AN12352

LPC54S0xx 在安全启动下的 XIP（Execute In Place）

版本 2 — 2020 年 9 月 18 日 应用笔记

内容

1 简介

安全启动是 LPC54S0xx 部件的重要功能。

安全启动可以确保未经授权的镜像（代码）不会在给定的产品上执行。ROM 中的安全启动加载程序代码是不可变的，用来构成 Root of Trust。当启用安全启动后，boot ROM 检查加载到片上 RAM 中的用户可执行镜像，以确定代码的真实性。如果代码是真实的，则进行控制权的转移。这个过程建立了从 ROM 到用户启动代码的可信代码链。

ROM 中的安全启动加载程序将用户代码加载到片上 RAM 中，并在验证或解密后在 RAM 中执行。当启用安全启动时，镜像大小即代码大小+RO 大小+RW 大小应小于其中一个 RAM 块，SRAMX 或 SRAM0。最大可启动大小，即代码大小+RO 大小+RW 大小为 192 KB。

图 1 显示了启动过程。

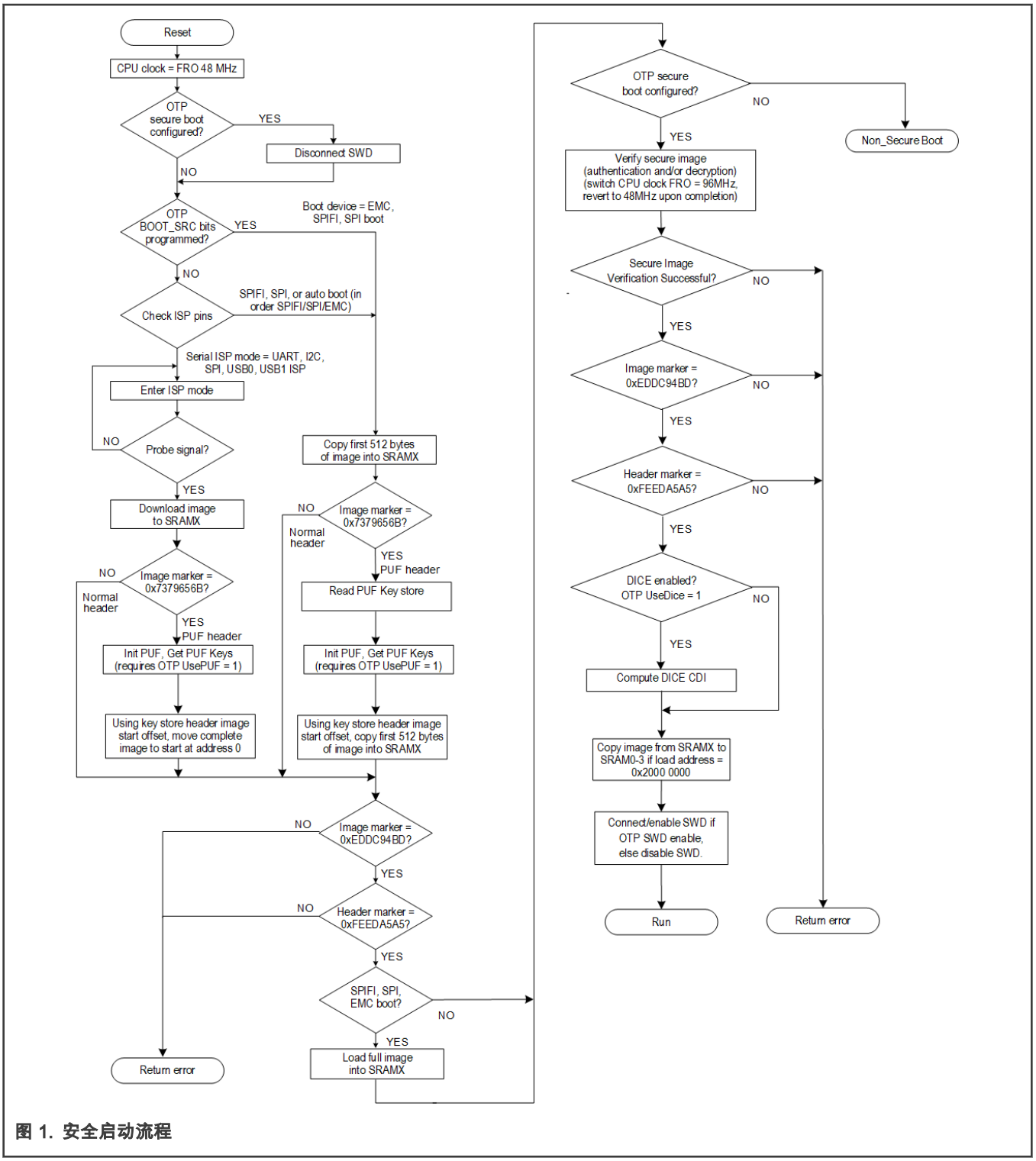


图 1. 安全启动流程

启用安全启动时存在大小限制和额外的代码限制。代码不是直接在 QSPI Flash 执行的（XIP）。为了解决上述限制，本应用文档介绍了一个简单的 demo。这个 demo 演示了如何将镜像分为可启动部分和 XIP 部分。可启动部分包含安全的可启动代码，而 XIP 部分包含纯文本代码。安全可启动部分可用于通过镜像加密和/或身份验证来保护核心代码。

注意

在本文档中，需要启用安全启动才能进行安全启动，这是通过在 OTP 中配置 secure boot type 字段来完成的。例如，本文档将其配置为强制加密，即将 OTP_SECURE_BOOT_TYPE 字段设置为 **b'10**。

修改 OTP 是一次性操作并且不可逆转。因此，在写入 OTP 安全启动类型字段和其他相关字段之前必须小心。

1.1 术语

表 1 列出了后续应用笔记部分中使用的术语。

表 1. 术语

术语	说明
安全可启动镜像	可启动镜像执行了加密或签名等操作。此外，它还满足安全启动类型的要求。
非安全（NS）镜像	纯文本镜像。
Flashloader	Flashloader 是加载到 LPC54S0xx 片上 RAM 中的辅助启动程序，支持 blhost。该工程位于 SDK 中，作为启动加载程序演示。
DFU Utility	DFU Utility 是主机应用程序，用于将 Flashloader 二进制文件加载到 LPC540xx 设备的内部 RAM 内存中，该设备以 USB DFU 模式连接到主机。dfu-util.exe 是一个开源的命令行应用程序。要下载该工具，请参阅 dfu util。
blhost	PC 命令行接口（CLI）工具，实现了 MCUBOOT 协议，它是 MCUBOOT 软件包的一部分。这个 blhost.exe 是一个主机示例程序，与运行 Flashloader 程序的 LPC54S0xx 进行交互。此工具可从 MCUBOOT 下载。
HxD	HxD 是一个二进制文件编辑器。它易于使用，HxD 对于私人和商业用途是免费的。
elftosb	elftosb 工具创建二进制输出文件，其中包含用户应用程序镜像以及一系列启动加载程序命令。输出文件称为安全二进制文件，或简称 SB 文件。这些文件的扩展名为*.sb。该工具使用输入命令文件来控制输出文件中存在的启动加载程序命令的顺序。这个命令文件简称为启动描述符文件或 BD 文件。这个工具可以从 MCUBOOT 下载。
elftosb-gui	elftosb-gui 是一个 GUI 工具，主要用于帮助用户准备安全的应用程序镜像，以及针对目标 MCU 平台的其他有用的安全操作。elftosb-gui 工具在 Elftosb 和 blhost 命令行应用程序之上提供直观的图形界面，并指导用户准备 ROM bootloader 所需的安全启动镜像。这个工具可以从 MCUBOOT 下载。

2 实现

本节介绍如何将代码分为两部分。

- 安全可启动部分（最高 192 kB）
  - 包含可能对性能敏感或不敏感的机密代码（向量表、时间受限的关键算法等）
  - 根据基于安全启动类型的安全镜像格式进行加密或签名。

    ROM 中的安全启动加载程序将这个安全启动部分加载到 RAM 中，并在验证成功后执行它。由于安全启动加载程序禁用 XIP，因此需要初始化 SPIFI 以启用 XIP。
- 非安全部件（XIP）
  - 包含非机密代码。
  - XIP 代码和加载到 RAM 中的代码以纯文本格式放在 flash 中。

2.1 概述

创建分离的镜像需要以下步骤。

1. 通过修改链接器脚本将镜像分成两部分。

   通过链接器脚本将镜像分为安全可启动部分和非安全部分。分区有助于将标识为受保护的代码放在安全可启动部分，而将不受保护的代码放在非安全部分。

2. 创建镜像。
   - 编码后，编译代码。二进制文件是基于链接器脚本生成的。
   - 使用工具将镜像分为两部分：安全可启动部分和非安全部分，然后进行处理。
3. 将镜像的两部分编程到闪存中。

   使用 MCUXpresso 和 CMSIS-DAP 对镜像进行编程。
4. 将 128 位 AES 密钥编程到 OTP。
5. 烧写相关的 OTP 位字段以启用安全启动。
   - 安全启动类型。
   - 安全启动启用。

2.2 拆分二进制镜像

图 2 显示了一个特殊镜像布局的示例。

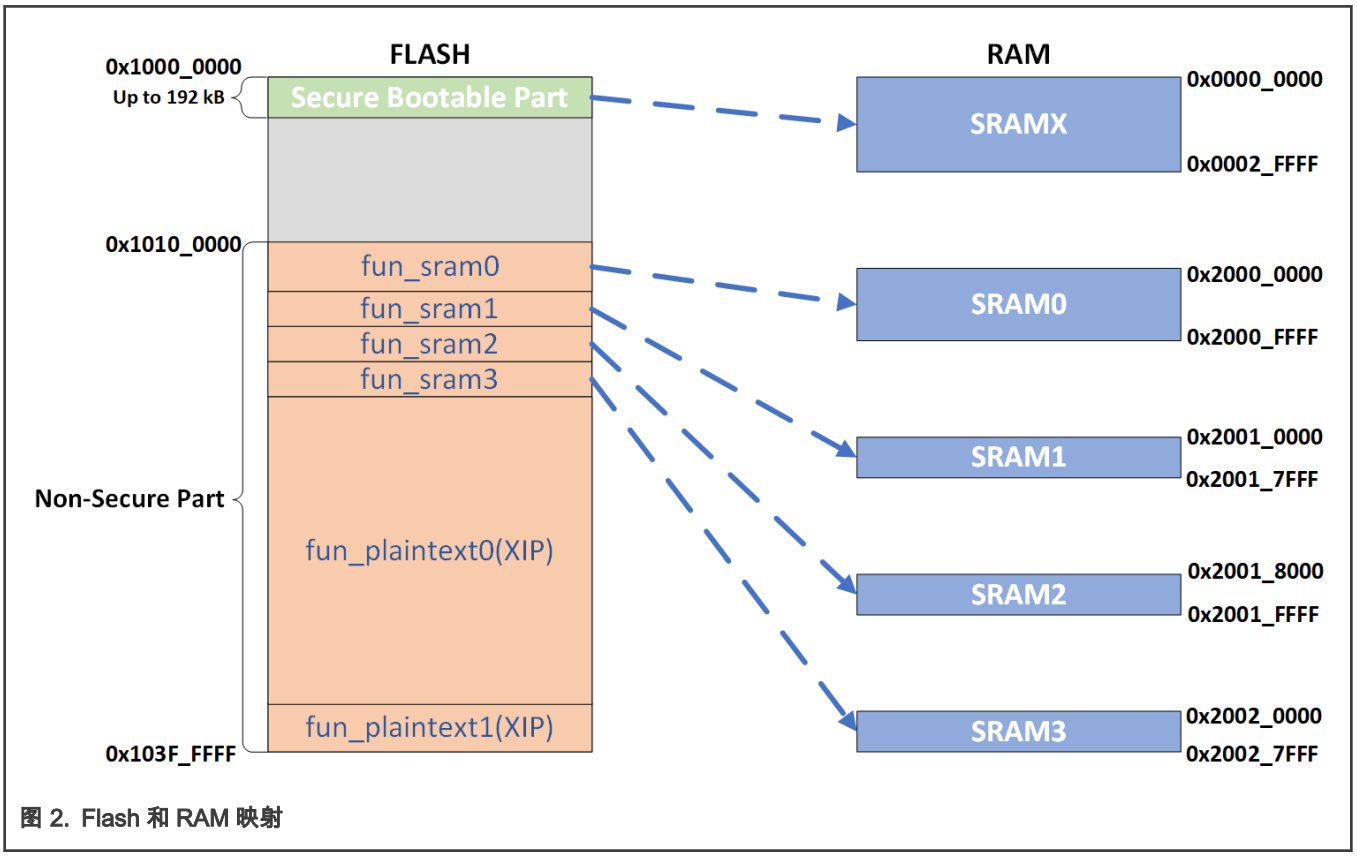


图 2. Flash 和 RAM 映射

注意

在执行代码之前，fun_sram0、fun_sram1、fun_sram2 和 fun_sram3 部分被加载到 RAM 中。

在基于 MCUXpresso IDE 的演示工程中，这些部分由 SDK 提供的 ResetISR 加载到执行地址中。在 MCUXpresso IDE 环境中，修改工程的 ld 文件以实现这种镜像布局。

如 图 3 所示，定义了 fun_plaintext1 节并将其放置在非安全部分中。fun_plaintext1 从 0x103F_FF00 开始。

```
.text_plaintext1 0x103FFF00 : AT(0x103FFF00)
{
    *(.fun_plaintext1)
} > BOARD_FLASH_NS
```

图 3. 链接器脚本

声明函数时，通过属性指令将非机密代码放置在镜像的上述非安全（NS）部分之一。下面的代码段将 ns_print_with_banner 函数代码放在非安全部分的 func_plaintext1 节中：

```
__attribute__ ((section(".fun_plaintext1"))) void ns_print_with_banner(void)
{
    PRINTF("<NS-FLASH:>I'm from non-secure part of QSPI Flash.\r\n");
    PRINTF("<NS-FLASH:>My address: 0x%08X.\r\n", ns_print_with_banner);
}
```

图 4. 属性功能的示例代码

2.3 创建镜像（MCUXpresso IDE）

在 MCUXpresso IDE 环境中创建镜像的步骤如下：

1. 初始化 SPIFI 以启用 XIP。
2. 构建并生成镜像。
3. 将镜像拆分为安全纯文本和非安全纯文本。
4. 基于安全纯文本镜像创建安全可启动部件镜像。

2.3.1 初始化 SPIFI 以启用 XIP

如果应用程序代码大于 192 KB 并且启用了安全启动，则必须执行此步骤才能启用 XIP。

在安全启动部分完成 SPIFI 初始化。

为 XIP 初始化 SPIFI 的代码如 图 5 所示。

```
void app_spifi_init(void)
{
    spifi_config_t config = {0};
    uint32_t sourceClockFreq;
    spifi_command_t command[COMMAND_NUM] = {
        {PAGE_SIZE, false, kSPIFI_DataInput, 1, kSPIFI_CommandDataQuad, kSPIFI_CommandOpcodeAddrThreeBytes, 0x6B},
        {PAGE_SIZE, false, kSPIFI_DataOutput, 0, kSPIFI_CommandDataQuad, kSPIFI_CommandOpcodeAddrThreeBytes, 0x32},
        {1, false, kSPIFI_DataInput, 0, kSPIFI_CommandAllSerial, kSPIFI_CommandOpcodeOnly, 0x05},
        {0, false, kSPIFI_DataOutput, 0, kSPIFI_CommandAllSerial, kSPIFI_CommandOpcodeAddrThreeBytes, 0x20},
        {0, false, kSPIFI_DataOutput, 0, kSPIFI_CommandAllSerial, kSPIFI_CommandOpcodeOnly, 0x06},
        {1, false, kSPIFI_DataOutput, 0, kSPIFI_CommandAllSerial, kSPIFI_CommandOpcodeOnly, 0x31}};

    RESET_PeripheralReset(kSPIFI_RST_SHIFT_RSTn);

    /* Set SPIFI clock source */
    CLOCK_AttachClk(kFRO_HF_to_SPIFI_CLK);
    sourceClockFreq = CLOCK_GetFroHfFreq();

    /* Set the clock divider */
    CLOCK_SetClkDiv(kCLOCK_DivSpifiClk, sourceClockFreq / EXAMPLE_SPI_BAUDRATE, false);

    /* Initialize SPIFI */
    SPIFI_GetDefaultConfig(&config);
    SPIFI_Init(EXAMPLE_SPIFI, &config);

#if defined QUAD_MODE_VAL
    /* Enable Quad mode */
    enable_quad_mode();
#endif

    /* Setup memory command to enable XIP */
    SPIFI_SetMemoryCommand(EXAMPLE_SPIFI, &command[READ]);
}
```

图 5. SPIFI 初始化示例代码

2.3.2 构建和生成镜像

工程的软件编写完成后，将工程编译，然后生成*.axf 文件。

创建一次性二进制或 hex 文件的最简单方法是打开 Project Explorer 中的 Debug 或 Release 文件夹。右键单击*.axf 文件，然后选择 Binary Utilities > Create Binary 选项，如 图 6 所示。

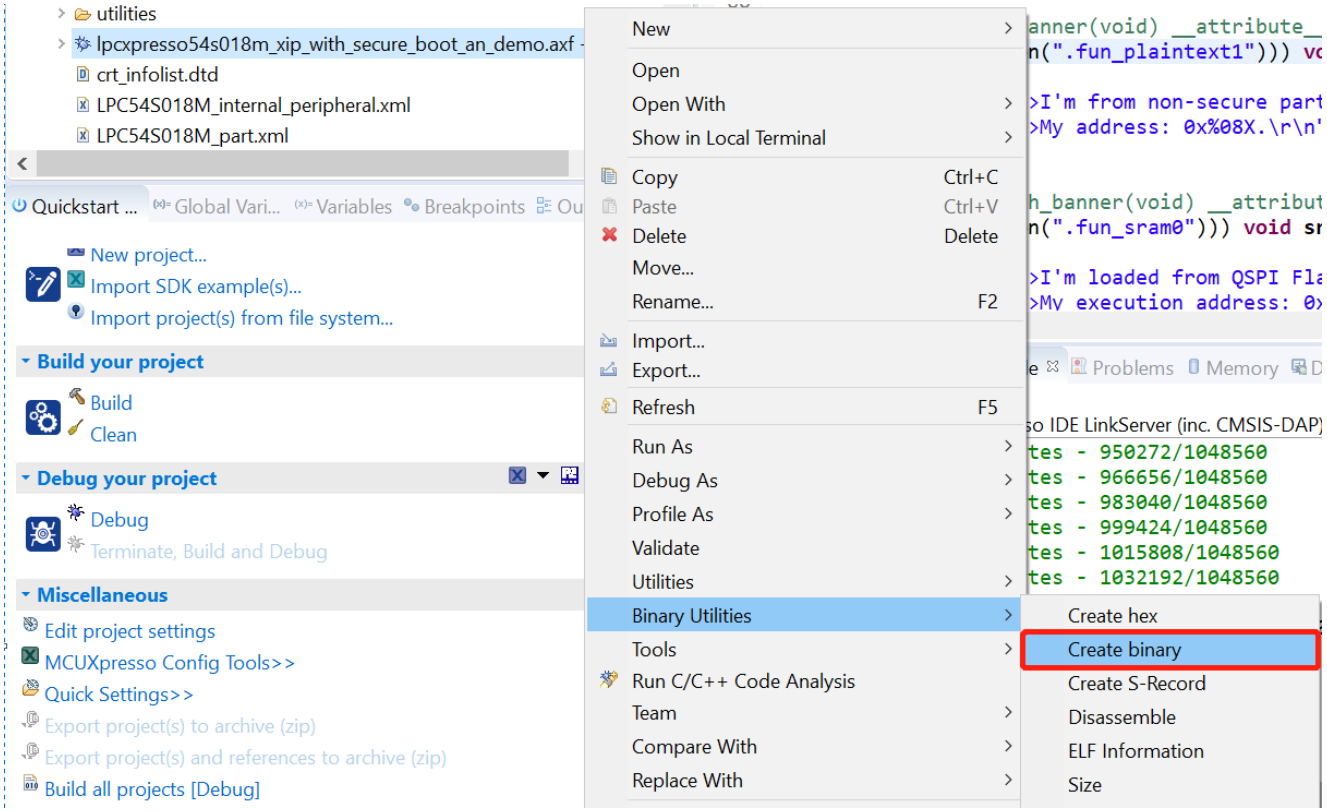


图 6. 生成二进制文件

2.3.3 将镜像拆分为安全纯文本和非安全纯文本

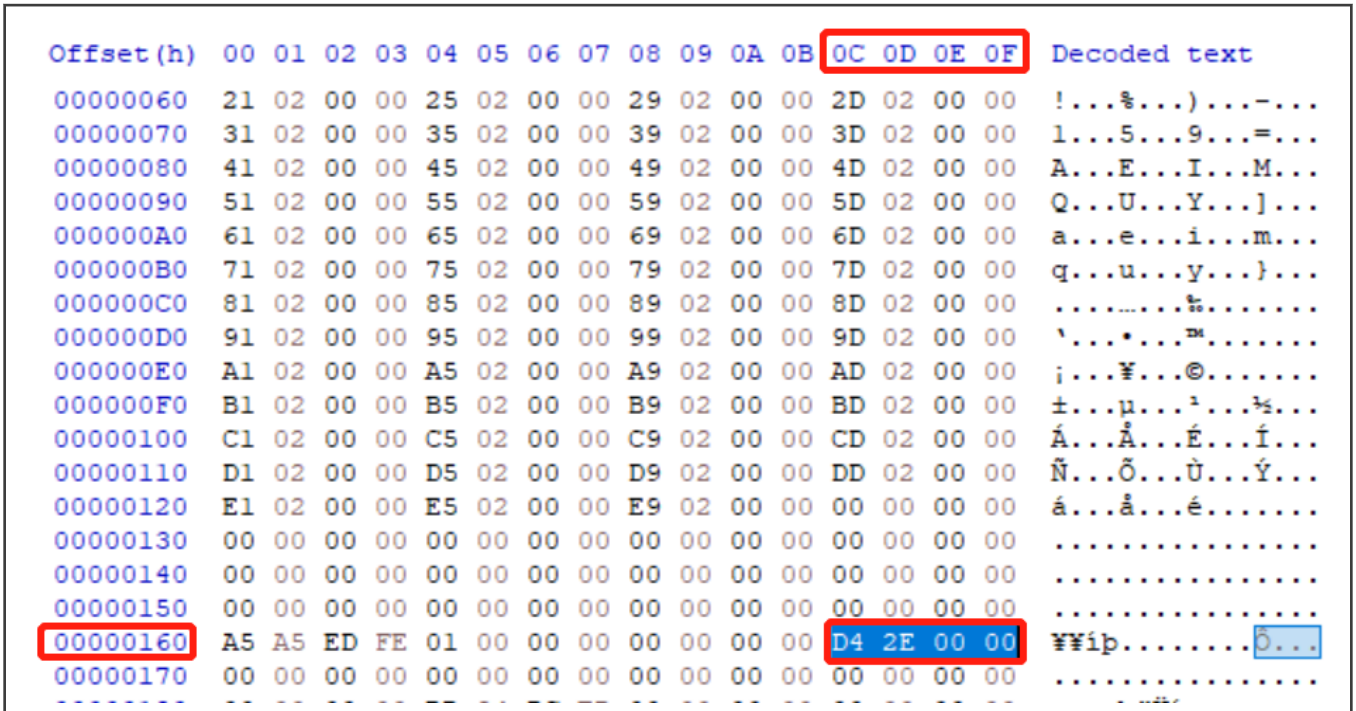
建议使用 HxD 分割镜像。

表 2 描述了明文普通镜像的布局。

表 2. 镜像布局

Offset	Block	值	说明
0x00	Arm 向量表	__initial_sp	栈指针
0x04	Arm 向量表	__initial_pc	镜像执行起始地址
……	……	……	……
0x28	HEADER_OFFSET	HEADER_OFFSET	典型的偏移值是 0x160。
……	……	……	……
HEADER_OFFSET+0x0C	Image_length	xxxxx	镜像的总长度-4。 长度不包括构成 CRC 值字段的四个字节。
……	……	……	……

镜像的长度是由镜像的 header 处得到的。



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000060	21	02	00	00	25	02	00	00	29	02	00	00	2D	02	00	00	!...%...)...-...
00000070	31	02	00	00	35	02	00	00	39	02	00	00	3D	02	00	00	1...5...9...=...
00000080	41	02	00	00	45	02	00	00	49	02	00	00	4D	02	00	00	A...E...I...M...
00000090	51	02	00	00	55	02	00	00	59	02	00	00	5D	02	00	00	Q...U...Y...]...
000000A0	61	02	00	00	65	02	00	00	69	02	00	00	6D	02	00	00	a...e...i...m...
000000B0	71	02	00	00	75	02	00	00	79	02	00	00	7D	02	00	00	q...u...y...}...
000000C0	81	02	00	00	85	02	00	00	89	02	00	00	8D	02	00	00	.....…...‰.....
000000D0	91	02	00	00	95	02	00	00	99	02	00	00	9D	02	00	00	'...•...™.....
000000E0	A1	02	00	00	A5	02	00	00	A9	02	00	00	AD	02	00	00	¡...¥...©.....
000000F0	B1	02	00	00	B5	02	00	00	B9	02	00	00	BD	02	00	00	±...µ...¹...½...
00000100	C1	02	00	00	C5	02	00	00	C9	02	00	00	CD	02	00	00	Á...Å...É...Í...
00000110	D1	02	00	00	D5	02	00	00	D9	02	00	00	DD	02	00	00	Ñ...Õ...Ù...Ý...
00000120	E1	02	00	00	E5	02	00	00	E9	02	00	00	00	00	00	00	á...å...é.....
00000130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000160	A5	A5	ED	FE	01	00	00	00	00	00	00	00	D4	2E	00	00	¥¥íþ.....Ô...
00000170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....

图 7. Image_length（在这个 header 中为 0x0000_2ED4)

镜像的总长度（字节）= Image_length+4。

为了生成安全可启动部分和非安全部分镜像，将原始镜像二进制分为安全纯文本镜像和非安全镜像。

安全纯文本镜像从地址 0 到原始镜像二进制文件的地址（镜像的总长度–1）。此镜像用于创建安全可启动部件镜像。

非安全镜像从地址 0x0010_0000（0x1010_0000-0x1000_0000）到原始镜像的末尾。此镜像作为非安全部分镜像。

2.3.4 基于安全纯文本镜像创建安全可启动部件镜像

使用 elftosb 和 elftosb-gui 创建安全可启动部分的镜像。

- 生成 128 位 AES 密钥。

  使用以下命令生成 128 位 AES 密钥。

  ```
  elftosb.exe --keygen 128 aes128_key.key
  ```

  其中“aes128_key.key”是存储 AES128 密钥的 AES 密钥文件的名称。

- 创建安全可启动镜像。

  打开 elftosb-gui，按照 图 8 所示的步骤创建安全的可启动镜像。

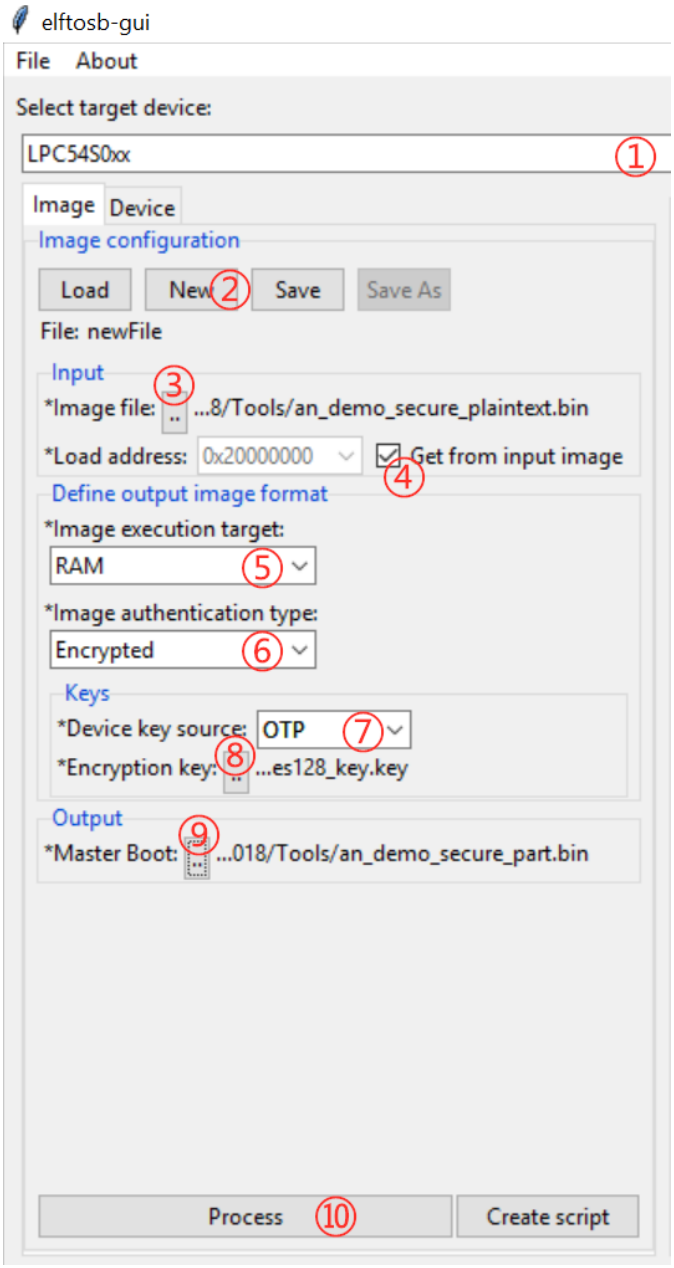


图 8. 通过 elftosb-gui 创建安全可启动镜像

1. 选择 LPC54S0xx 设备。
2. 创建新配置。
3. 选择安全纯文本二进制镜像。
4. 从输入镜像中获取加载地址。
5. 选择镜像执行目标为 RAM。
6. 选择加密作为镜像身份验证类型。
7. 选择 OTP 作为设备密钥源。
8. 选择加密密钥（之前生成的 128 位 AES 密钥）。

9. 选择输出加密镜像的路径和名称。
10. 单击“Process”按钮创建安全可启动部件镜像。

2.4 烧写安全可启动和非安全镜像

要对闪存编程，建议使用 MCUXpresso 和 CMSIS-DAP。

注意

不建议使用 Jflash。Jflash 在编程过程中将校验和填充到镜像中,导致镜像在安全启动期间无法通过验证。

2.4.1 将安全可启动镜像编程到闪存中

- 使用 MCUXpresso IDE 打开 LPC54S018M 或 LPC54S018 的任一 SDK 工程。

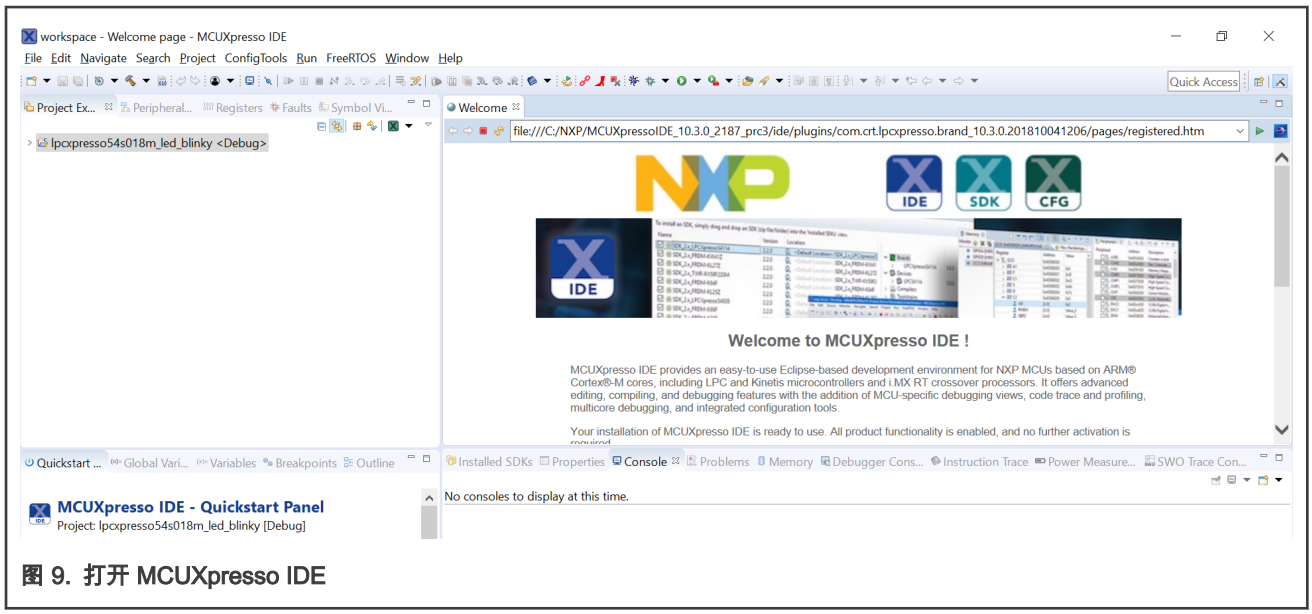


图 9. 打开 MCUXpresso IDE

- 按 图 10 所示的顺序单击按钮，打开 MCUXpresso IDE LinkServer (inc. CMSIS-DAP)。

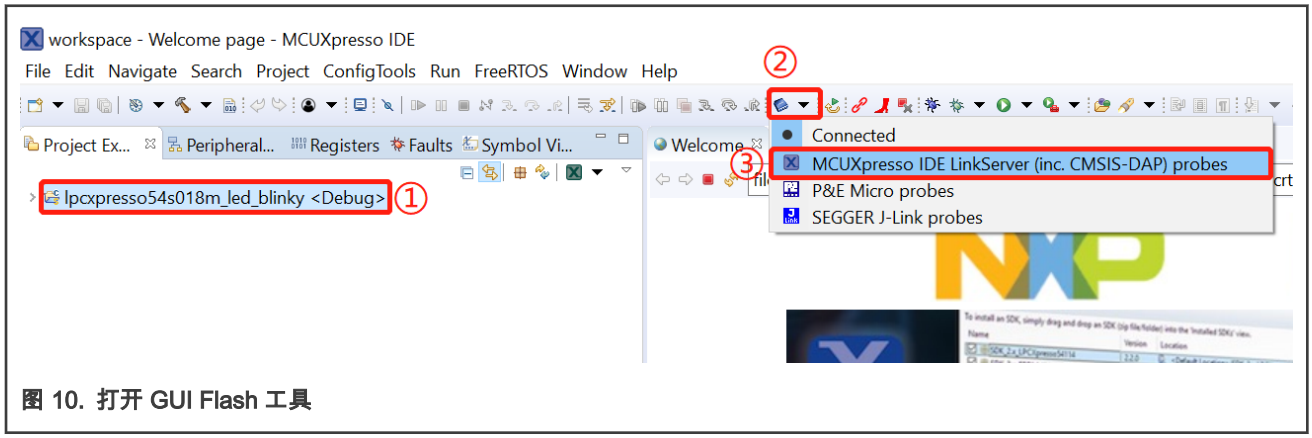


图 10. 打开 GUI Flash 工具

结果在 图 11 中展示。

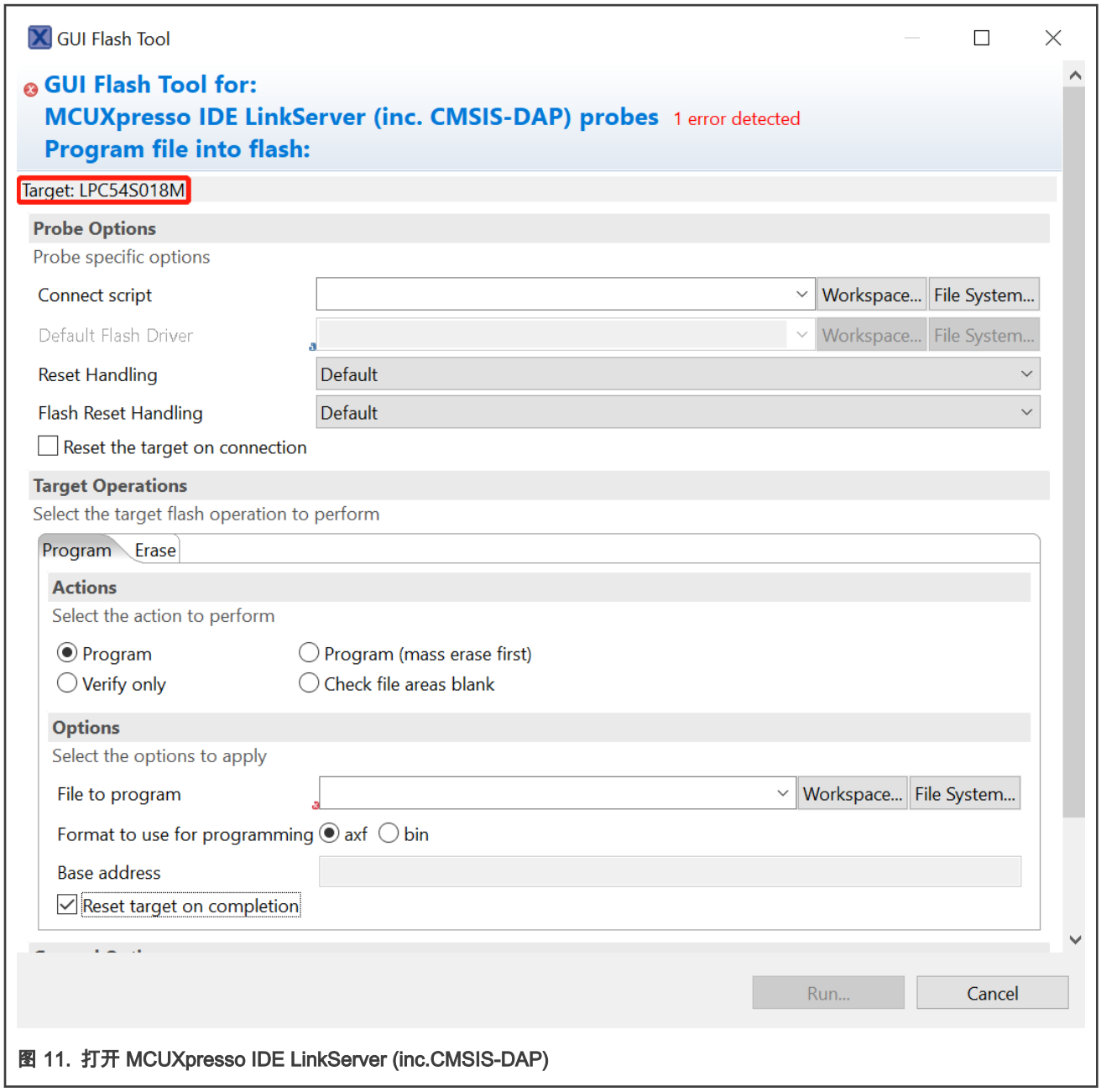


图 11. 打开 MCUXpresso IDE LinkServer (inc.CMSIS-DAP)

按 图 12 所示的屏幕截图进行配置，尤其是屏幕截图的红色部分。

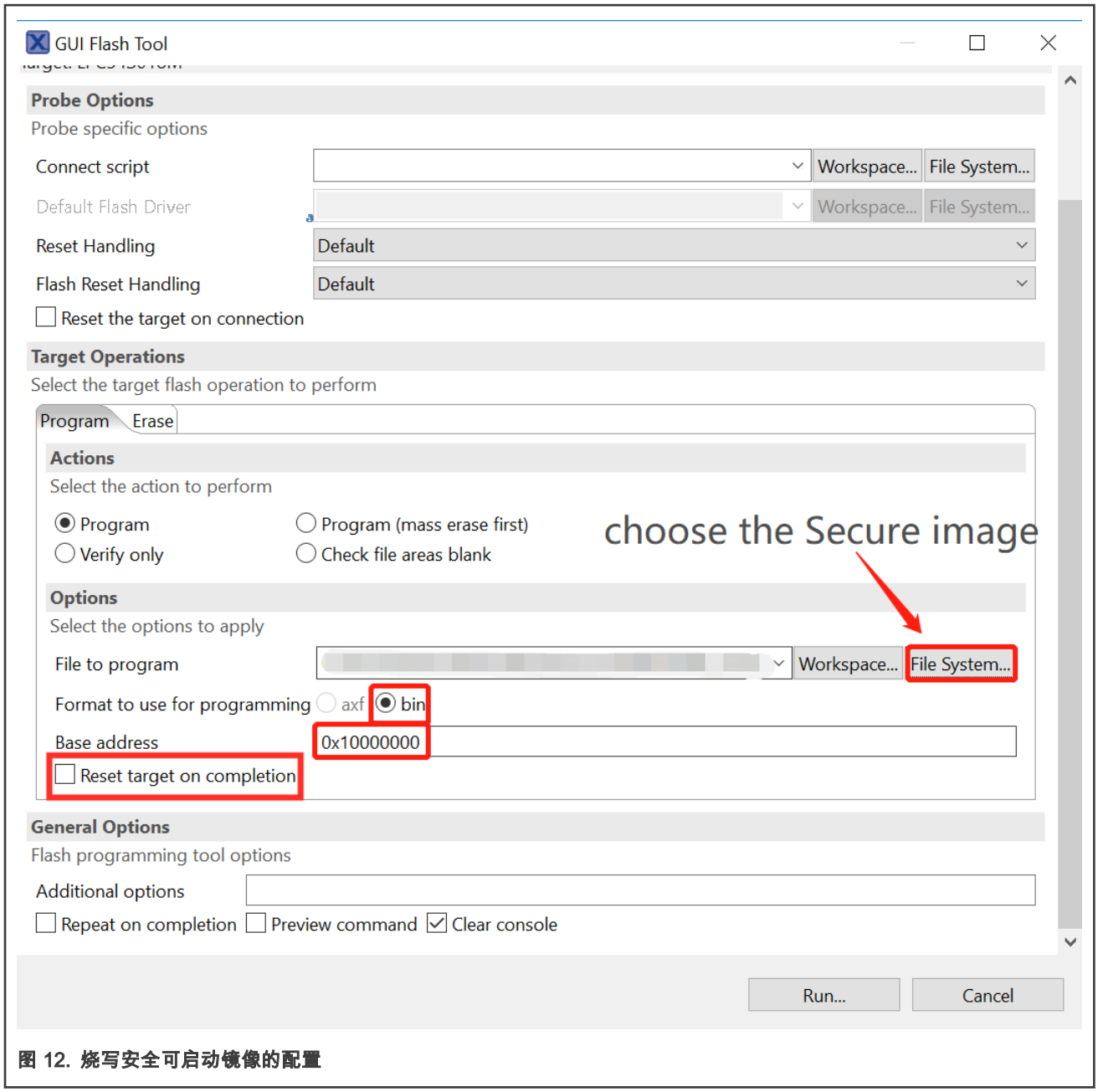


图 12. 烧写安全可启动镜像的配置

- 单击 Run 按钮将安全可启动镜像烧写到 Flash 中。

2.4.2 将非安全镜像烧写到 Flash 中

1. 遵循步骤 1 和步骤 2。
2. 更改如 图 13 所示的配置，特别是屏幕截图中的红色部分。

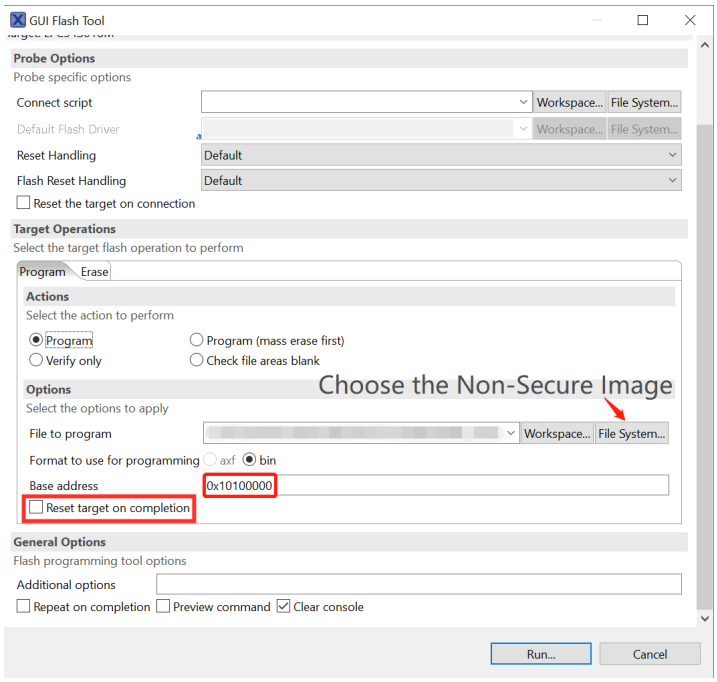


图 13. 烧写非安全镜像的设置

3. 单击 Run 按钮将非安全镜像烧写到 Flash 中。

2.5 转换 elftosb 生成的密钥文件

elftosb 生成的密钥文件是 ASCII 格式的。对于 blhost，应该将其转换为十六进制格式，图 14 和 图 15 显示了如何转换密钥文件。

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000   30 45 30 31 37 37 39 35 37 33 44 32 37 45 43 38  0E01779573D27EC8
00000010   31 38 39 34 35 31 30 30 31 36 42 34 39 39 42 32  1894510016B499B2
```

图 14. aes128_key.key

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000   0E 01 77 95 73 D2 7E C8 18 94 51 00 16 B4 99 B2  ..w•sÒ~È.”Q..´™²
```

图 15. aes128_key.bin

2.6 烧写 128 位 AES 密钥和相关 OTP 位字段以启用安全启动

建议使用 blhost 对 OTP 位进行编程。对于 LPC54S0xx，flashloader 应该加载到片上 RAM 中，然后 blhost 将被启用。

2.6.1 使用 DFU 将 flashloader 加载到 RAM 中

配置 ISP 引脚，使芯片进入 USB0 DFU 启动模式。

表 3. 基于 ISP 引脚的启动源

启动模式	ISP2 PIO0\U 6 引脚	ISP1 PIO0\ U 5 引脚	ISP0 PIO0\U 4 引脚	说明
USB0 DFU 启动	低	高	低	USB DFU 类用于通过 USB0 全速端口将镜像下载到 SRAM 中。

用 USB 连接 LPC54S0xx 设备 USB0 和 PC。

使用以下命令将 flashloader 加载到 RAM 中。flashloader.bin 文件位于 `an_lpc54s0_xip_with_secureboot` 中。它也可以通过编译位于在 `sdk\boards\lpcxpresso54s018\bootloader_examples\flashloader` 中的工程。

```
dfu-util.exe -D flashloader.bin
```

2.6.2 使用 blhost 编程 128 位 AES 密钥和相关 OTP 位字段

在以 USB DFU 模式连接的设备上下载 flashloader 二进制文件并在 LPC54S0xx 平台上开始执行后，LPC54S0xx 平台高速 USB1 和主机之间仍然存在物理 USB 连接。此时 flashloader 做好了从 USB1 接收命令的准备。

2.6.2.1 128 位 AES 密钥

使用以下命令编程 128 位 AES 密钥。

```
blhost.exe -u 0x1fc9,0x01a2 -- program-aeskey aes128_key.bin
```

2.6.2.2 安全启动类型位字段

使用以下命令将安全启动类型编程为强制加密。

```
blhost.exe -u 0x1fc9,0x01a2 -- efuse-program-once 12 00000010
```

2.6.2.3 安全启动启用位字段

使用以下命令启用安全启动。

```
blhost.exe -u 0x1fc9,0x01a2 -- efuse-program-once 12 00000004
```

3 演示

本节介绍集成环境以及演示步骤和结果。

3.1 环境

本节介绍硬件和软件环境。

3.1.1 硬件环境

- 开发板
  - LPCXpresso54S018（LPC54S018-EVK）或 LPCXpresso54S018M（LPC54S018M-EVK）
- 调试器
  - 板上集成 CMSIS-DAP 调试器
- 杂项

— 两根 Micro USB 线

— PC

3.1.2 软件环境

- 工具链
  - MCUXpresso IDE 版本 10.3.0
- 软件包
  - an_lpc54s0_xip_with_secureboot.zip

3.2 步骤和结果

基本步骤如下：

1. 构建和编译

   构建并编译位于 an_lpc54s0_xip_with_secureboot/an_demo 中的演示工程。

2. 处理镜像

   根据将镜像拆分为安全纯文本和非安全纯文本和基于安全纯文本镜像创建安全可启动部件镜像处理镜像。

3. 下载

   按照烧写对安全可启动和非安全部件镜像进行编程下载镜像。

4. 烧写 AES 密钥

   按照转换 elftosb 生成的密钥文件对 AES 密钥进行烧写。

5. 烧写相关的 OTP 位字段

   按照烧写 128 位 AES 密钥和相关 OTP 位字段以启用安全启动，以烧写相关 OTP 位字段。

6. 运行

   按下电路板上的复位按钮，电路板将复位并运行。

7. 结果

   图 16 显示了演示代码在终端 115200+8+N+1 上输出的消息。

```
<S:>The image has been loaded into SRAMX from flash by ROM code.
<S:>The encrypted image has been descrypted by ROM code.
<S:>Executed the image located in SRMAX.
<S:>Enable the SPIFI for executing the plain-text code located in flash.
<S:>Call a function located in flash(XIP).

<NS-FLASH:>I'm from non-secure part of QSPI Flash.
<NS-FLASH:>My address: 0x103FFF0D.

<NS-SRAM0:>I'm loaded from QSPI Flash, and excuted from SRAM0.
<NS-SRAM0:>My execution address: 0x2000100D.

<NS-SRAM1:>I'm loaded from QSPI Flash, and excuted from SRAM1.
<NS-SRAM1:>My execution address: 0x20010001.

<NS-SRAM2:>I'm loaded from QSPI Flash, and excuted from SRAM2.
<NS-SRAM2:>My execution address: 0x20018001.

<NS-SRAM3:>I'm loaded from QSPI Flash, and excuted from SRAM3.
<NS-SRAM3:>My execution address: 0x20020001.

<S:>The LED will blink per second.
<S:>Enter any character, which will be echoed to terminal.
```

图 16. 终端上输出的信息

带有<S:>的信息表示它是从安全可启动部件镜像中输出。带有<NS:>的信息，表示它在非安全部件镜像中输出。

如终端上显示的输出信息所述，程序将回显每个输入的字符。板载 LED3 也会闪烁。

4 修订记录

版本号	日期	说明
0	2019 年 2 月 18 日	首次发布
1	2019 年 2 月 25 日	更新了图 13 和术语中的工具路径。
2	2020 年 9 月 18 日	• 更新表 1 • 更新了 elftosb 生成的转换密钥文件 • 更新程序 128 位 AES 密钥和相关 OTP 位字段以启用安全启动 • 更新使用 blhost 编程 128 位 AES 密钥和相关 OTP 位字段 • 添加 128 位 AES 密钥

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Limited warranty and liability — Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

Right to make changes - NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer’s applications and products. Customer’s responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer’s applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 2020 年 9 月 18 日

Document identifier: AN12352

arm