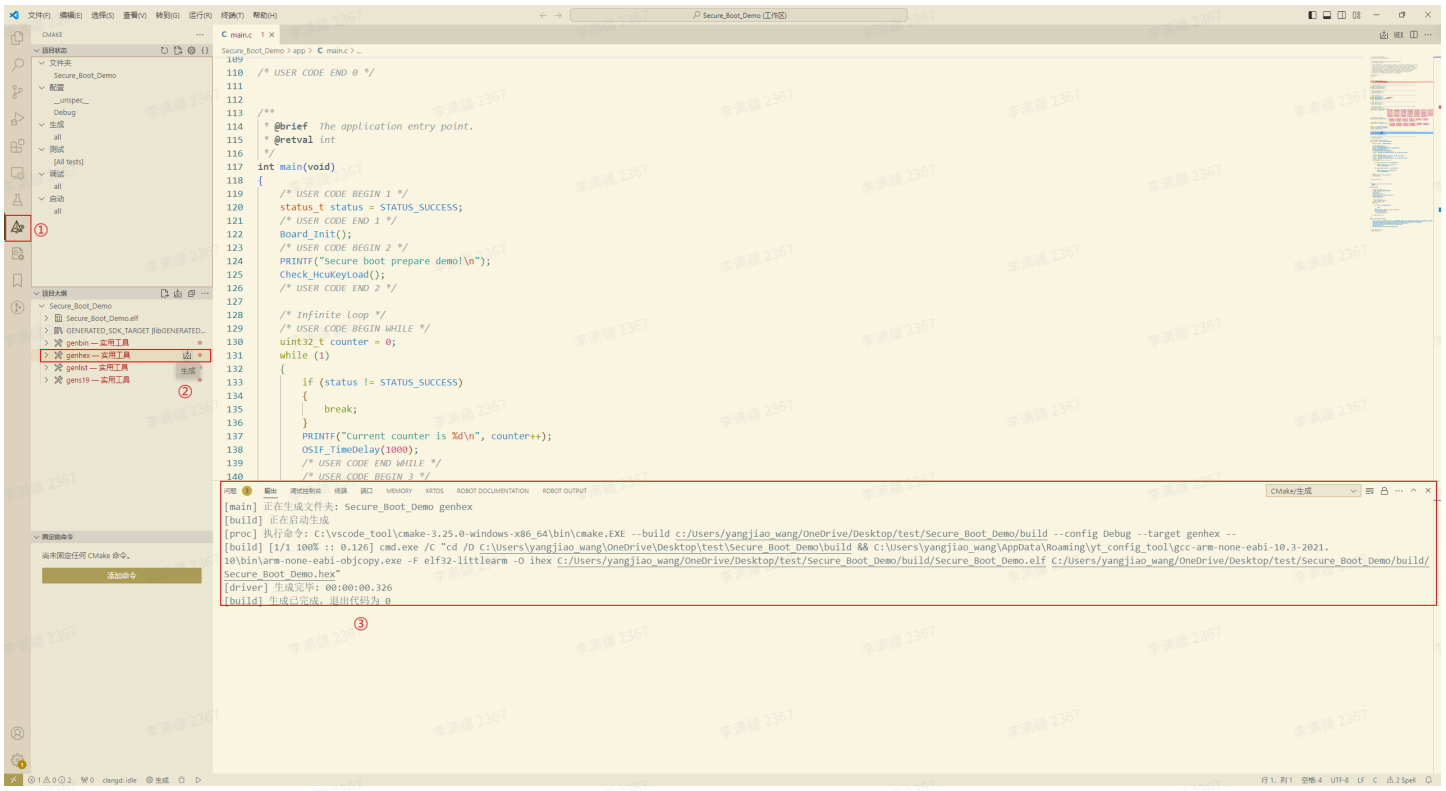
3.2.3 生成签名

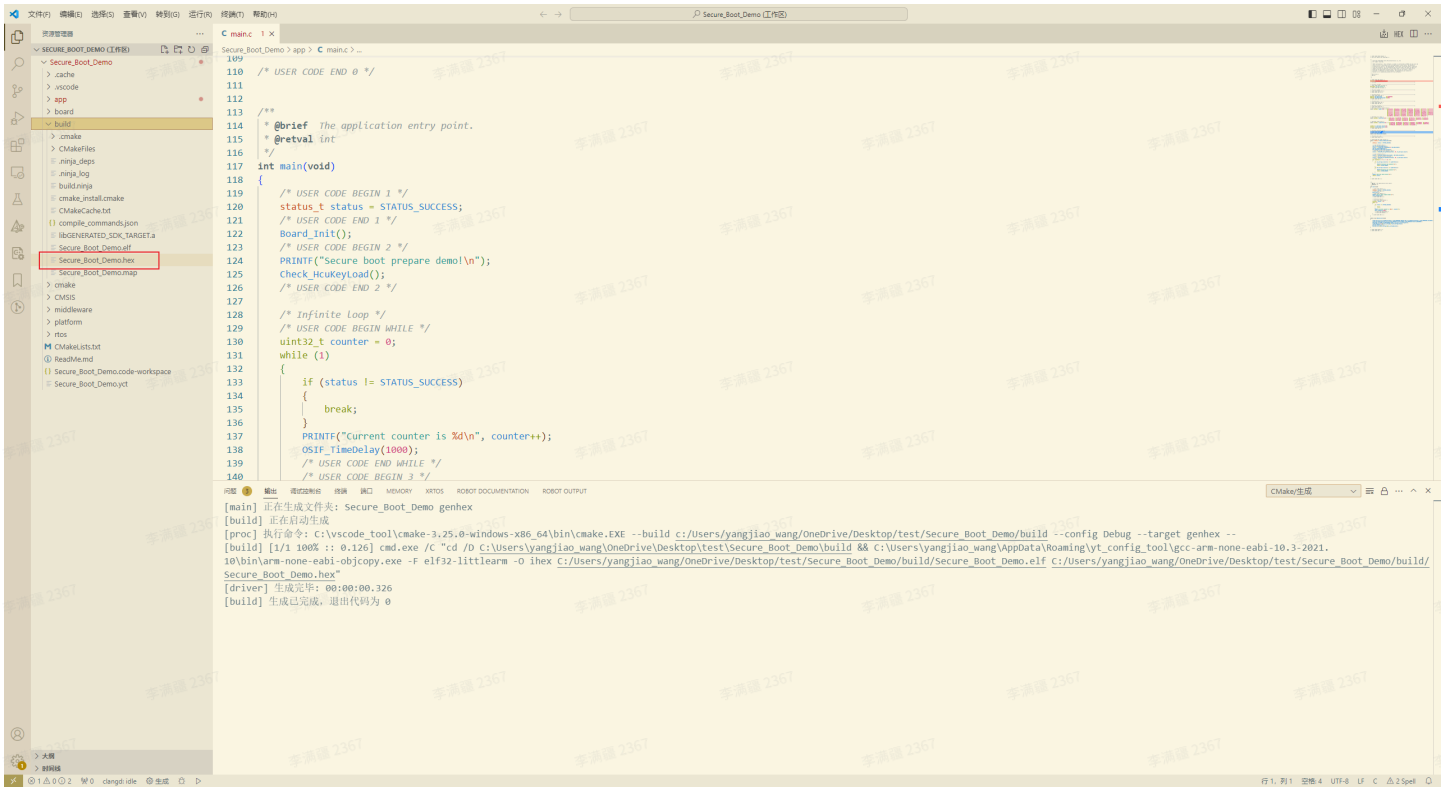
3.2.3.1 生成 Hex

目前 YT Config Tool 仅支持 Hex 文件的验签，故先要生成 Hex 文件。

3.2.3.1.1 CMakeGCC

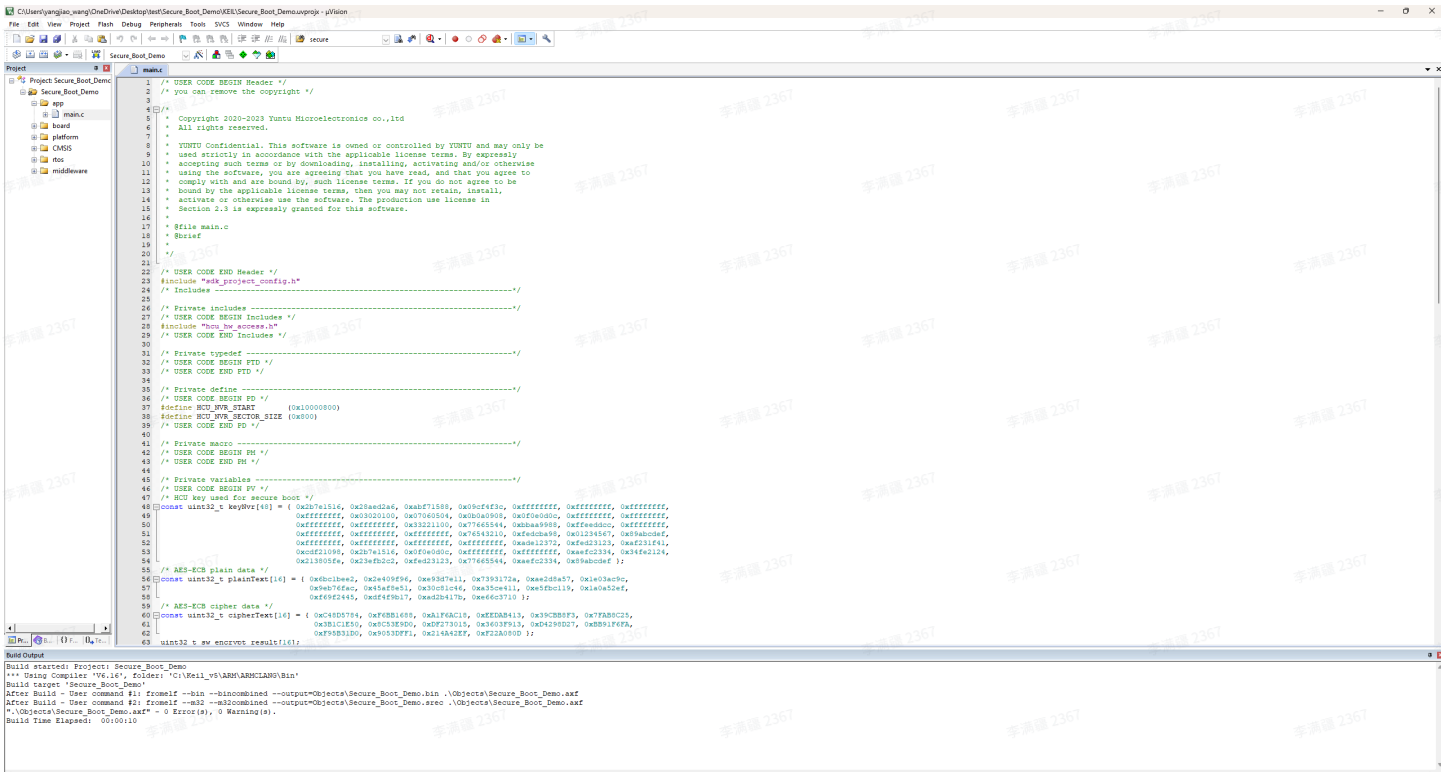
按照如下操作依次点击，即可生成 Secure_Boot_Demo.hex

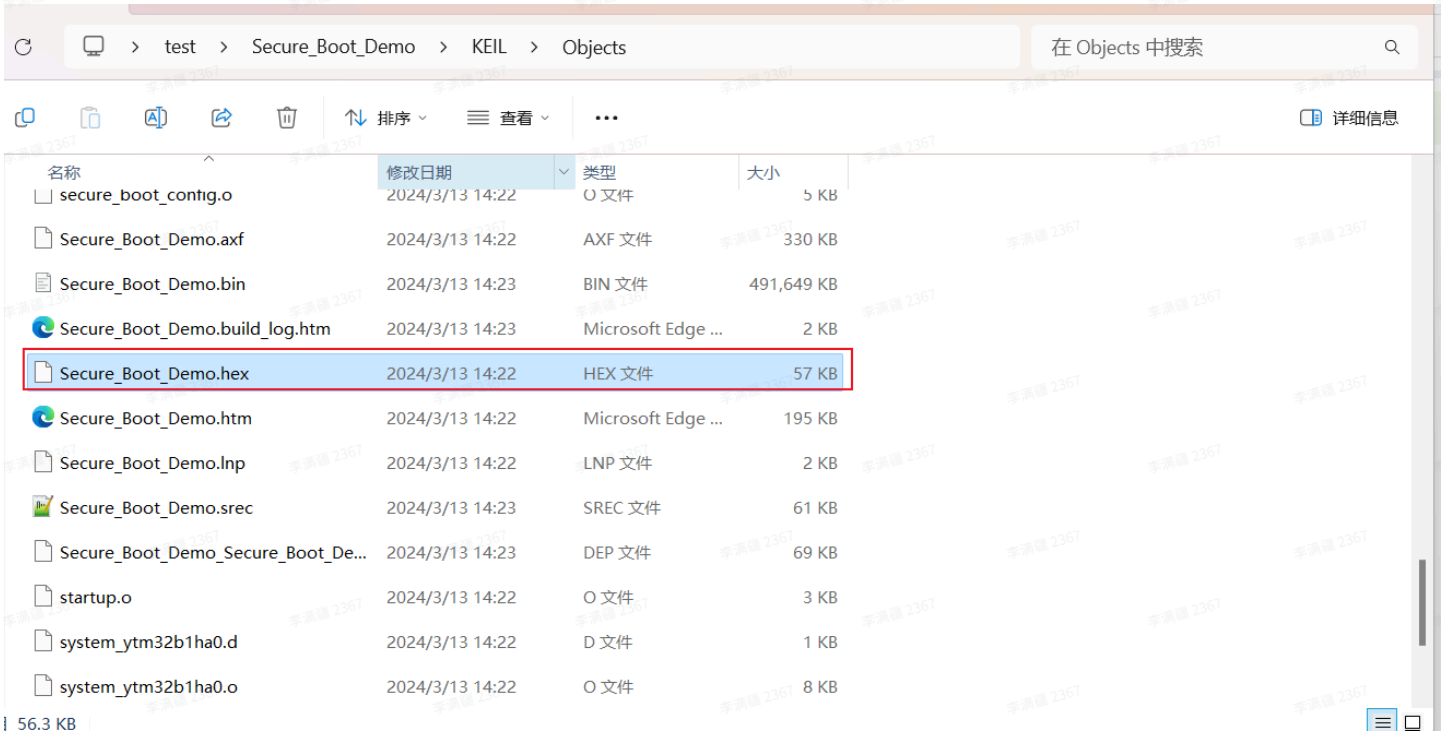




3.2.3.1.2 KEIL

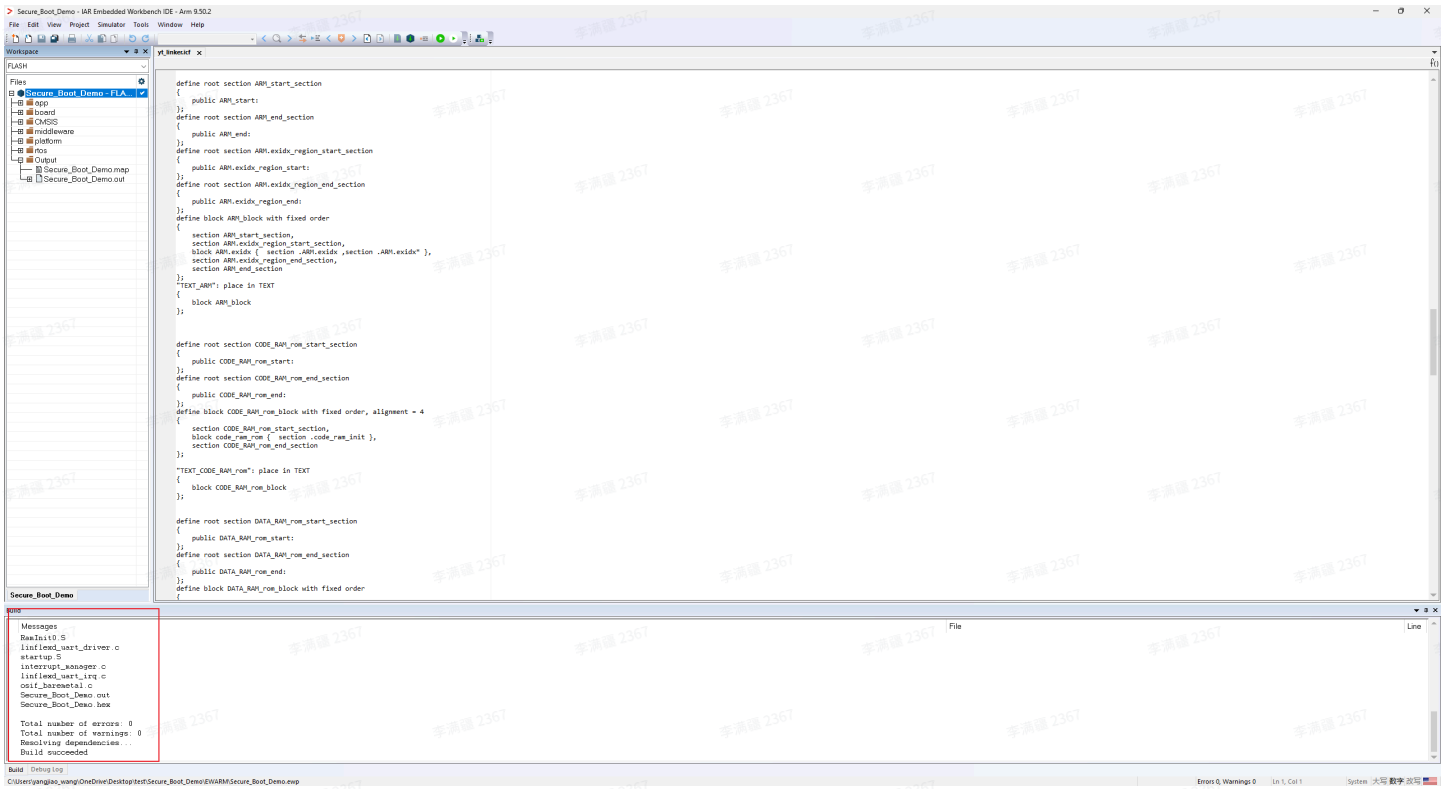
KEIL 工程在编译成功后，可在 Object 目录下找到 Secure_Boot_Demo.hex

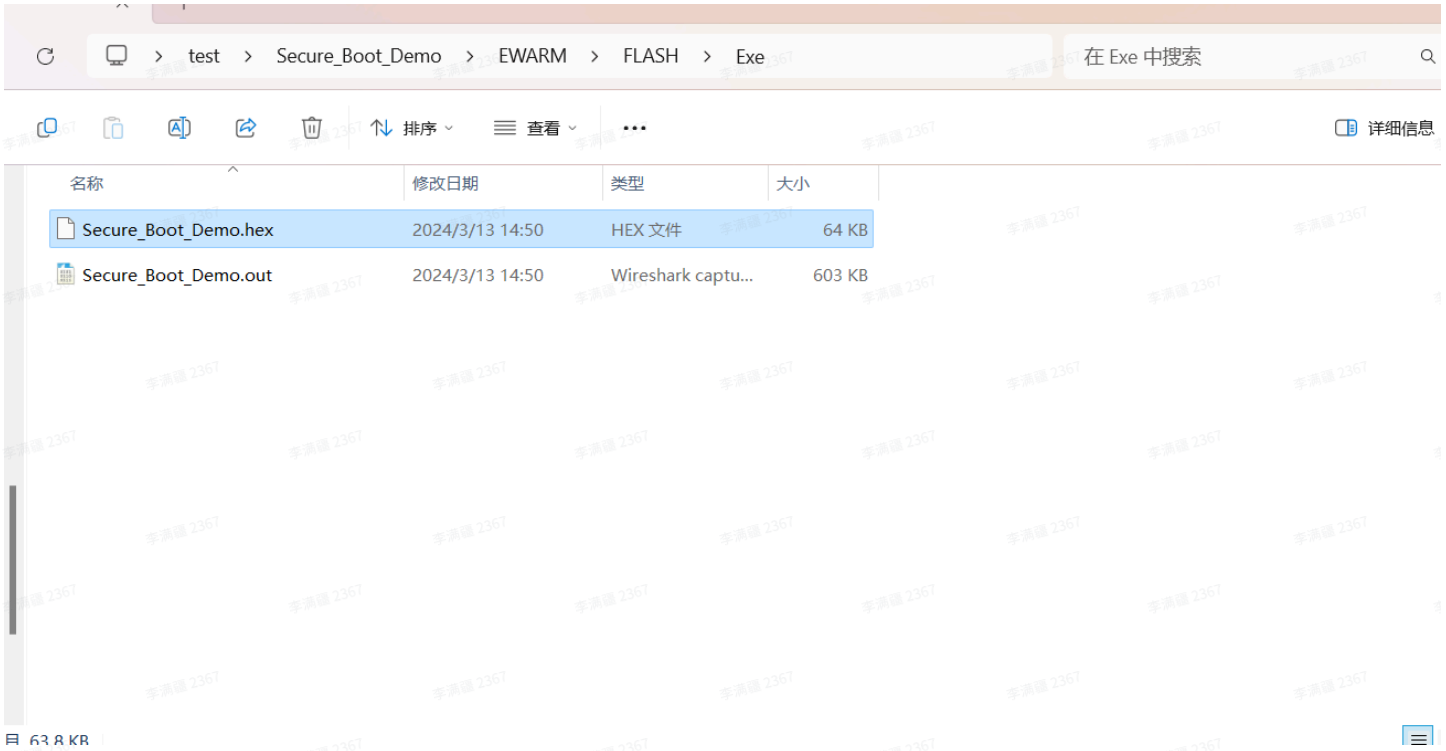




3.2.3.1.3 IAR

IAR 工程在编译完成后，可在 FLASH 目录下找到 Secure_Boot_Demo.hex

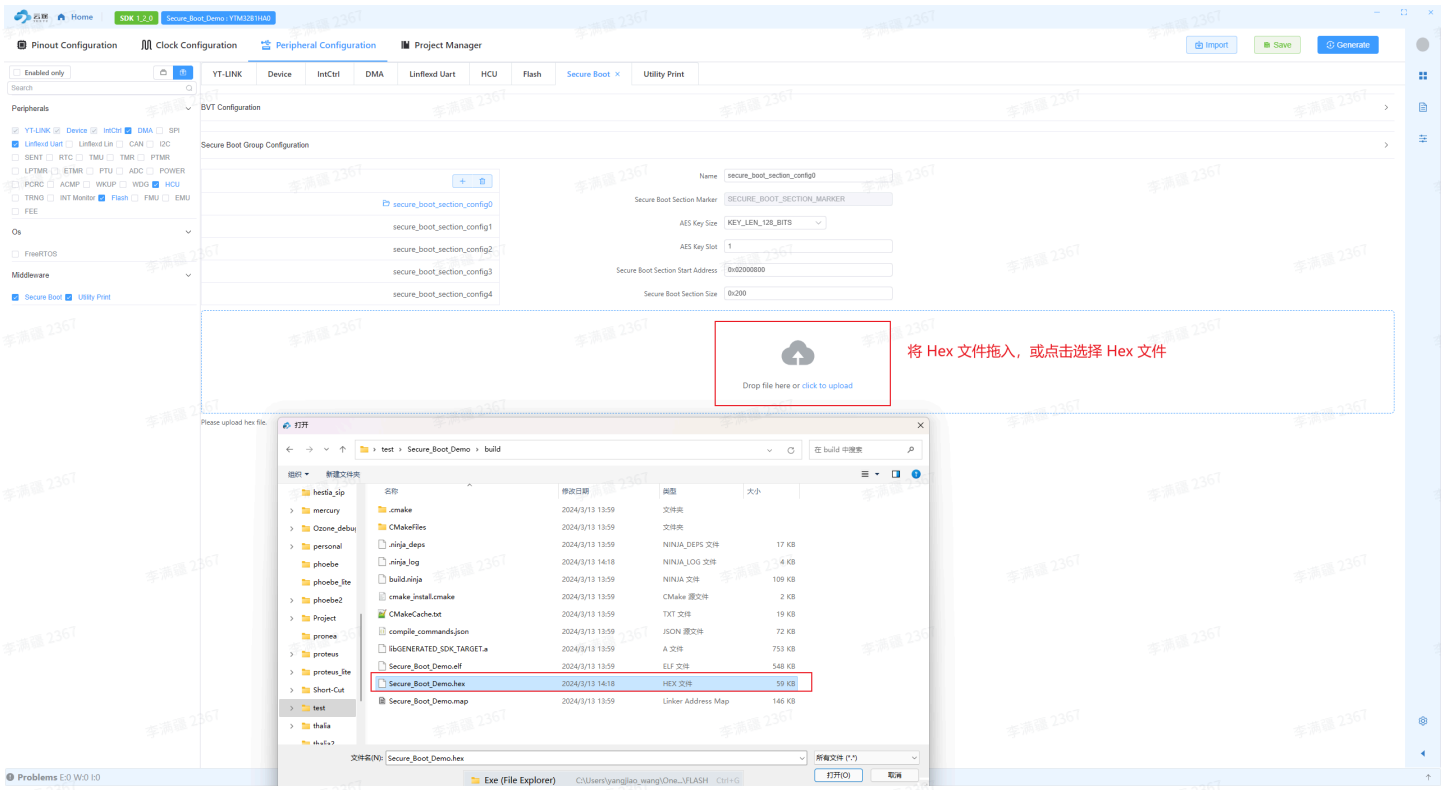




3.2.3.2 加密程序

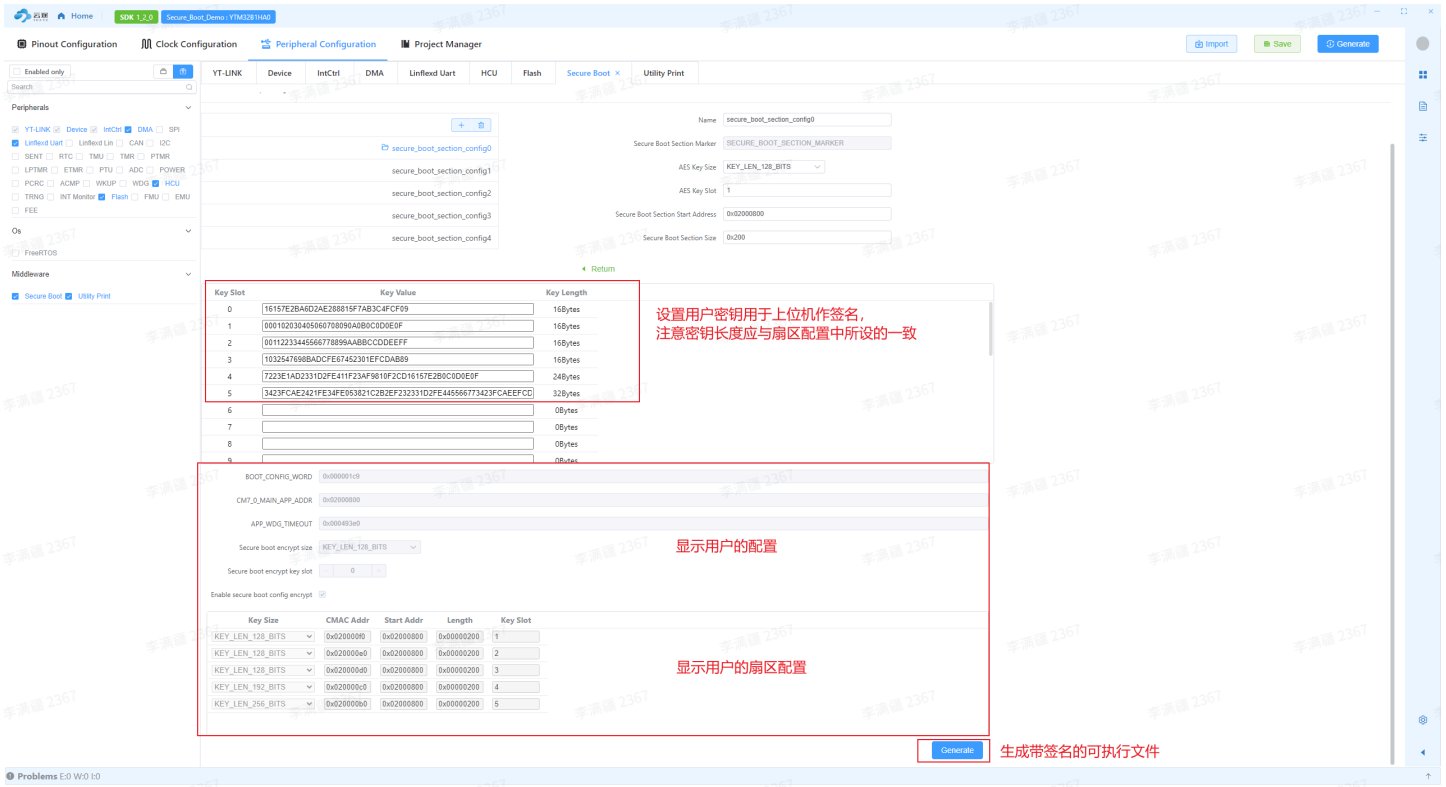
3.2.3.2.1 导入Hex

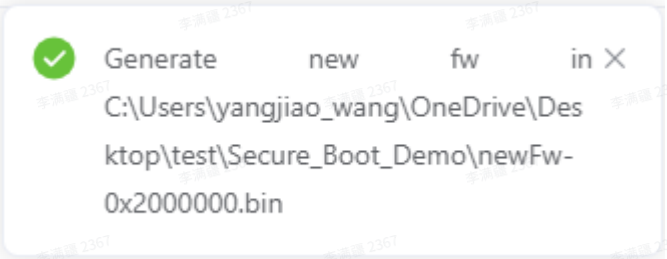
将 Hex 文件导入 YT Config Tool 中，如下操作



3.2.3.2.2 设置密钥

用户需写入自己的密钥用于 YT Config Tool 工具计算签名值，配置完成后点击 Generate 按钮，生成 **newFw-0x2000000.bin**



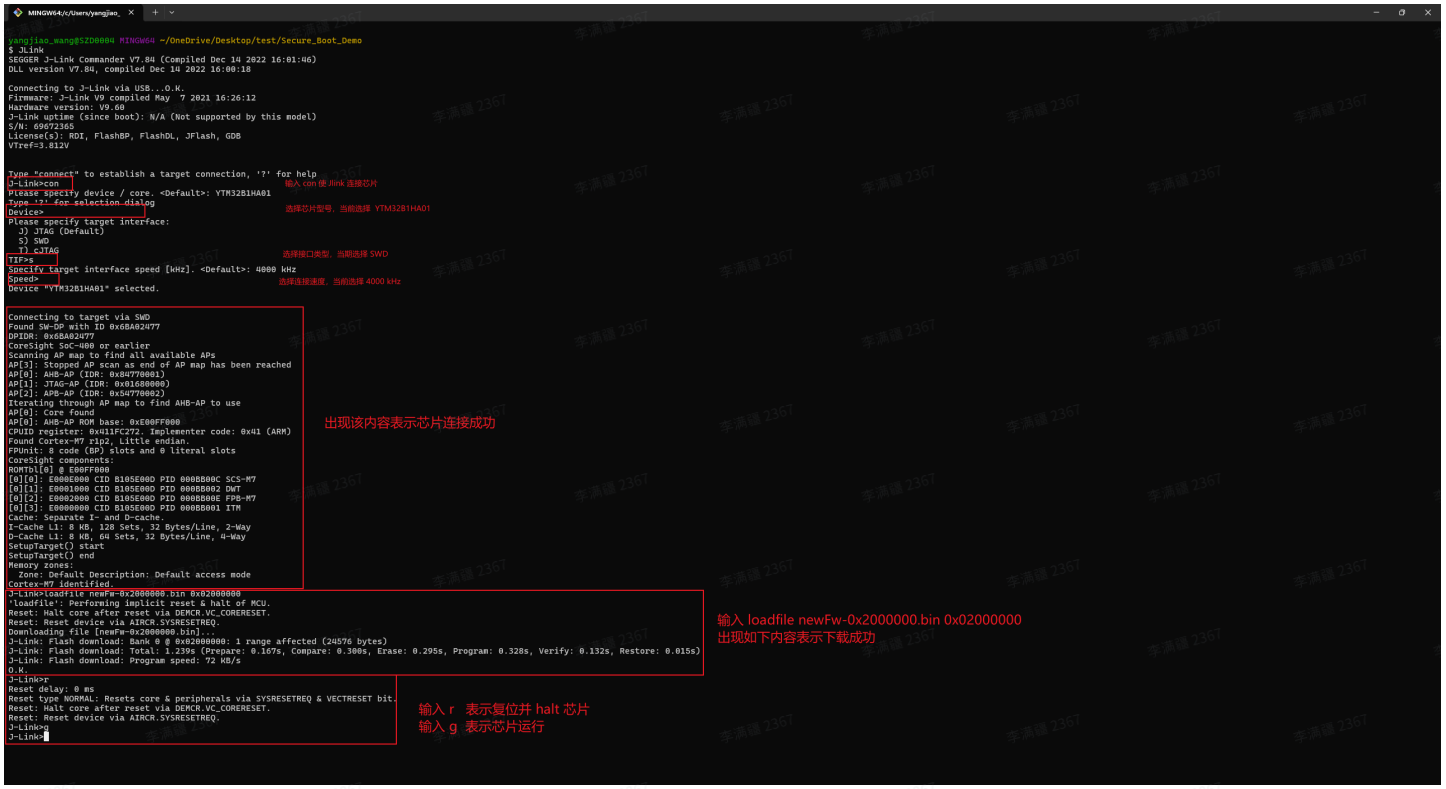


3.2.3.3 下载程序

将带签名的程序下载至芯片内，这里推荐两种方法，J-Link Commander 或 J-Flash

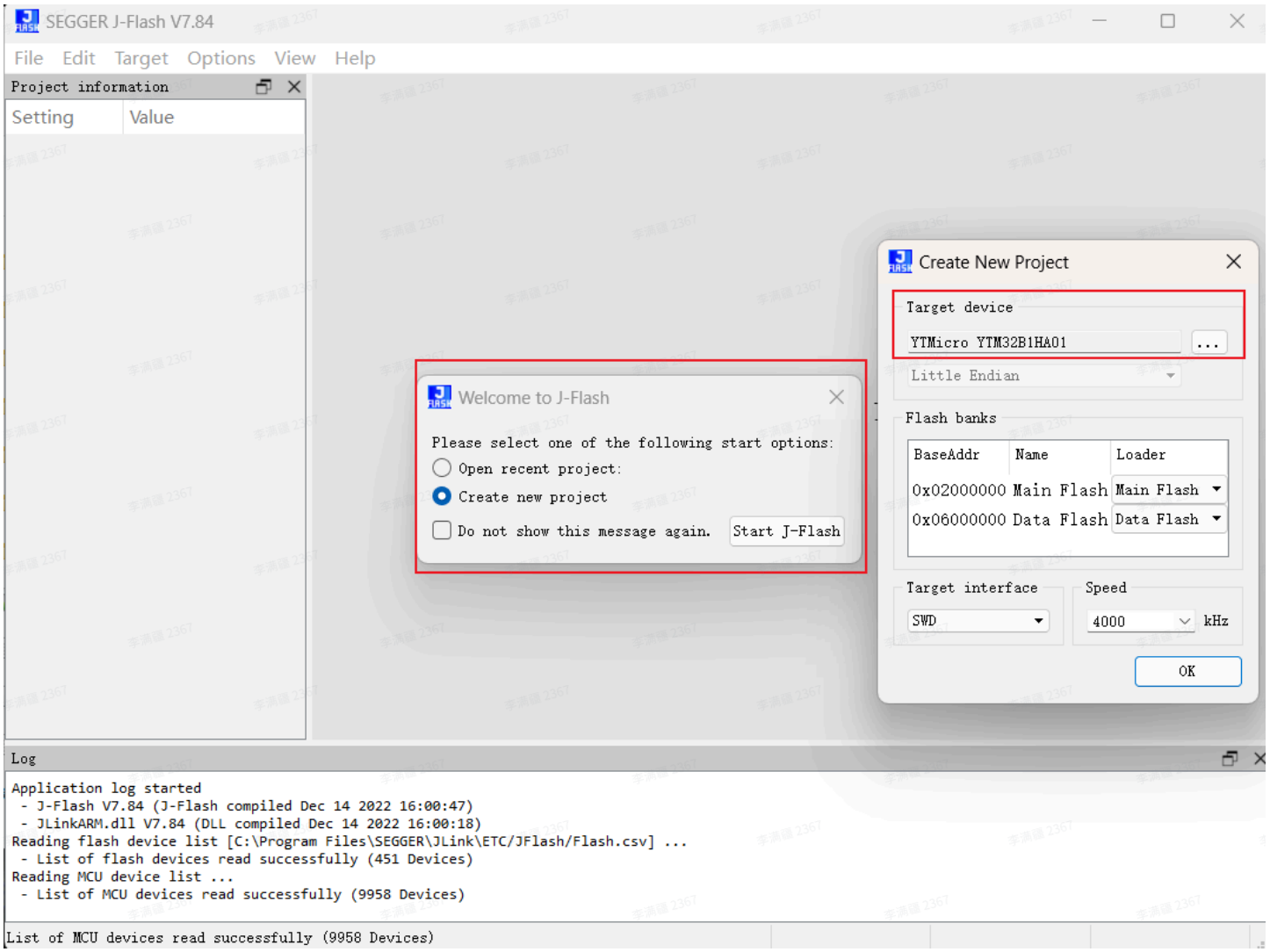
3.2.3.3.1 J-Link Commander

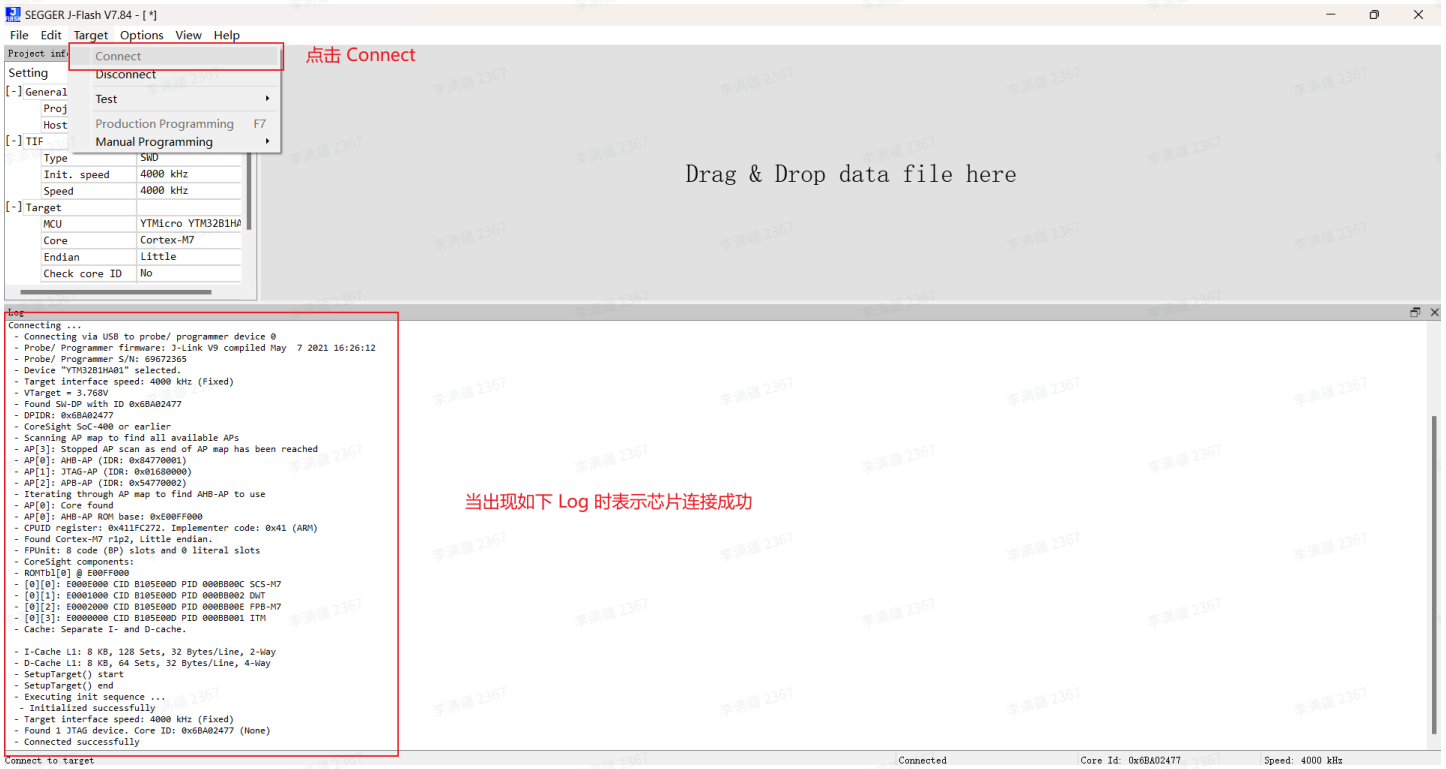
按照下述流程将程序下载至芯片内，下载完成后退出 J-Link Commander

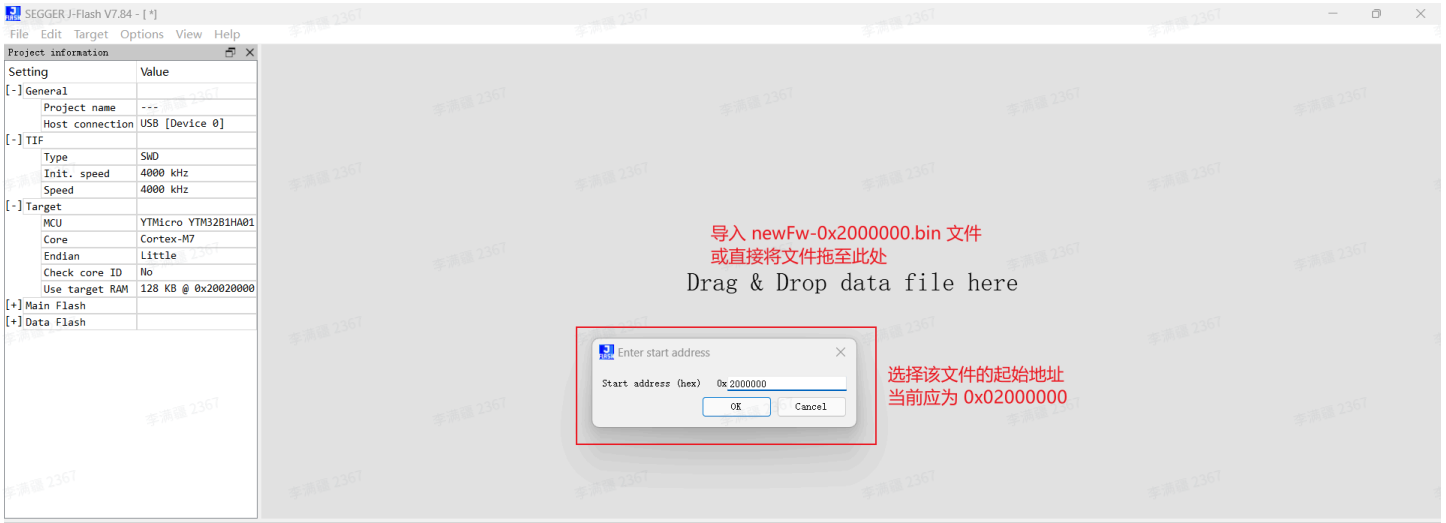


3.2.3.3.2 J-Flash

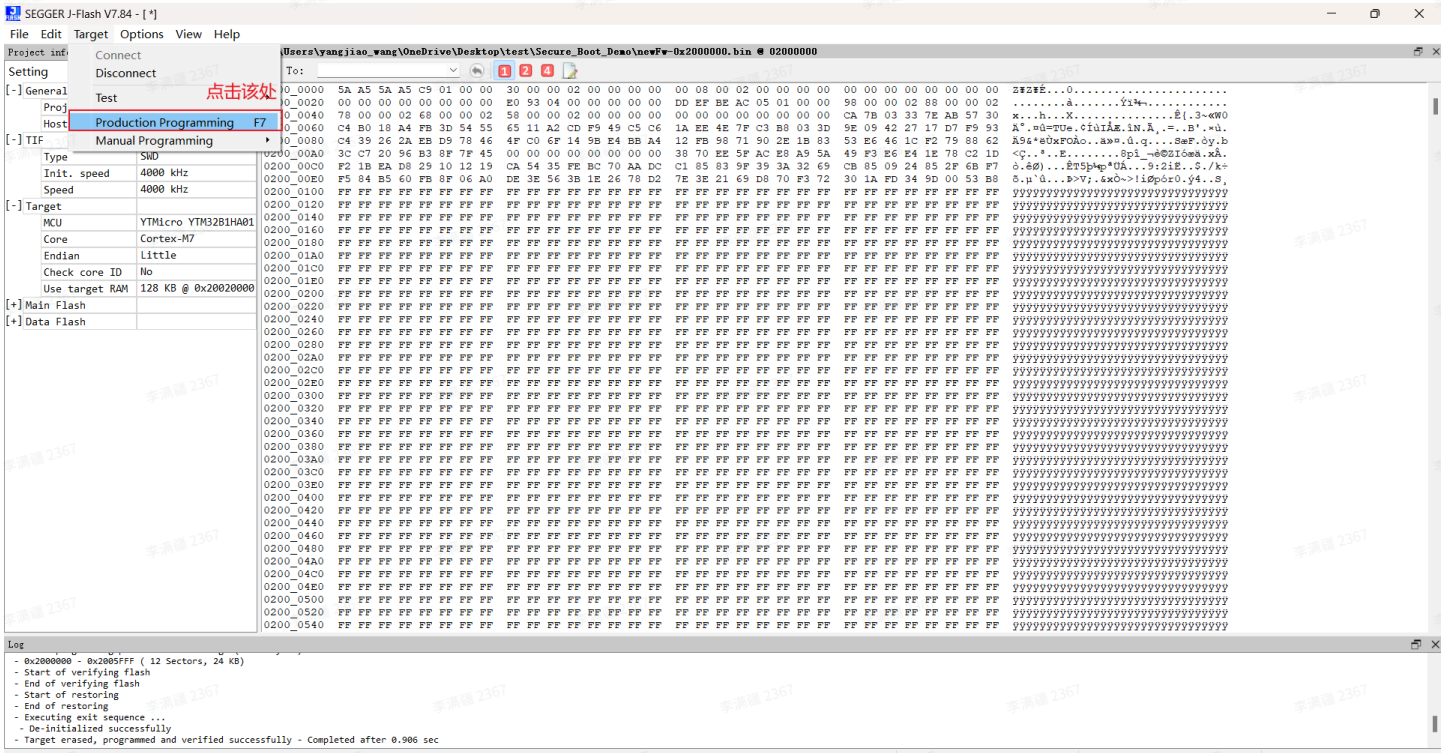
创建一个新的 J-Flash 工程，选择 YTM32B1HA01 芯片

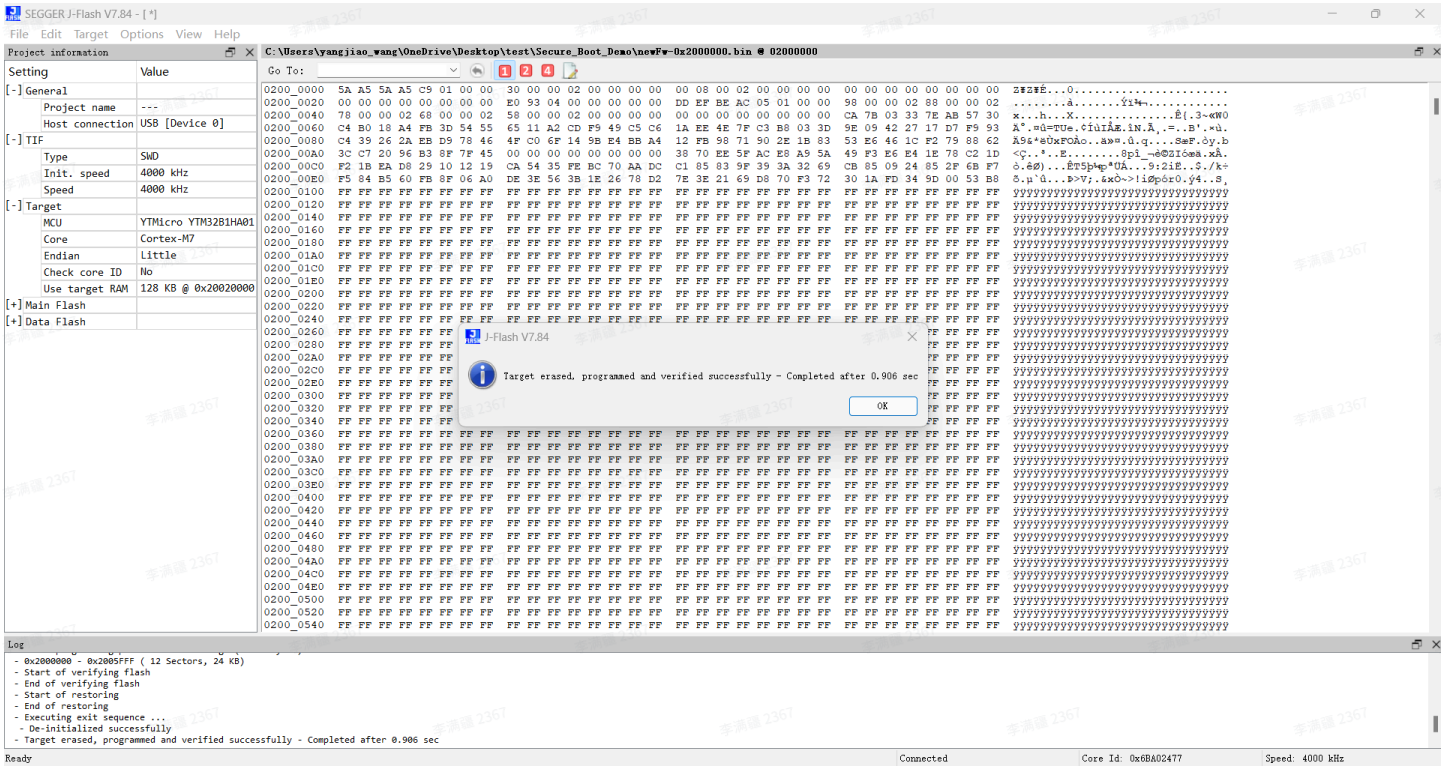

SEGGER J-Flash V7.84
File Edit Target Options View Help
Project information
Setting
Value
Welcome to J-Flash
Please select one of the following start options:
Open recent project:
Create new project
Do not show this message again.
Start J-Flash
Create New Project
Target device
YTMicro YTM32B1HA01
Little Endian
Flash banks
BaseAddr
Name
Loader
0x02000000 Main Flash
Main Flash
0x06000000 Data Flash
Data Flash
Target interface
Speed
SWD
4000
kHz
OK
Log
Application log started
- J-Flash V7.84 (J-Flash compiled Dec 14 2022 16:00:47)
- JLinkARM.dll V7.84 (DLL compiled Dec 14 2022 16:00:18)
Reading flash device list [C:\Program Files\SEGGER\JLink\ETC/JFlash/Flash.csv] ...
- List of flash devices read successfully (451 Devices)
Reading MCU device list ...
- List of MCU devices read successfully (9958 Devices)
List of MCU devices read successfully (9958 Devices)


SEGGER J-Flash V7.84 - [*]
File Edit Target Options View Help
Connect
Disconnect
Test
Production Programming F7
Manual Programming
点击 Connect
Setting
Type SWD
Init. speed 4000 kHz
Speed 4000 kHz
Target
MCU YTMicro YTM32B1HA
Core Cortex-M7
Endian Little
Check core ID No
Drag & Drop data file here
Log
Connecting ...
- Connecting via USB to probe/ programmer device 0
- Probe/ Programmer firmware: J-Link V9 compiled May 7 2021 16:26:12
- Probe/ Programmer S/N: 69672365
- Device "YTM32B1HA01" selected.
- Target interface speed: 4000 kHz (Fixed)
- VTarget = 3.768V
- Found SW-DP with ID 0x6BA02477
- DPIDR: 0x6BA02477
- CoreSight SoC-400 or earlier
- Scanning AP map to find all available APs
- AP[3]: Stopped AP scan as end of AP map has been reached
- AP[0]: AHB-AP (IDR: 0x84770001)
- AP[1]: JTAG-AP (IDR: 0x01680000)
- AP[2]: APB-AP (IDR: 0x54770002)
- Iterating through AP map to find AHB-AP to use
- AP[0]: Core found
- AP[0]: AHB-AP ROM base: 0xE00FF000
- CPUID register: 0x411FC272. Implementer code: 0x41 (ARM)
- Found Cortex-M7 r1p2, Little endian.
- FPUnit: 8 code (BP) slots and 0 literal slots
- CoreSight components:
- ROMTbl[0] @ E00FF000
- [0][0]: E000E000 CID B105E00D PID 000BB00C SCS-M7
- [0][1]: E0001000 CID B105E00D PID 000BB002 DWT
- [0][2]: E0002000 CID B105E00D PID 000BB00E FPB-M7
- [0][3]: E0000000 CID B105E00D PID 000BB001 ITM
- Cache: Separate I- and D-cache.
- I-Cache L1: 8 KB, 128 Sets, 32 Bytes/Line, 2-Way
- D-Cache L1: 8 KB, 64 Sets, 32 Bytes/Line, 4-Way
- SetupTarget() start
- SetupTarget() end
- Executing init sequence ...
- Initialized successfully
- Target interface speed: 4000 kHz (Fixed)
- Found 1 JTAG device. Core ID: 0x6BA02477 (None)
- Connected successfully
当出现如下 Log 时表示芯片连接成功
Connect to target
Connected
Core Id: 0x6BA02477
Speed: 4000 kHz


SEGGER J-Flash V7.84 - [*]
File Edit Target Options View Help
Project information
Setting
Value
[-] General
Project name

Host connection
USB [Device 0]
[-] TIF
Type
SWD
Init. speed
4000 kHz
Speed
4000 kHz
[-] Target
MCU
YTMicro YTM32B1HA01
Core
Cortex-M7
Endian
Little
Check core ID
No
Use target RAM
128 KB @ 0x20020000
[+] Main Flash
[+] Data Flash
导入 newFw-0x2000000.bin 文件
或直接将文件拖至此处
Drag & Drop data file here
Enter start address
Start address (hex) 0x2000000
OK
Cancel
选择该文件的起始地址
当前应为 0x02000000
Log
- [0][1]: E0001000 CID B105E00D PID 000BB002 DWT
- [0][2]: E0002000 CID B105E00D PID 000BB00E FPB-M7
- [0][3]: E0000000 CID B105E00D PID 000BB001 ITM
- Cache: Separate I- and D-cache.
- I-Cache L1: 8 KB, 128 Sets, 32 Bytes/Line, 2-Way
- D-Cache L1: 8 KB, 64 Sets, 32 Bytes/Line, 4-Way
- SetupTarget() start
- SetupTarget() end
- Executing init sequence ...
- Initialized successfully
- Target interface speed: 4000 kHz (Fixed)
- Found 1 JTAG device. Core ID: 0x6BA02477 (None)
- Connected successfully
Opening data file [C:\Users\yangjiao_wang\OneDrive\Desktop\test\Secure_Boot_Demo\newFw-0x2000000.bin] ...
- Failed to open data file.
No start address given for C:\Users\yangjiao_wang\OneDrive\Desktop\test\Secure_Boot_Demo\newFw-0x2000000.bin
Opening data file [C:\Users\yangjiao_wang\OneDrive\Desktop\test\Secure_Boot_Demo\newFw-0x2000000.bin] ...
Ready
Connected
Core Id: 0x6BA02477
Speed: 4000 kHz

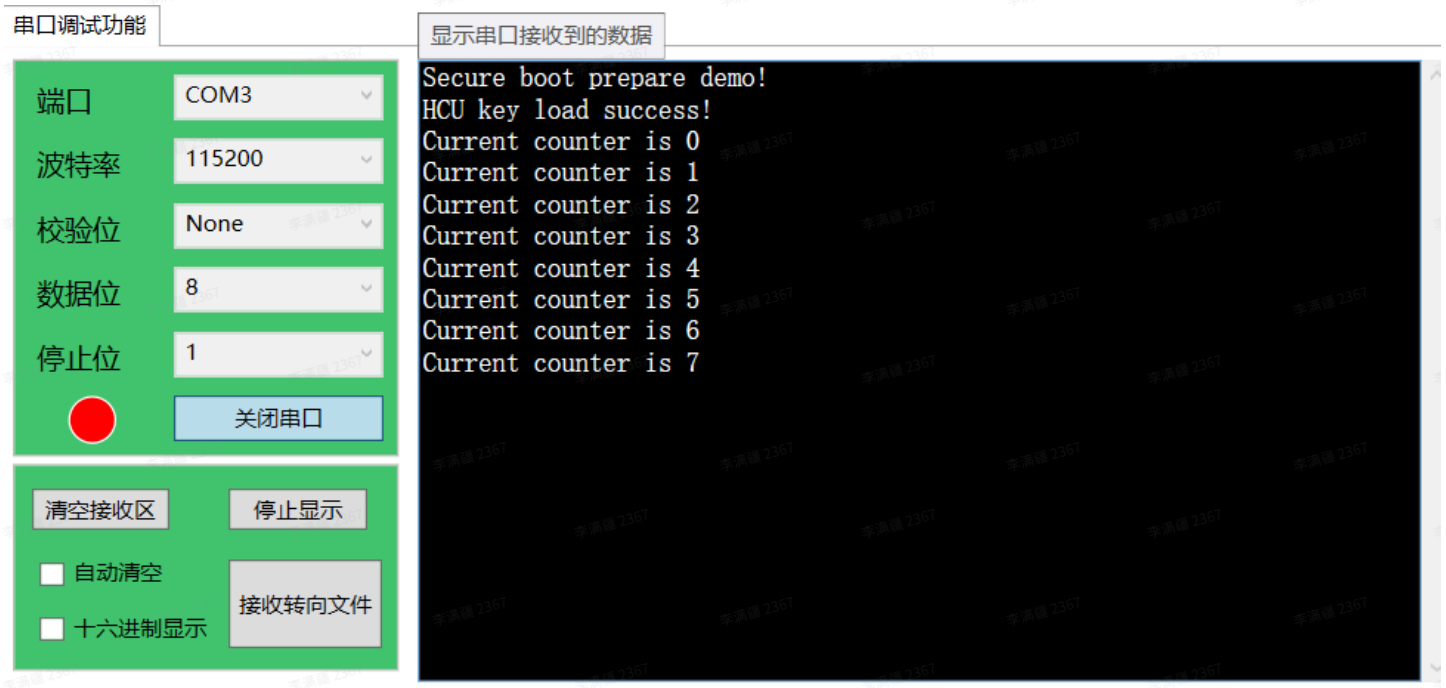

SEGGER J-Flash V7.84 - [*]
File Edit Target Options View Help
Connect
Disconnect
Test
点击该处
Production Programming F7
Manual Programming
Setting
Value
[-] General
[-] TIF
Type
SWD
Init. speed
4000 kHz
Speed
4000 kHz
[-] Target
MCU
YTMicro YTM32B1HA01
Core
Cortex-M7
Endian
Little
Check core ID
No
Use target RAM
128 KB @ 0x20020000
[+] Main Flash
[+] Data Flash
\Users\yangjiao_wang\OneDrive\Desktop\test\Secure_Boot_Demo\newFw-0x2000000.bin @ 02000000
To:
Log
- 0x2000000 - 0x2005FFF (12 Sectors, 24 KB)
- Start of verifying flash
- End of verifying flash
- Start of restoring
- End of restoring
- Executing exit sequence ...
- De-initialized successfully
- Target erased, programmed and verified successfully - Completed after 0.906 sec
Erase, program and verify target
Connected
Core Id: 0x6BA02477
Speed: 4000 kHz



下载完成后，关闭 J-Flash。

3.2.4 测试结果（Pass）

重新上电或复位后，看到串口打印如下内容，代表 Secure Boot 验签成功，并成功跳转至应用程序。



3.3 Secure Boot Address Protect (软件 Secure Boot 特供)

3.3.1 工程配置

工程配置与 Secure Boot Demo 流程一致，故不再赘述。

3.3.2 Secure Boot 软件保护

由于是软件 Secure Boot，存放在 Flash 中，不是真实的 ROM ，故无法保护 Secure Boot 区域被篡改或修改。

在 Secure Boot 1.1.0 版本中更新了软件 Secure Boot 固件，当用户下载 Secure Boot 固件并运行后，用户将无法擦除或修改存放 Secure Boot 所在区域(0x0000~0x4000)。

下面提出两个方案来提高软件 Secure Boot 的安全等级。

3.3.2.1 JTAG 连接保护

用户向 Customer NVR 指定区域写入指定值，可禁止 JTAG (SWD) 连接芯片，擦除该区域可以恢复 JTAG(SWD) 连接。

在 *Secure Boot Demo* 中有提供操作代码，用户按照正确流程将程序下载至 App 区域后，摁下 SW2 按键可***禁止*** JTAG (SWD) 连接芯片，摁下 SW3 按键可***恢复*** JTAG(SWD) 连接。

下面是按下 SW3 恢复 JTAG(SWD) 连接后，芯片打印的 log 信息。

代码块

```
Secure boot demo!
HCU key load success!
Red LED means LOCK, and green LED means UNLOCK!
Press SW2 to disable SWD and press SW3 to enable SWD!
Current counter is 0
Current counter is 1
Current counter is 2
Current counter is 3
Current counter is 4
Enable SWD and unlock chip!
```

3.3.2.2 地址保护

用户向 Customer NVR 指定区域写入指定值，可以保护部分区域，即禁止用户去擦除或编程待保护的Flash。

在 *Secure Boot Address Protect Demo* 中有提供操作代码，用户按照正确流程将程序下载至 App 区域后，摁下 SW2 按键可***保护*** Secure Boot (0x0000~0x4000) 区域被修改，摁下 SW3 按键可***解除*** Secure Boot (0x0000~0x4000) 区域的保护。

下面是按下 SW3 解除保护后，芯片打印的 log 信息。

代码块

```
Secure boot address protect demo!
HCU key load success!
Red LED means protected, and green LED means not protected!
```

```
Press SW2 to enable address protect!
Press SW3 to disable address protect!
Current counter is 0
Current counter is 1
Current counter is 2
Current counter is 3
Current counter is 4
Disable Chip address protect!
```

4. 注意事项总结

4.1 密钥编程

不同芯片对 HCU_NVR 的处理方式不同，例如 YTM32B1MD14 **不可擦除** HCU_NVR，其余几款芯片可以擦除 HCU_NVR，但是需要 **Customer Key**，Customers Key 默认值可在 Reference Manual 中查看，用户可自行修改 Customers Key，只有用户自己知道。HCU_NVR 编程没有限制，只需要保证要写入的 HCU_NVR 为空即可。详情可查看 **AN_0062_EFM_ApplicationNote.pdf**

4.2 密钥长度

不同芯片支持的密钥长度不同，YTM32B1ME05 与 YTM32B1HA01 支持 128/192/256-bit 密钥，YTM32B1MD14 与 YTM32B1MC03 支持 128-bit 密钥，故在 Secure Boot 配置中需注意密钥长度。且下载密钥时也应注意密钥长度。详情可查看 **AN_0063_HCU_ApplicationNote.pdf**

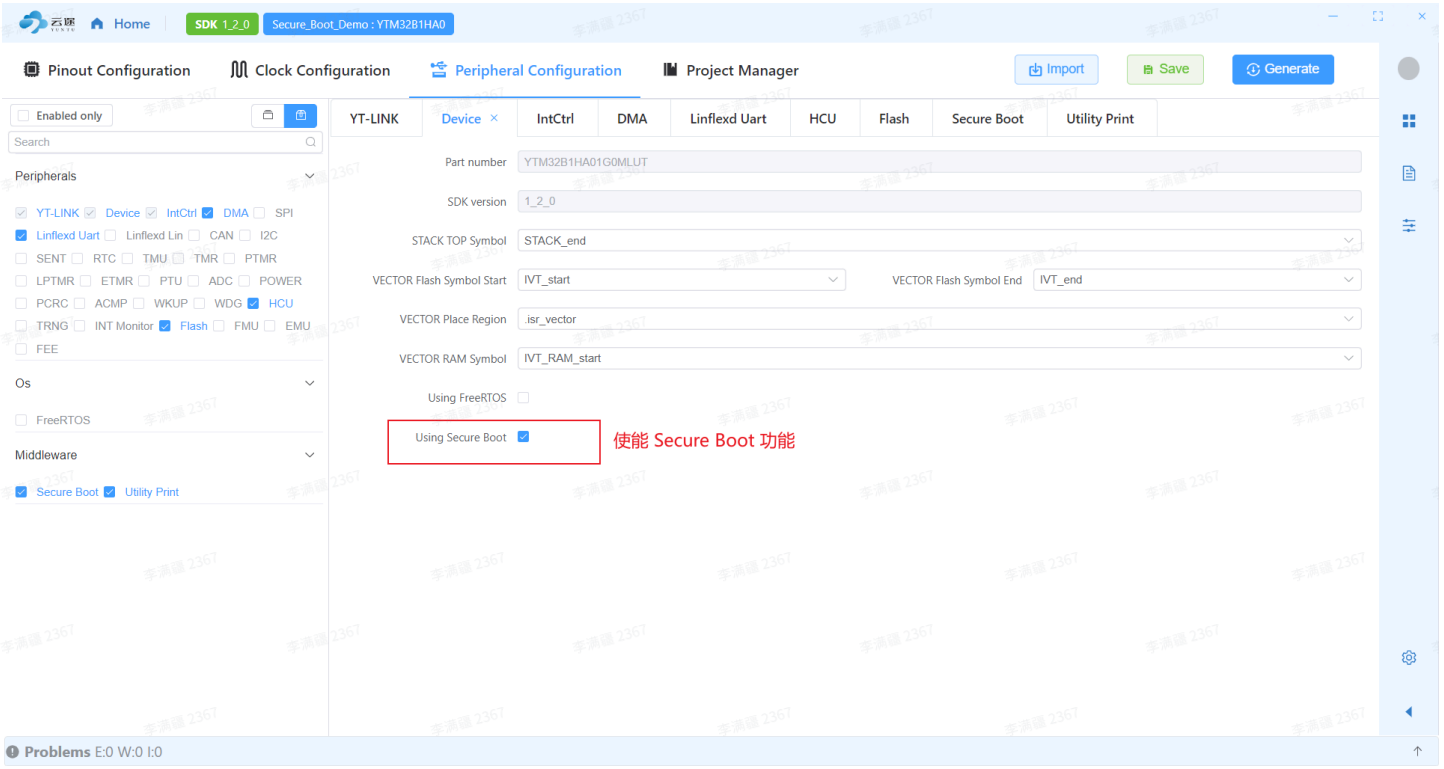
4.3 YT-LINK

Secure Boot 由于需要 BVT，Secure Boot Group，Secure Boot Section 与 CMAC，**Link File** 需要作修改。现通过 YT-LINK 增加四个 Region 用于存储 Secure Boot 相关配置信息。YT-LINK 的使用可参考 YT-LINK User Manual。

YTM32B1HA01 默认的 YT-LINK 便分配了 Secure Boot 相关配置区域，其余芯片暂无。故用户若需要修改 Secure Boot 配置信息位置，应自行重新分配。

4.4 Device

若需要使用 Secure Boot ，需勾选 Device 模块下的 Using Secure Boot 框。



4.5 扇区保护

所保护的扇区**不可包含可变区域**。因为若该区域内容发生变化，则产生的签名势必不一致，导致验签失败。例如用户程序中会反复擦写的区域（可变数据存储区），例如扇区配置信息区（扇区配置信息可能会被加密存储，导致与原始数据不一致），例如签名值存放区域

另外所保护的扇区也不能包含不可读区域。

目前的 YT Config Tool 关于 Secure Boot 仅有基础功能，只能根据已有的 Hex 文件对程序作签名，对 Hex 文件内不包含的区域无法作签名，需要用户自行签名。

4.6 软件 Secure Boot

YTM32B1ME05 与 YTM32B1MD14 是软件 Secure Boot。~~在 Secure Boot 1.0.0 中，目前是在生成签名时（章节 3.2.3.2），附上 Secure Boot 固件，将 Secure Boot 固件与应用程序合并成一个文件，下载至芯片内，实现软件 Secure Boot。~~

在 Secure Boot 1.1.0 版本中，对 Secure Boot 固件进行升级，目前请按照如下流程进行 Demo 运行。

1. 按照 3.1 流程，下载 Secure Boot Prepare 的程序，来加载硬件 HCU 的密钥；
2. 下载 Secure Boot 的固件至芯片内，Secure Boot 固件在 **board** 目录下（*YTM32B1ME05_Secure_Boot.hex* 或 *YTM32B1MD14_Secure_Boot.hex*），该固件在 Secure Boot Demo 生成工程后的 **board** 目录下。
3. 按照 3.2 流程，下载 Secure Boot Demo 的程序，即可实现软件 Secure Boot。

注意：由于Secure Boot 固件内会**自动保护** Secure Boot 区域(0x0000~0x4000)，故必须先执行第一步，将密钥编程后再加载 Secure Boot 固件。