Application Note

*用于具有 **AES** 硬件加密功能的 **TM4C129** 器件的共享密钥引导加载程序*

Bob Crosby and Ralph Jacobi

摘要

本应用报告提供了一个基础架构，其中产品所有者可以使用一对 256 位共享密钥在其工厂创建用于现场产品升级的映像。其中一个密钥用于生成签名。另一个密钥用于加密映像。现场产品有一个引导加载程序，共享相同的两个密钥。该程序解密签名和映像，并用一个密钥将其编程至闪存中。然后，用第二个密钥对映像进行签名验证。仅当对映像验证时，该引导加载程序才会将执行转移到新映像。

此示例演示了一种机制，防止意外或简单的尝试上传非由本报告中所述工具创建的代码映像。这并不意味着该方法足以防止更具复杂性的攻击。用户有责任确定该方法是否适合其应用。

可以从以下 URL 下载本文档所述的项目配套资料：https://www.ti.com/lit/zip/spma083。

内容

1 实现.....2
1.1 闪存引导加载程序项目.....2
1.2 映像创建项目.....3
1.3 密钥映像工程.....4
1.4 EK-TM4C129EXL 示例应用程序项目.....4
1.5 DK-TM4C129X 示例应用程序项目.....4
1.6 基于 RAM 的 EEPROM 擦除项目.....4
2 示例走查.....4
2.1 编译环境.....4
2.2 将示例导入 Code Composer Studio.....5
2.3 设置密钥和变量.....8
2.4 运行 shared_key_image_encrypt 工具.....9
2.5 运行共享密钥串行引导加载程序.....15
2.6 返回引导加载程序.....21
3 总结.....22

插图清单

图 2-1. 导入 CCS 项目步骤 1。.....5
图 2-2. 导入 CCS 项目步骤 2。.....6
图 2-3. 导入 CCS 项目步骤 3。.....7
图 2-4. 导入 CCS 项目步骤 4。.....8
图 2-5. 执行 *shared_key_image_encrypt* 程序.....10
图 2-6. 识别 COM 端口.....10
图 2-7. 在 Tera Term 中选择 COM 端口.....11
图 2-8. 在 Tera Term 中配置 COM 端口.....11
图 2-9. 执行 *shared_key_image_encrypt* 程序.....12
图 2-10. *shared_key_image_encrypt* 命令菜单.....12
图 2-11. 加载密钥映像.....13
图 2-12. 已成功加载密钥.....14
图 2-13. 加载程序映像.....14
图 2-14. 验证程序映像.....15

English Document: SPMA083

图 2-15. 使用 Code Composer Studio 擦除闪存.....17
图 2-16. 加载 eeprom_erase.out.....17
图 2-17. 配置 LM Flash Programmer 以对引导加载程序进行编程.....18
图 2-18. 使用 LM Flash Programmer 对引导加载程序进行编程.....19
图 2-19. 配置 LM Flash Programmer 以对应用程序代码进行编程.....20
图 2-20. 使用 LM Flash Programmer 对应用程序代码进行编程.....21

商标

1 实现

代码加密和验证方法将通过五个示例项目实现：

- 闪存引导加载程序项目
- 映像创建项目
- 密钥创建项目
- EK-TM4C129EXL LaunchPad 开发套件示例应用程序项目
- DK-TM4C129X 开发套件示例应用程序项目

还包括第六个项目，用于擦除 EEPROM 中的密钥。该项目只是一个补充资源，供开发期间使用。

1.1 闪存引导加载程序项目

闪存引导加载程序项目 *shared_key_boot_serial* 是一个基于当前闪存串行引导加载程序实现的示例，具有以下附加特性：

- 引导加载程序位于闪存空间的前 16KB 中。
- 引导加载程序将检查存储在两个 EEPROM 块中的 256 位密钥。
- 如果未找到有效密钥（全部为 FF 的密钥无效）：
  - 引导加载程序将进入等待上传映像而不解密该映像的模式。如果从地址 0x4000 开始的映像仅包含 1 到 4 个有效密钥，这些密钥将被复制到 EEPROM，并且 0x4000 处的扇区将被擦除。
  - 引导加载程序将检查 JTAG 是否被禁用。如果未被禁用，它将禁用 JTAG 并对闪存的第一个扇区进行写保护（仅限发布版配置）。
- 如果找到了有效的密钥映像：
  - 引导加载程序在对数据进行编程前，将使用带有 256 位密钥的 AES 解密来解密传入的数据流。
  - 传入的映像应包含从地址 0x4000 开始的完整闪存映像，闪存末尾的 16 个字节包含身份验证签名。
    - 闪存结束地址 `APP_END` 在头文件 `linker_defines.h` 中定义。该文件由密钥创建项目和引导加载程序项目使用。可以将闪存应用结束地址定义为小于实际的闪存结束地址，以减少更新器件的时间，但一旦在引导加载程序中进行了定义，这个值就将成为可以上传到该器件的映像的最大尺寸。
  - 引导加载程序将根据地址 0x4000 到闪存末尾减去 16 个字节的数据，计算 AES-CBCMAC 签名。
  - 如果该签名与存储在闪存末尾的签名匹配，引导加载程序将隐藏包含密钥的 EEPROM 块并跳转到应用程序代码。

> 更改加密密钥的方法是添加一个新函数，该函数可以使用具有当前密钥和特定命令的有效散列来撤销当前密钥。

1.1.1 对示例项目 boot_serial 的变更

1.1.1.1 对 bl_config.h 的变更

- 第 29 行：通过添加“**#include** "linker_defines.h"”修改空白行。这定义了闪存应用程序的起始地址
- 第 80 行：修改为“**#define** APP_START_ADDRESS APP_BASE”。
- 第 218 行：取消 FLASH_CODE_PROTECTION 的注释。
- 第 1402 行：取消“void MyReinitFunc(void);”的注释。
- 第 1405 行：取消“#define BL_INIT_FN_HOOK MyReinitFunc”的注释。
- 第 1442 行：取消“void MyEndFunc(void);”的注释。
- 第 1445 行：取消“#define BL_END_FN_HOOK MyEndFunc”的注释。
- 第 1453 行：取消“void MyDecryptionFunc(unsigned char *pucBuffer, unsigned long ulSize);”的注释并将“long”更改为“int”。
- 第 1460 行：取消“#define BL_DECRYPT_FN_HOOK MyDecryptionFunc”的注释。
- 第 1471 行：取消“unsigned long MyCheckUpdateFunc(void);”的注释。
- 第 1479 行：取消“#define BL_CHECK_UPDATE_FN_HOOK MyCheckUpdateFunc”的注释。

1.1.1.2 添加的新函数

文件 `shared_key_functions.c` 中添加了以下新函数。

1.1.1.2.1 MyCheckUpdateFunc

该例程在引导加载程序的开始处调用，确定引导加载程序应运行应用程序代码，还是等待新的映像。它检查以下五个条件：

- 如果代码在不支持 AES 硬件加密的器件上运行，引导加载程序将不执行应用程序。
- 如果器件不具备 EEPROM 中存储的有效密钥，将不执行应用程序。相反，引导加载程序将进入一种状态，等待密钥的未加密映像进行上传。
- *MyCheckUpdateFunc* 将调用用户定义的函数 *CheckGPIOPin*。如果此函数返回 true，引导加载程序将不执行应用程序，而会等待上传新的加密应用程序映像。在示例引导加载程序中，它将检查 EK-TM4C129EXL LaunchPad *SW2* 按钮是否被按低了。
- *MyCheckUpdateFunc* 将检查是否发生了软件复位，以及是否设置了基于 RAM 的预定值，以确认应用是否调用了引导加载程序。如果结果都为 true，引导加载程序将等待上传新的加密应用程序映像。
- *MyCheckUpdateFunc* 将计算应用程序代码区域的 AES-CBCMAC 哈希值，并将结果与应用程序映像的最后 16 个字节进行比较。如果映像匹配，引导加载程序将执行应用程序代码。如果映像不匹配，它将等待上传新的加密应用程序映像。

1.1.1.2.2 MyReinitFunc

由于密钥只能在复位后、调用应用程序前从 EEPROM 中读取，因此，如果直接从应用程序中调用，引导加载程序将无法工作。所以，此函数将在一个静态 RAM 位置设置一个预定义值，然后引起软件复位。*MyCheckUpdateFunc* 函数将使用预定义的 RAM 值检查软件复位，并利用该条件留在引导加载程序中。

1.1.1.2.3 MyEndFunc

此函数用于将新的密钥映像加载到 EEPROM 中。它会在所有数据都编程完毕后、执行传递给新程序前调用。如果器件包含有效密钥，或器件不具备 AES 模块，它将返回而不执行。如果器件不包含有效密钥且具备 AES 模块，此函数将检查加载的映像以确认其是否为有效的密钥映像。如果是，它会将密钥编程到 EEPROM 中，隐藏 EEPROM 块，擦除 `APP_BASE` 处的闪存，最终重置器件。

1.1.1.2.4 MyDecryptionFunc

此函数使用带 256 位密钥的 AES-CBC 对输入缓冲器进行解密。输入缓冲器的大小必须为 16、32、48 或 64 字节。

1.2 映像创建项目

映像创建项目 *shared_key_image_encrypt* 仅在带有 AES 模块的 TM4C129 器件上运行。它使用 UART0 运行命令行程序。该程序执行以下命令：

- K – 加载两个 256 位密钥对并将它们保存在 EEPROM 中
- D - 禁用 JTAG (仅当密钥已在 EEPROM 中时)
- L – 使用 XMODEM 传输加载从地址 0x4000 开始的程序。
- X – 执行在地址 0x4000 加载的程序
- O – 输出经过签名和加密的二进制映像。
- H – 打印帮助菜单

1.3 密钥映像工程

key_image 是一个简单的汇编语言项目，用于创建加密和验证密钥的二进制映像。

1.4 EK-TM4C129EXL 示例应用程序项目

shared_key_boot_demo 是一个简单的应用程序项目，可用于演示创建应用程序、对其进行签名和编码，然后使用安全启动加载程序对其进行编程的过程。应用程序运行时，一个 LED 灯会一直闪烁，直到长按 *SW1* 按钮为止，这会让应用程序返回到引导加载程序。

1.5 DK-TM4C129X 示例应用程序项目

shared_key_boot_demo_DK 是一个简单的应用程序项目，可用于演示创建应用程序、对其进行签名和编码，然后使用安全启动加载程序对其进行编程的过程。运行时，应用程序会在 LCD 屏幕上显示一条消息。按下 *Down* 开关时，应用程序将返回到引导加载程序。

1.6 基于 RAM 的 EEPROM 擦除项目

对于存储在 EEPROM 中的密钥，除非器件执行“解锁”程序，或者它们被复位后运行的应用程序明确擦除，否则将一直保留。项目 *eeprom_erase* 在系统重置后由 Code Composer Studio™ 运行时会擦除密钥。此项目只是供引导加载程序开发期间使用的一个辅助工具，可从 EEPROM 中清除任何已加载的密钥，无需经历更繁琐的器件解锁过程。

2 示例走查

2.1 编译环境

这些示例是使用以下工具编译和测试的：

- EK-TM4C129EXL LaunchPad 开发套件
- Code Composer Studio v10.2
- TI Arm C Compiler v20.2.5.LTS
- TivaWare v2.2.0.295

2.2 将示例导入 Code Composer Studio

本文档附加了六个 CCS 项目示例作为配套资料。本文档中讨论的项目配套资料可从以下 URL 下载：https://www.ti.com/lit/zip/spma083。项目可以解压缩至一个文件夹，也可以保留为 zip 文件。两种格式都可以导入 CCS。

1. 要将项目导入 CCS，请先选择“File”->“Import”。

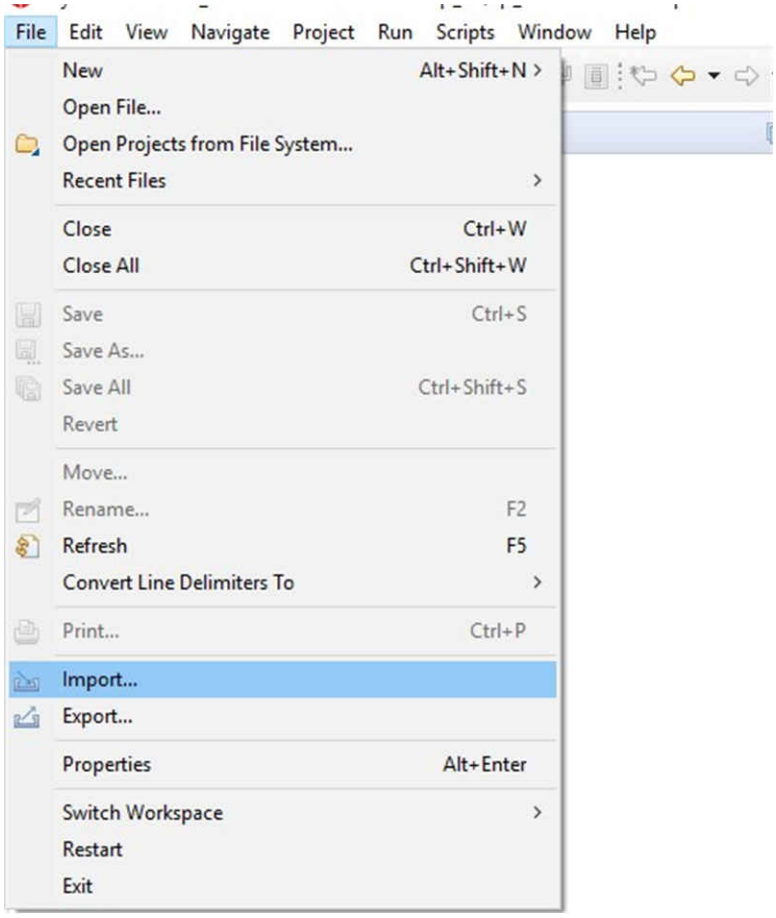

图 2-1. 导入 CCS 项目步骤 1。

2. 选择“CCS Projects”以导入示例，然后点击“Next”（下一步）。

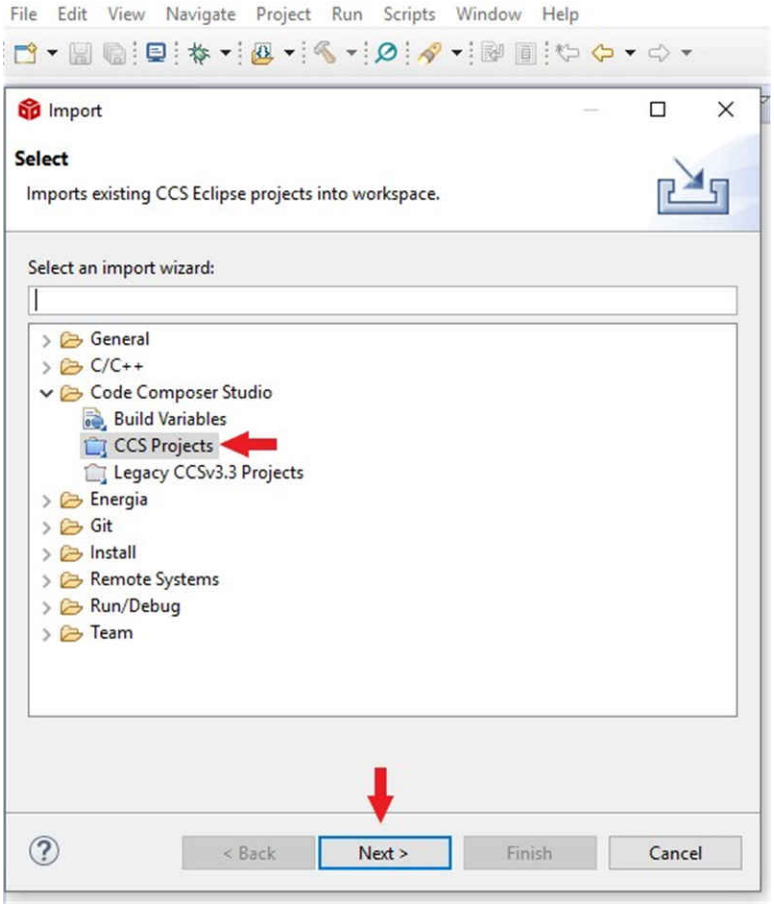

图 2-2. 导入 CCS 项目步骤 2。

3. 接下来，提供解压缩项目（选择第一个单选按钮）或直接导入 zip 文件（选择第二个单选按钮）的路径。选中“Copy projects into workspace”框。

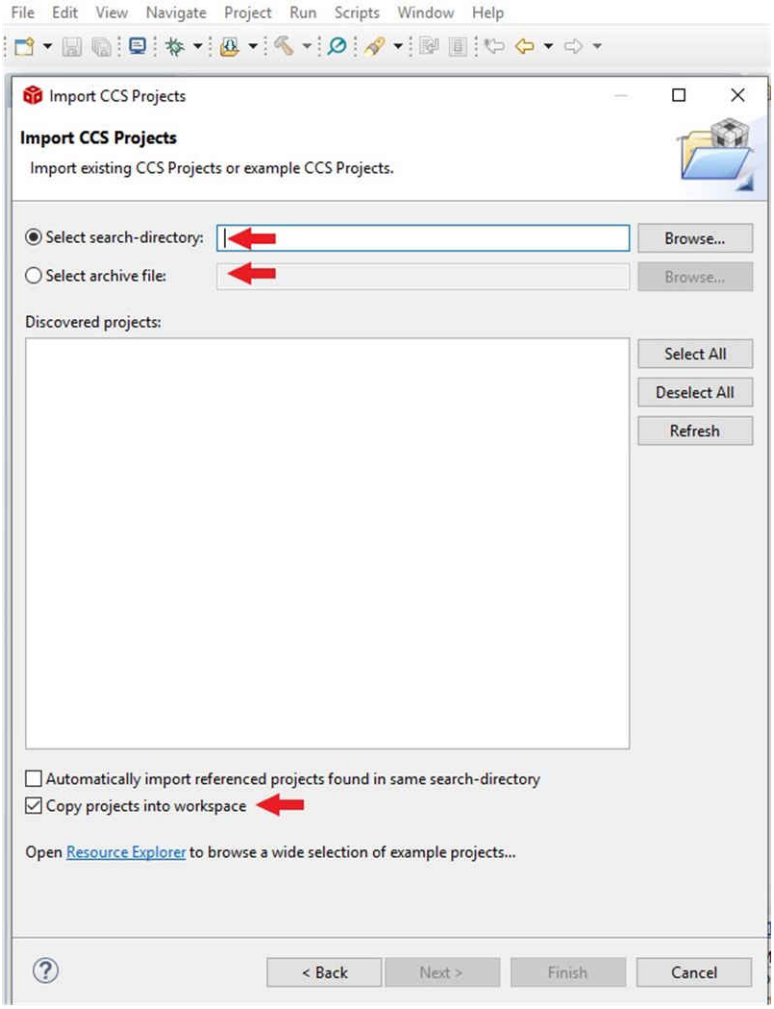

图 2-3. 导入 CCS 项目步骤 3。

4. 提供项目路径后，总共会显示六个已发现的项目。首先点击“Select All”（全选）按钮，然后点击“Finish”（结束）按钮完成导入。

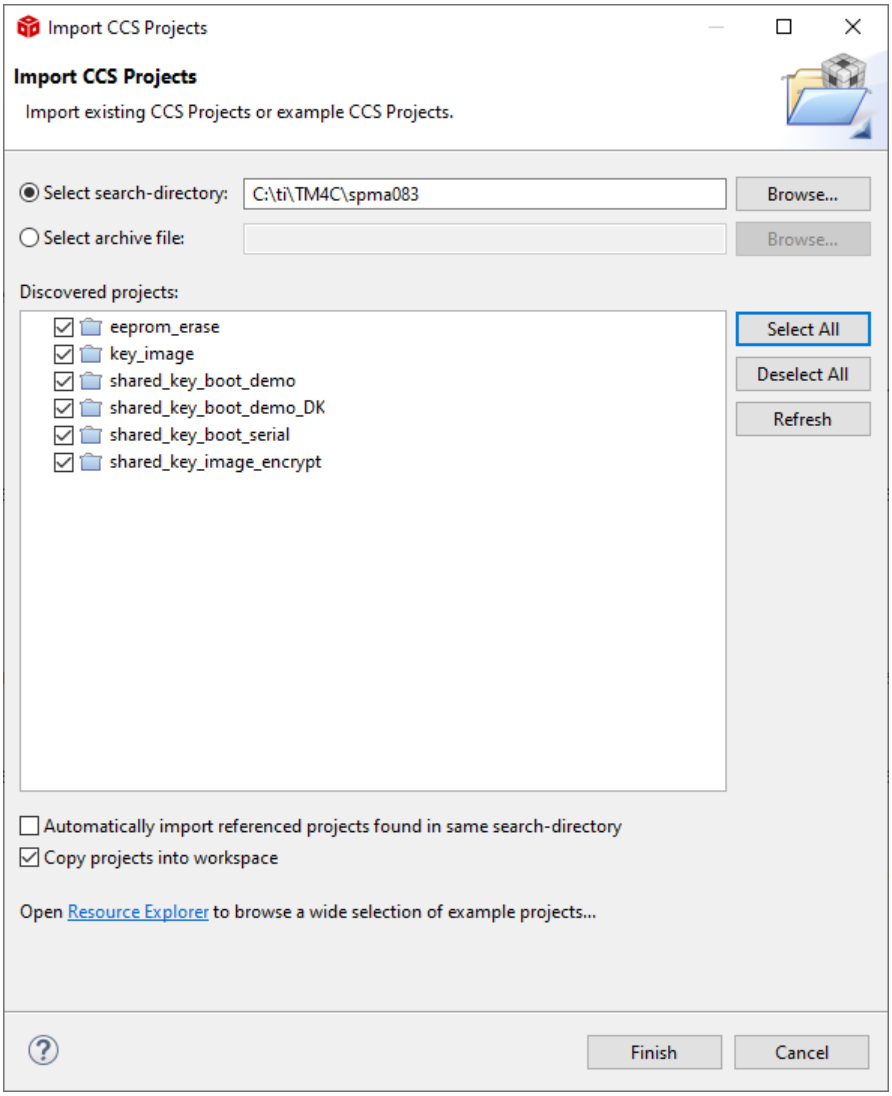

图 2-4. 导入 CCS 项目步骤 4。

2.3 设置密钥和变量

2.3.1 密钥

密钥在 `key_image` 目录的 `key_image.asm` 文件中定义。该文件在 ***key_image*** 项目中使用，也通过 ***shared_key_boot_serial*** 示例项目中的链接使用。

```
  .sect ".keyimage"
  .global key
key
; Encryption/Decryption Key 0
      .word 0x3ed44417, 0x8d849ffc, 0x4719e4dc, 0x71583965
      .word 0x84b6ba7d, 0xa96ff68a, 0xb79d9da8, 0x5f747e02
; Authentication Key 0
      .word 0xf74e59bf, 0xd19cd4e1, 0x630303f8, 0xe5bb8089
      .word 0x0e3bc945, 0xffc85239, 0x53289e9c, 0x5f906df8
; Encryption/Decryption Key 1
      .word 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF
      .word 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF
; Authentication Key 1
      .word 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF
      .word 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF
```

从代码中可以看出，实际上定义了四个 256 位密钥。第一个密钥用于加密/解密。 第二个密钥用于身份验证。最后两个密钥当前未使用。

对于初始评估和开发，可以使用本例中给出的密钥，但对于真实项目，则需要更改这些密钥并在安全的位置保密。

2.3.2 初始化矢量

初始化矢量在文件 `initialization_vector.c` 中定义。该文件位于项目 *shared_key_image_encrypt* 中，由项目 *shared_key_boot_serial* 中的链接引用。在本例中，初始化矢量全部为零。建议为真实产品选择一个随机的 128 位初始化矢量。

2.3.3 应用程序起始地址和闪存大小

项目 *shared_key_image_encrypt* 中的文件 `linker_defines.h` 定义了闪存中用于应用程序的空间和 RAM 的大小。它在 *shared_key_image_encrypt* 和 *shared_key_boot_serial* 项目的源代码和链接命令文件中使用。

```
/*
 * linker_defines.h
 */
#ifndef LINKER_DEFINES_H_
#define LINKER_DEFINES_H_

#define APP_BASE 0x00004000
#define APP_END 0x00100000
#define RAM_BASE 0x20000000

#endif /* LINKER_DEFINES_H_ */
```

2.3.3.1 APP_BASE

`APP_BASE` 是应用程序闪存区域的开始。它必须在闪存扇区边界的开始处。对于 TM4C129 器件，这意味着它必须是 0x4000 的倍数。由于引导加载程序在第一个扇区中，因此 `APP_BASE` 不能为零。通常不需要更改 `APP_BASE` 的默认值 0x4000。

2.3.3.2 APP_END

`APP_END` 是应用程序可用闪存之外的第一个字节的地址。AEC-CBCMAC 身份验证签名存储在 `APP_END` 之前的 16 个字节中。具有 1MB 闪存器件的默认值为 0x100000。具有 512KB 闪存器件的默认值为 0x80000。

用户可以选择使用更小尺寸的闪存来减少下载新映像的时间。如果是这样，`APP_END` 的值必须是扇区大小 0x4000 的倍数。引导加载程序和创建加密映像的工具中的 `APP_END` 必须使用相同的值。一旦将引导加载程序编程到器件中，就无法更改应用程序可用闪存的大小。确保在闪存中留出足够的未使用空间，以支持未来应用程序的增长。

每次加载固定大小映像的原因是，有一种针对 AES-CBC 编码数据的攻击使用恶意代码来扩展长度。使用固定数据长度可以防御这种攻击方法。

2.3.3.3 RAM_BASE

`RAM_BASE` 用于定义器件上 RAM 的起始地址。通常没有理由更改此值的默认值 0x2000.0000。

2.4 运行 shared_key_image_encrypt 工具

在进行第 3.3 节中描述的任何修改后，重新编译项目 *shared_key_image_encrypt* 和 *key_image* 的 `Debug` 配置。然后，重新编译项目 *shared_key_boot_serial* 的 `Debug_wKey` 配置。调试版配置不会锁定 JTAG，可以在开发期间使用它。发布版配置将锁定 JTAG，且应当用于共享密钥引导加载程序的生产版本。

1. 使用 USB 电缆将 EK-TM4C129EXL LaunchPad 开发套件从 LaunchPad 的 DEBUG 端口连接到 PC 上的 USB 端口，启动 *shared_key_image_encrypt* 项目。这可以通过在 Project Explorer 窗口中突出显示该项目，然后点击图 2-5 中的黄色调试图标来完成。

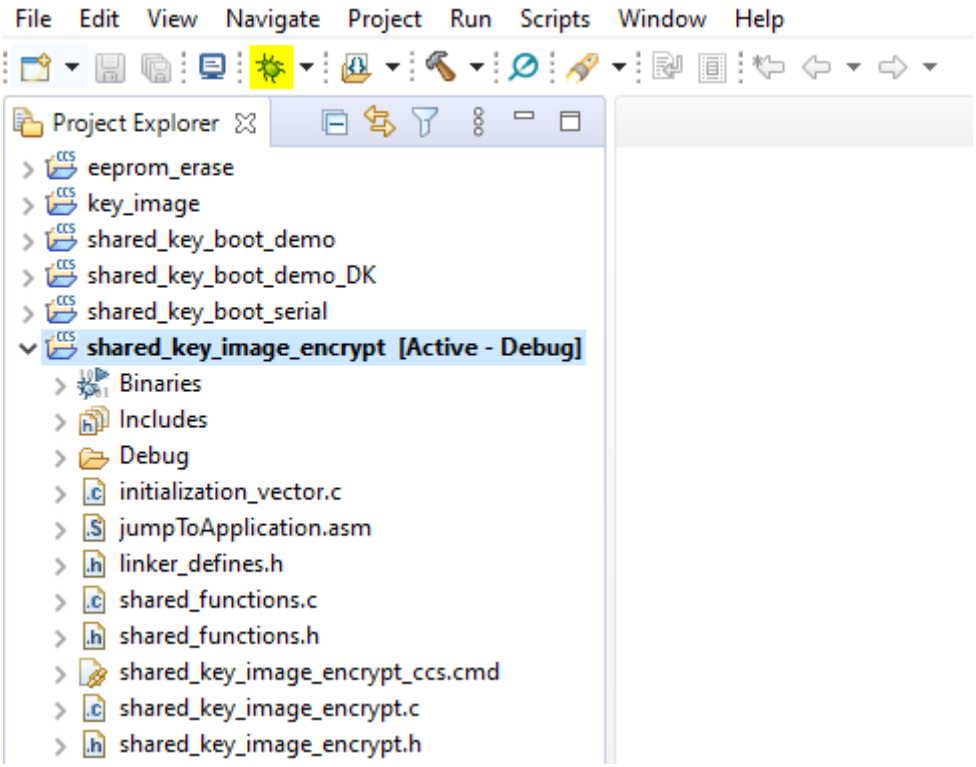

图 2-5. 执行 *shared_key_image_encrypt* 程序

2. 确定串行端口已由 PC 分配给 LaunchPad 的虚拟串行端口。这可以通过在 PC 上运行器件管理器找到。在图 2-6 中，LaunchPad 被分配给了 COM19。

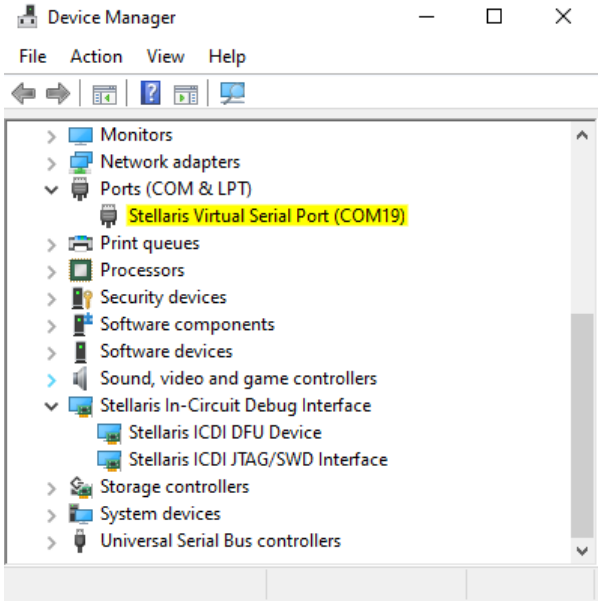

图 2-6. 识别 COM 端口

3. 打开支持 XMODEM 通信的终端仿真器。本例中使用了“Tera Term”。通过串行连接到 LaunchPad 的已识别的 COM 端口。

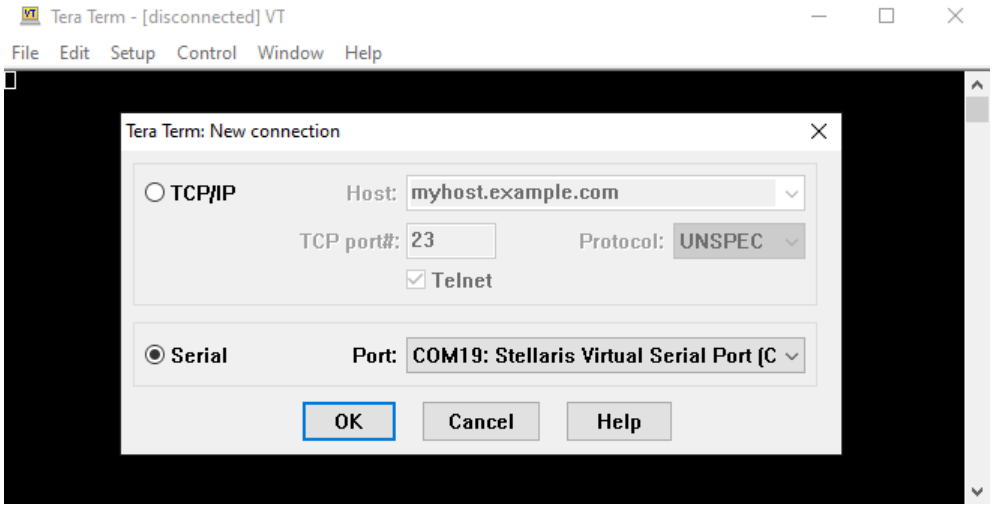

图 2-7. 在 Tera Term 中选择 COM 端口

4. 将终端仿真器配置为使用 460800 波特、8 位数据、1 个停止位和无奇偶校验的串行端口。

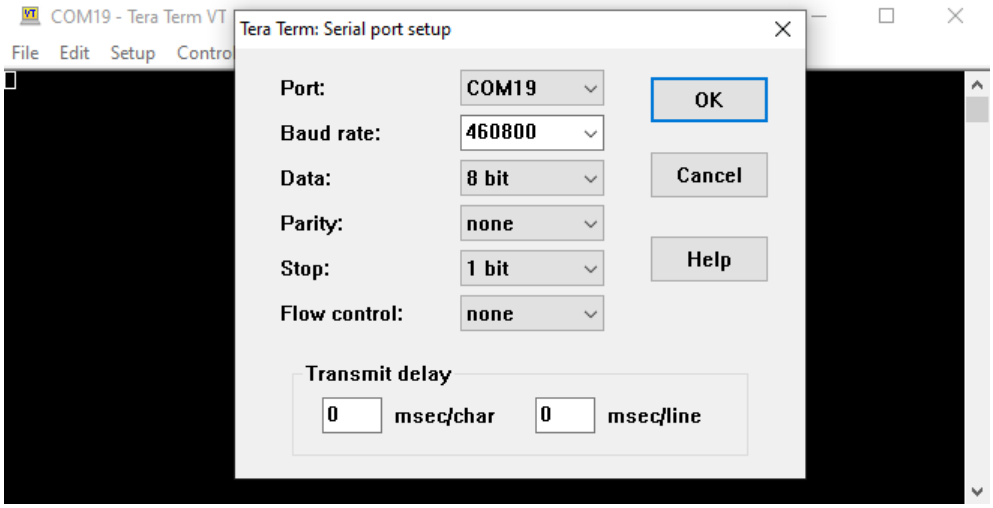

图 2-8. 在 Tera Term 中配置 COM 端口

5. 通过点击 Code Composer Studio 中的绿色箭头或按 **F8** 来执行 *shared_key_image_encrypt* 程序。

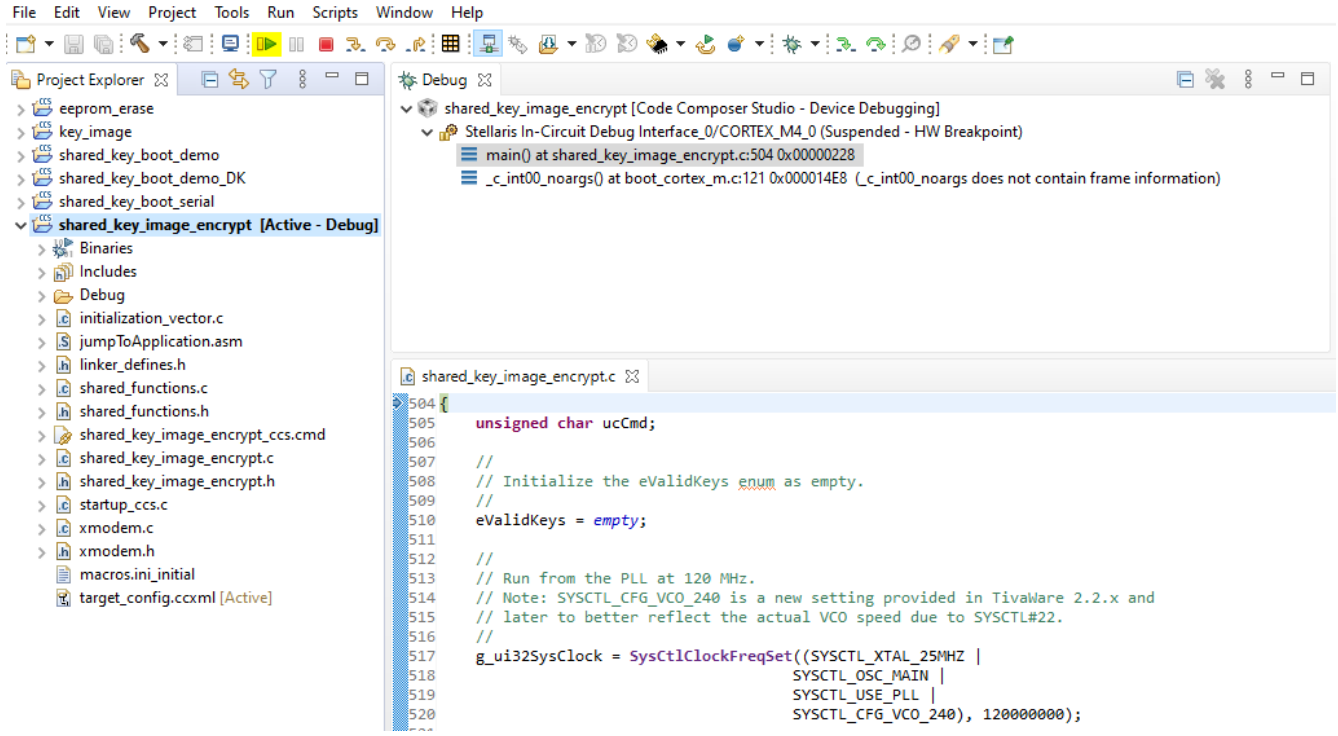

图 2-9. 执行 *shared_key_image_encrypt* 程序

6. 现在终端上应显示以下菜单。

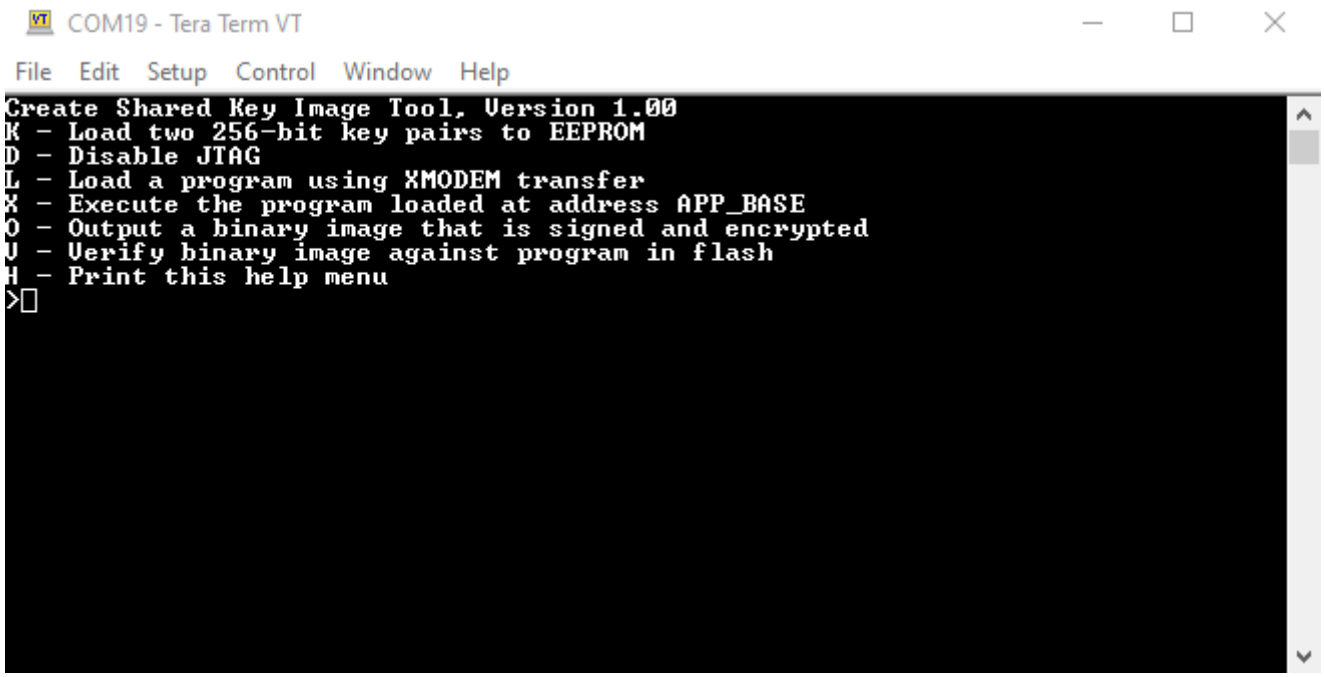

图 2-10. *shared_key_image_encrypt* 命令菜单

7. 要创建由引导加载程序使用的签名和加密映像，请执行以下步骤：
   a. 使用 XMODEM 协议，将文件 `key_image.bin` 从项目 ***key_image*** 的调试子目录传输到器件上。按如下步骤使用 Tera Term：
      i. 在终端仿真器中键入 K（不区分大小写）。
      ii. 选择“File”->“Transfer”->“XMODEM”->“Send”。
      iii. 浏览到 `key_image\Debug\key_image.bin`，然后选择“Open”。

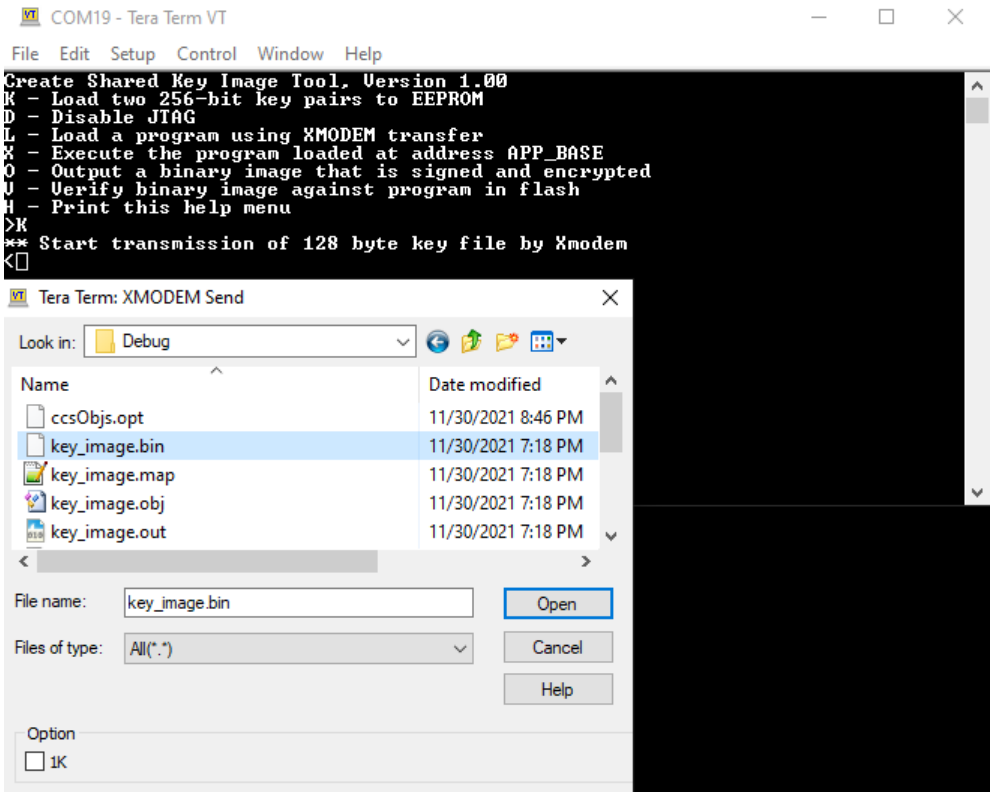

图 2-11. 加载密钥映像

   iv. 终端现在应显示 **Keys successfully programmed into EEPROM**。

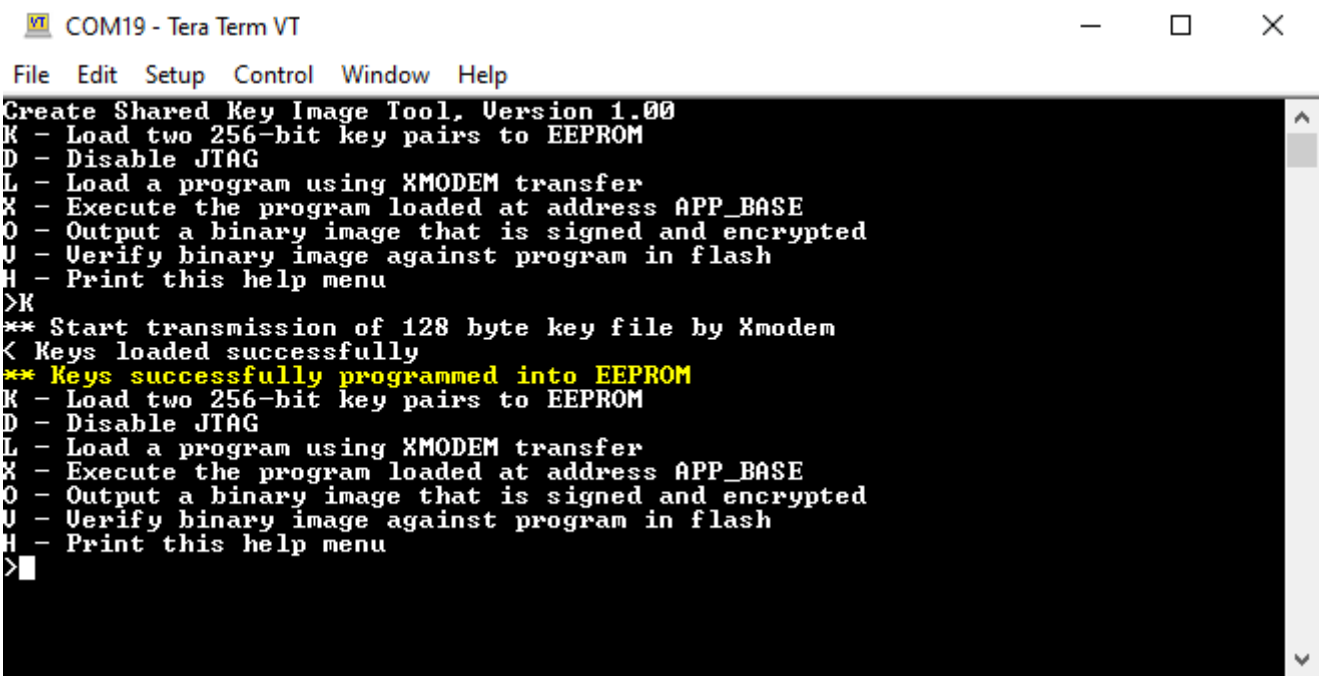

图 2-12. 已成功加载密钥

b. 将应用程序映像传输到器件中。按如下步骤使用 Tera Term：
   i. 在终端仿真器中键入 L。
   ii. 选择“File”-> “Transfer”-> “XMODEM”-> “Send”。
   iii. 浏览到 `shared_key_boot_demo\Debug\shared_key_boot_demo.bin`，然后选择“Open”。

   对于一个完整的 1MB 闪存器件，传输最多可能需要一分钟。完成后，终端上将显示消息 **File programming complete**。

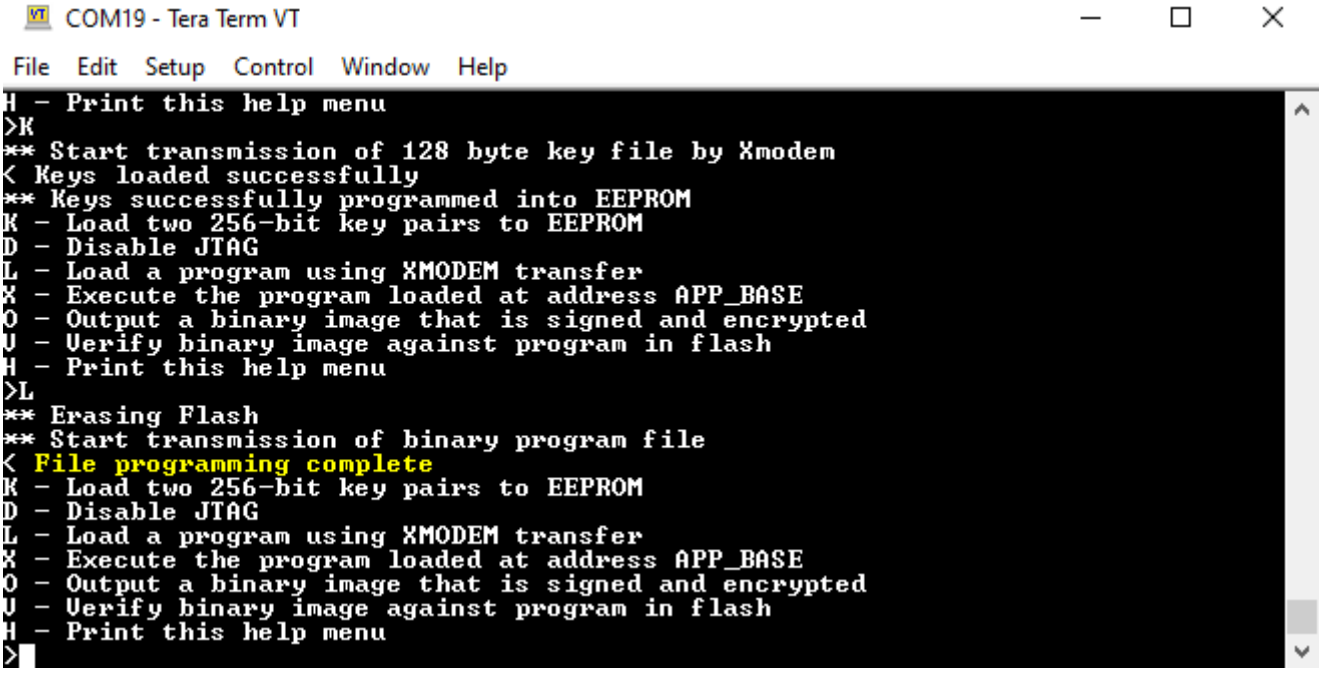

图 2-13. 加载程序映像

c. 输出二进制映像的签名和加密版本。按如下步骤使用 Tera Term：
   i. 在终端仿真器中键入 O。

ii. 选择“File”-> “Transfer”-> “XMODEM”-> “Receive”。
iii. 选择一个输出文件名，例如：

`shared_key_boot_demo\Debug\shared_key_boot_demo_encrypted.bin`，然后选择“Save”。

d. 或者，可以通过以下方式，根据存储在闪存中的未加密映像验证刚刚创建的加密文件：
i. 在终端仿真器中键入 V。
ii. 选择“File”-> “Transfer”-> “XMODEM”-> “Send”。
iii. 浏览到 `shared_key_boot_demo\Debug\shared_key_boot_demo_encrypted.bin`，然后选择“Open”。
iv. 对于一个完整的 1MB 闪存器件，传输大约需要一分钟。完成后，如果加密文件的哈希值验证成功，终端会显示消息 **File verification complete**。

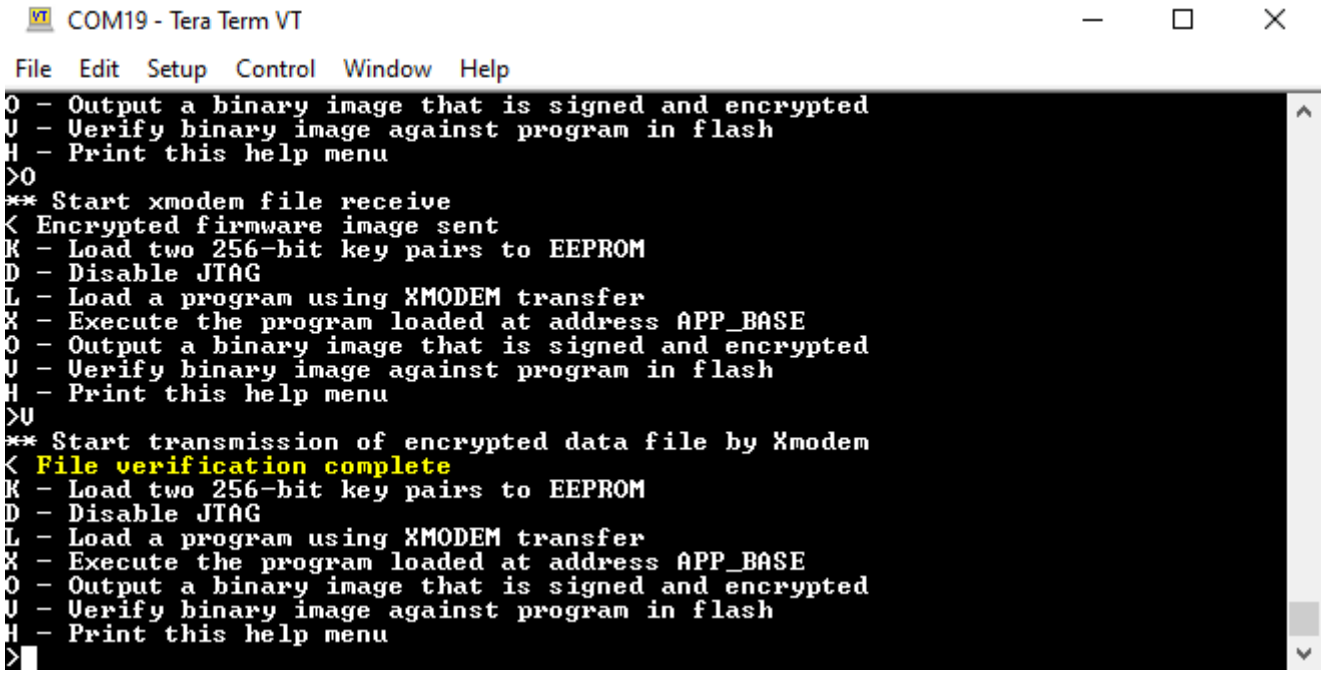

图 2-14. 验证程序映像

e. 或者，可以通过以下方式执行已编程到器件中的应用程序代码：
i. 在终端仿真器中键入 X。
ii. 终端上将显示消息 **Hash value authenticated**，LaunchPad 上的 LED D1 将闪烁。
iii. 使用 Code Composer Studio 或 LaunchPad 上的重置按钮重置 LaunchPad，终端将指示程序返回 *shared_key_image_encrypt* 工具。

2.5 运行共享密钥串行引导加载程序

尽管在非安全环境中可以对引导加载程序进行编程，但会导致初始化矢量暴露的风险。因此，最好的方法是在安全环境中同时对引导加载程序和密钥进行编程。如果在第一个扇区中对引导加载程序进行编程，而在 `APP_BASE` 的扇区中对密钥的映像进行编程，引导加载程序将把密钥复制到 EEPROM 中，然后擦除 `APP_BASE` 的扇区。如果使用“发布”版配置，引导加载程序会对其自身进行写保护并禁用 JTAG。

引导加载程序的“调试”版配置不应在现场发布。他人可以轻松使用仿真器连接到 JTAG 端口并编写公开密钥的代码。他们随后可以制作自己的程序，用这些密钥加载。

用户应考虑是否在其电路板上外露 JTAG 引脚。使用 ROM 引导加载程序对初始引导加载程序和密钥进行编程是可以实现的。因此，不需要访问 JTAG。使用引导加载程序的“发布”版配置将禁用 JTAG 接口，但他人仍可使用恢复锁定的微控制器方法来完全重置器件。这将擦除 EEPROM 中的引导加载程序、应用程序代码和密钥。之后，他们可以将自己的代码编程到该器件中。这与他们拆焊 TM4C 器件并用新器件进行替换的效果相同。对于球

栅阵列（BGA）器件，可以通过不对 JTAG 球添加布线来隐藏 JTAG。对于四方扁平封装（QFP）器件，即使没有对引脚布线，它们也会暴露在外。

无法访问 JTAG 引脚的主要缺点是无法从外部恢复器件。因此，用户必须确保，在重新进入引导加载程序之前，需要通过一个经过验证的过程。此外，对于故障器件，如果不将其从印刷电路板上移除，就无法进行分析。

2.5.1 对引导加载程序进行编程

2.5.1.1 擦除现有代码和密钥

要模拟真实的生产环境，可以从一个完全擦除的器件开始。有两种方法可以实现这一点：

2.5.1.1.1 使用 Code Composer Studio 擦除闪存和 EEPROM

1. 要擦除闪存，请从下拉菜单中选择“Tools”->“On-Chip Flash”。输入正确的晶体频率。对于 EK-TM4C129EXL LaunchPad 开发套件，其为 25MHz。在擦除部分，确保选中“Entire Flash”，然后点击“Erase Flash”按钮。

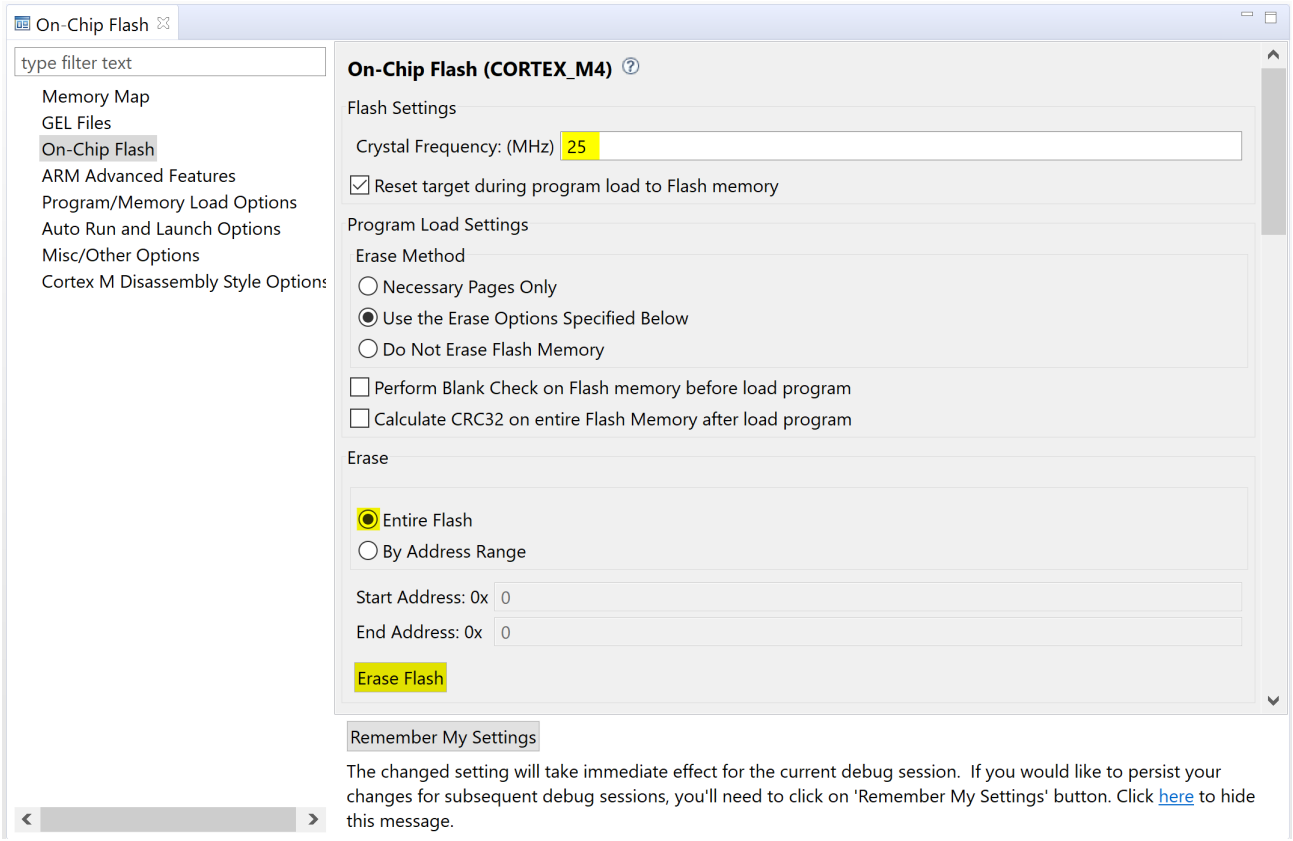

图 2-15. 使用 Code Composer Studio 擦除闪存

2. 要擦除 EEPROM，请加载并执行 *eeprom_erase* 程序，该程序已与本文档的配套资料一起加载到工作区中。从下拉菜单中选择“Run”->“Load”->“Select Program to Load”。然后点击“Browse project”。选择文件 `eeprom_erase\Debug\eeprom_erase.out`，然后点击“OK”。
3. 再次点击“OK”以加载并执行 *eeprom_erase* 程序。

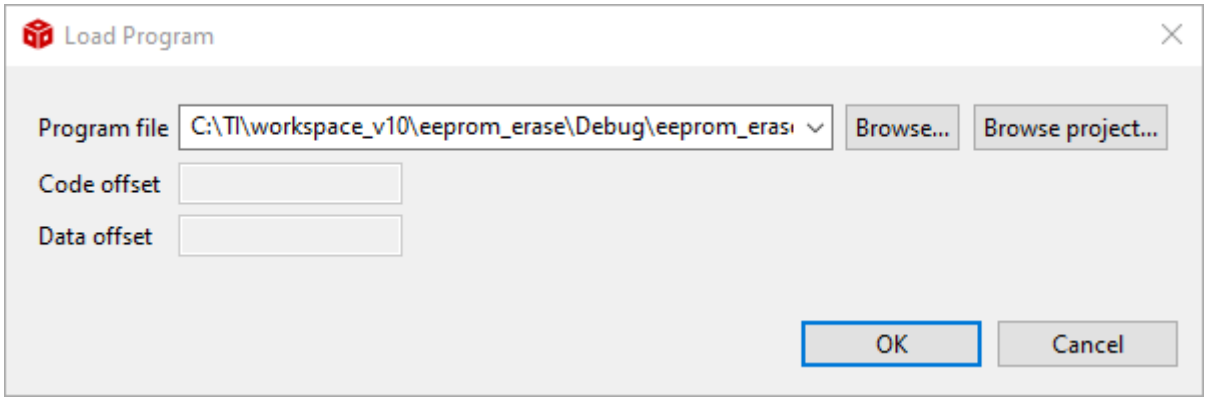

图 2-16. 加载 eeprom_erase.out

4. LaunchPad 上的两个 LED 灯将闪烁，表示程序运行并擦除了 EEPROM。

2.5.1.1.2 使用解锁程序擦除闪存和 EEPROM

解锁程序会将所有闪存、EEPROM 和器件设置恢复到出厂状态。这会导致存储在 LaunchPad 的用户寄存器中的 MAC 地址被擦除。建议在解锁该器件之前记录 MAC 地址，以便后续可以恢复。

通过 JTAG 接口使用 TM4C12x 器件 的*执行解锁序列* 部分中介绍了使用 LM Flash Programmer 的解锁程序。

2.5.1.2 使用 ROM 引导加载程序对共享密钥引导加载程序进行编程

现在闪存已被擦除，LaunchPad 正在执行 ROM 引导加载程序。前面关于共享密钥引导加载程序的示例使用了串行接口。此 ROM 引导加载程序示例将使用相同的串行接口来演示如何将共享密钥引导加载程序加载到空白器件中。

备注

请确保 TeraTerm 终端窗口与串行端口断开连接，并且 LaunchPad 未连接到 Code Composer Studio。USB 电缆仍必须保持连接。

1. 如下所示，打开 LM Flash Programmer 并设置配置选项卡上的项目，选择与 LaunchPad 关联的 COM 端口。

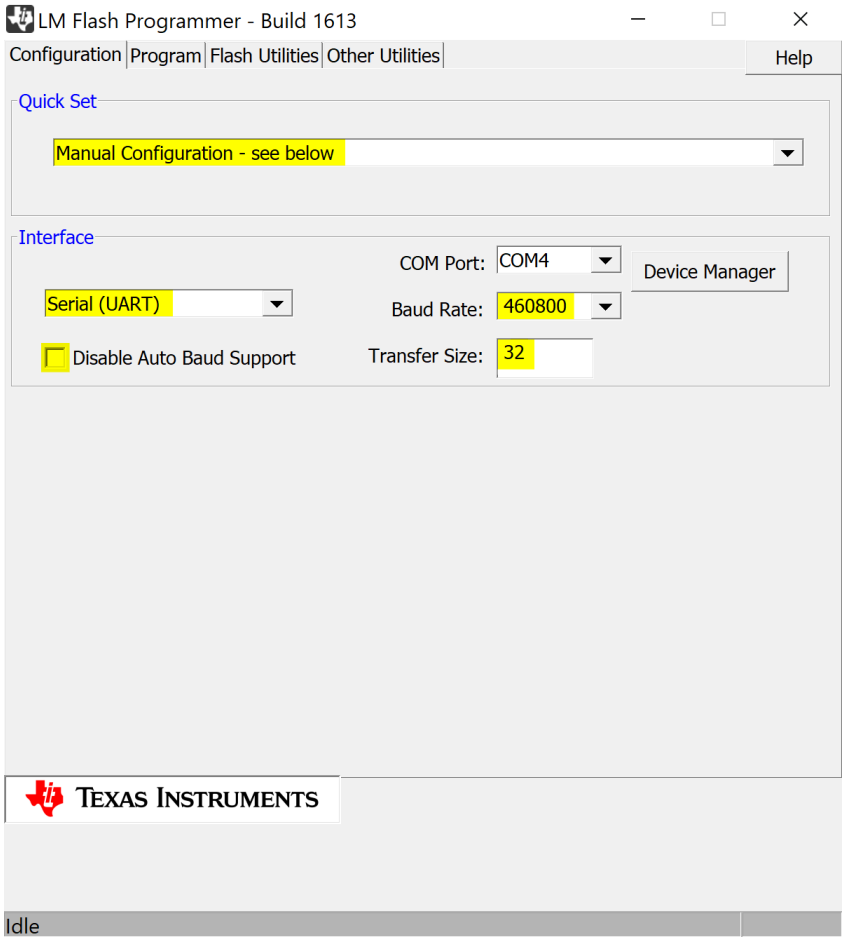

图 2-17. 配置 LM Flash Programmer 以对引导加载程序进行编程

2. 然后，在“Program”选项卡上，浏览到 `Debug_wKey\shared_key_boot_serial.bin` 文件。确保选中“Reset MCU After Program”并将“Program Address Offset”设置为 0。

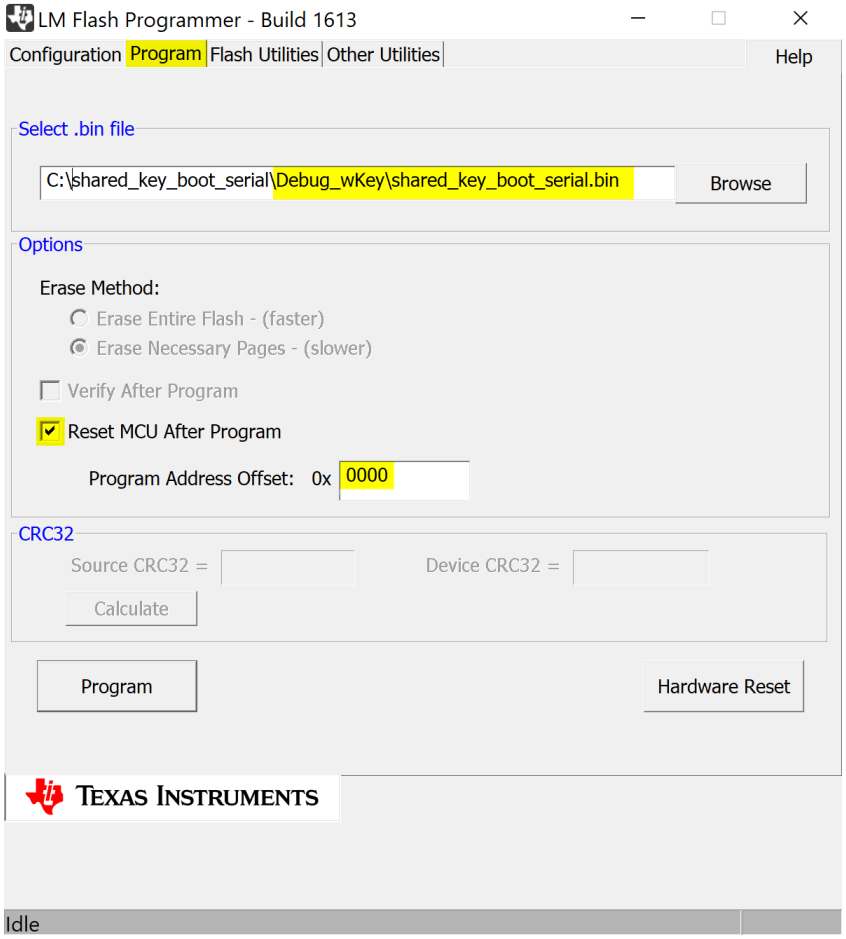

图 2-18. 使用 LM Flash Programmer 对引导加载程序进行编程

3. 点击“Program”按钮。

编程完成后，引导加载程序将执行并将密钥复制到 EEPROM 中，然后再从闪存中擦除密钥。如果使用 `Release_wKey\shared_key_boot_serial.bin` 文件而非调试版本，引导加载程序将对闪存的第一个扇区进行写保护并锁定 JTAG。

2.5.2 使用共享密钥引导加载程序对应用程序代码进行编程

现在可以使用 LM Flash Programmer 将加密的应用程序代码编程到器件中。由于密钥现在处于隐藏状态，可以在现场或工厂中安全性稍差的地方完成此操作。

1. 在 LM Flash Programmer 的配置选项卡上，必须选中“Disable Auto Baud Support”，并且必须将波特率设置为文件 `bl_config.h` 中选择的值。本例中使用了 460800。

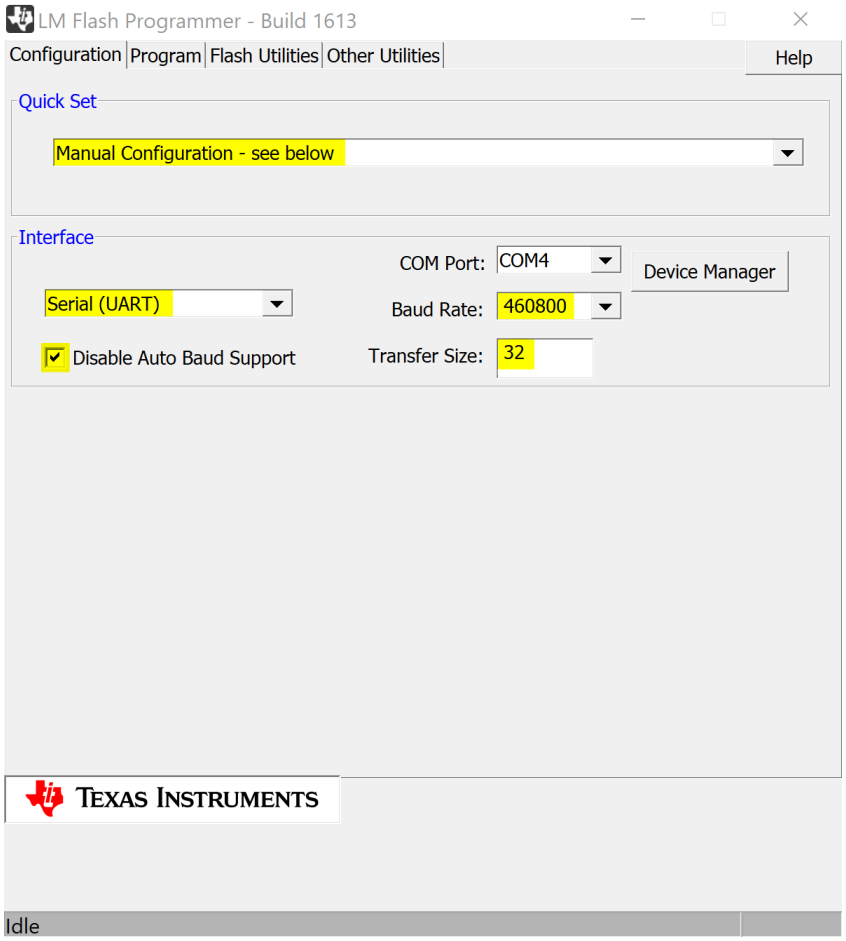

图 2-19. 配置 LM Flash Programmer 以对应用程序代码进行编程

2. 在“Program”选项卡上，选择加密的应用程序文件。本例中使用了 `shared_key_boot_demo\Debug\shared_key_boot_demo_encrypted.bin`。

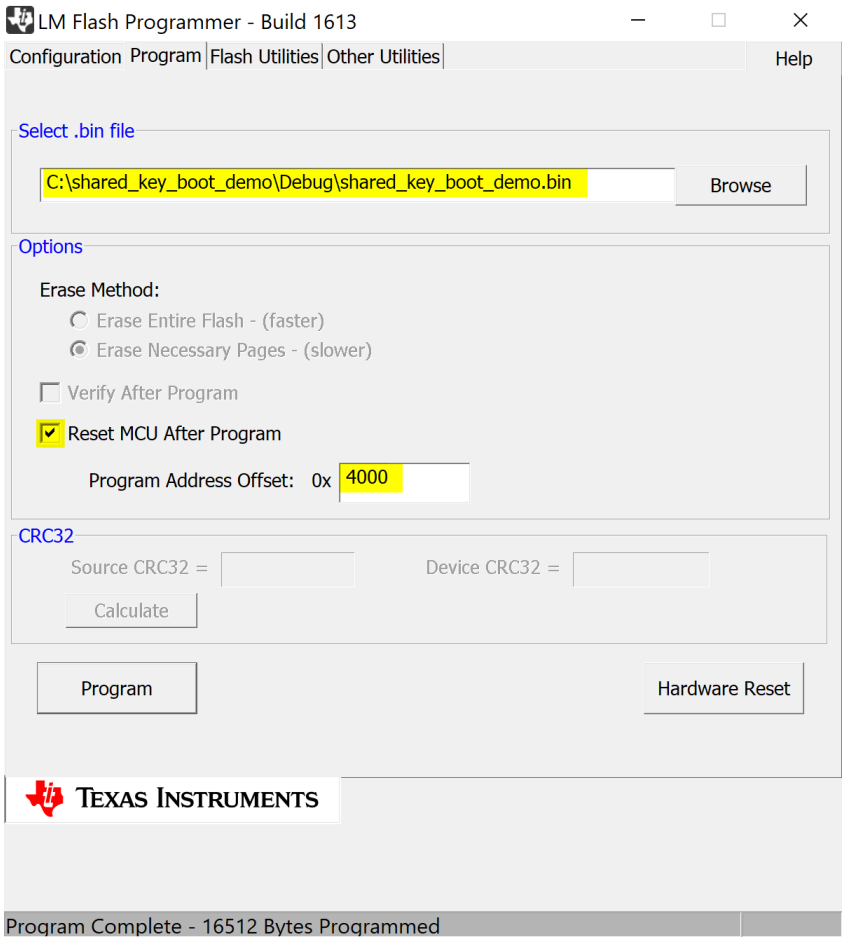

图 2-20. 使用 LM Flash Programmer 对应用程序代码进行编程

3. 点击“Program”按钮。

在 1MB 器件上对应用程序代码空间进行编程大约需要 100 秒。*shared_key_boot_demo* 应用程序运行时，LED1 会闪烁。

2.6 返回引导加载程序

在此示例中，有两种方法可以返回引导加载程序。第一种方法是在引导加载程序中实现的。复位结束后，引导加载程序将检查 LaunchPad 上的 *SW2* 按钮是否被按下。在按下复位按钮的同时长按 *SW2* 。LED1 将停止闪烁，引导加载程序将准备好加载新的加密应用程序代码。只需在不按下 *SW2* 的情况下再次按下复位按钮，应用程序代码将再次运行。

返回引导加载程序的第二种方法是在应用程序本身中实现的。这个 *shared_key_boot_demo* 应用程序代码将读取 LaunchPad *SW1* 按钮。如果长按一秒钟，应用程序代码将调用引导加载程序。

3 总结

对系统实施有效的保护措施继续成为系统设计人员需要优先考虑的事项。本应用报告和相关的配套资料旨在提供可考虑用于 TM4C 微控制器的合理选项。利用 EEPROM 进行密钥存储、使用哈希签名验证固件映像以及锁定 JTAG 访问等技术，都可以增加对系统的保护，使其免受单纯或意外覆盖代码映像行为的影响。这些概念适用于所有引导加载程序选项，只要重点是确保映像传输能够以验证所有数据包都已正确接收的方式进行。虽然本应用报告的重点集中在具有硬件 AES 加密外设的器件上，但同样的概念可应用于所有 TM4C 器件，前提是使用软件 AES，弊端是代码空间需要额外的成本，引导加载过程的执行速度也比较慢。最终，定义合适的系统保护所需的安全措施标准及评估正确的实施以满足这些要求，都取决于系统设计人员。

重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265