Application Note

C2000 器件上的安全引导

Pramod Prabhakara, Karthik Rajakumar, Christopher Chiarella, Matt Kukucka, and Ronak Harsora

摘要

本应用报告讨论了如何使用 F2838x 器件上的安全闪存启动特性在实际代码执行之前通过附加的启动代码身份验证安全层从闪存执行应用启动。本文档还适用于 F28003x、F280013x、F280015x、F28P55x、F28P551x 和 F28P65x 器件系列。

内容

插图清单

表格清单

商标

C2000™ is a trademark of Texas Instruments.
Arm® is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.
所有商标均为其各自所有者的财产。

English Document: SPRACT3
Copyright © 2026 Texas Instruments Incorporated

1 简介

TMS320F2838x 是一款功能强大的 32 位浮点实时微控制器，专为高级闭环控制应用（如工业电机驱动器、光伏逆变器和数字电源、电动汽车和运输以及感应和信号处理）而设计。该器件支持双核 C28x 架构，并带有可减轻关键通信任务负担的新型连接管理器 (CM)，显著提升了系统性能。虽然这些实时微控制器的硬件效率支持强大的应用，但在创建具体的独特应用程序时使用的是代码。该器件内置的双代码安全模块 (DCSM) 可帮助您创建安全解决方案，而模块中包含的安全启动特性可增强系统能力，以防通过运行您的应用程序来进行未经授权的更新。

DCSM 采用双安全区域概念，因此在闪存一次性可编程 (OTP) 区域中编程的安全配置决定了安全资源（闪存扇区和 RAM）的 Zone1/Zone2/Unsecure 分配情况。除此之外，Zone1/Zone2 资源也可配置为 EXEONLY，从而只允许从该区域执行代码。

有关更多详细信息，请参阅 *TMS320F2838x 技术参考手册*中的 *DCSM* 一章 [1]。

2 安全闪存启动概述

与应用程序闪存启动相关的 DCSM 特性之一是能够在执行闪存中的用户应用程序代码之前对其进行身份验证。这样可确保应用程序代码在被编程到闪存存储器后未被篡改，以此确定应用程序代码的完整性。当应用于 Zone1 EXEONLY 闪存扇区时，此特性可用作关键用户应用程序代码的附加安全层。除了传统的闪存启动选项外，安全闪存启动特性还提供了一组额外的启动选项。

安全闪存启动是通过使用 128 位 AES-CMAC 身份验证算法实现的，该算法在会返回通过/失败状态的应用程序代码内容上运行，并且只有在身份验证成功时才继续执行应用程序代码。表 2-1 概述了此特性在器件不同子系统上的情况。每个 CPU 子系统的 BootROM 发起针对该子系统应用代码前 16KB 的身份验证，这称为“主要安全启动”。超出每个 CPU 子系统前 16KB 的应用程序代码的身份验证称为“扩展安全启动”。此过程可由预先进行了身份验证的应用程序代码选择性地发起。

由于在安全闪存启动期间会执行 CMAC 身份验证算法，该启动序列与正常（非安全）闪存启动相比需要更多的时间才能进入用户应用程序。请注意，与 CPU1 或 CPU2 安全闪存启动实现方案相比，器件 CM 内核安全闪存启动需要的时间更少，因为 CM 会使用硬件 AES 加速器。

表 2-1. 器件上的安全闪存启动概述

子系统（内核）	安全启动特性	CMAC 算法实现方案	对闪存启动代码前 16KB 进行身份验证所需的额外时间
CPU1 SS (C28_1)	是	软件（安全 ROM 实用程序）+ AES ROM 表	~400ms（在 INTOSC 上以 10MHz 运行）
CPU2 SS (C28_2)	是	软件（安全 ROM 实用程序）+ AES ROM 表	~20ms（在 PLL 上以 200MHz 运行）
CMSS (CM4)	是	软件（安全 ROM 实用程序）+ 硬件 AES 加速器	~6ms（在 PLL 上以 125MHz 运行）

English Document: SPRACT3
Copyright © 2026 Texas Instruments Incorporated

3 CMAC 身份验证

基于密码的消息身份验证代码 (CMAC) 是一种基于 AES 的身份验证算法，它根据输入数据块构造一种身份验证标签。每次会以 128 位将输入数据块与 128 位 CMAC 密钥一起馈入到加密引擎/软件（根据 CPU 子系统而定）中。该密钥位于 CPU1 USER OTP 的 0x78018-0x7801F 位置，并由器件所有内核上的 CMAC 身份验证算法使用。加密引擎/软件对整个输入数据块运行 CMAC 算法，并生成最终的 128 位身份验证标签。

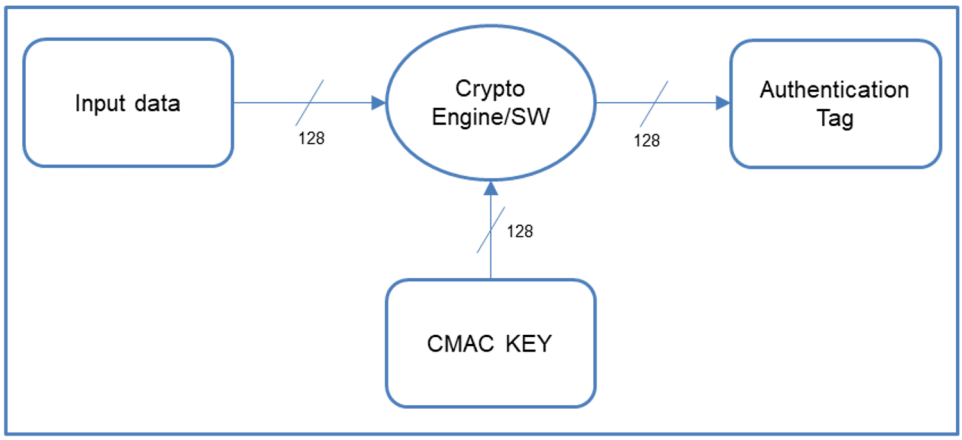

图 3-1. CMAC 工作原理

4 安全闪存启动选项

表 4-1 所示为三个内核上可用的不同主要安全闪存启动选项及其相应的配置。由应用程序代码启动的“扩展安全启动”在节 7 中有相关说明。更多详细信息，请参阅《*TMS320F2838x 技术参考手册*》[1] 中的 *ROM 代码和外设启动* 一章。

表 4-1. 安全闪存启动模式配置详细信息

安全启动选项 (1)	BOOTDEFx/ BOOTMODE 值 (2)	闪存入口点 (3)	CPU1/CPU2 入口地址	CPU1/CPU2 128 位黄金 CMAC 标签位置	CM 入口地址	CM 128 位黄金 CMAC 标签位置
选项 0	0x0A	扇区 0	0x00080000	0x00080002	0x00200000	0x00200004
选项 1	0x2A	扇区 4	0x00088000	0x00088002	0x00210000	0x00210004
选项 2	0x4A	扇区 8	0x000A8000	0x000A8002	0x00250000	0x00250004
选项 3	0x6A	扇区 13	0x000BE000	0x000BE002	0x0027C000	0x0027C004

(1) 可为 CPU1/CPU2/CM 分别选择安全启动选项。
(2) 对于 CPU1，BOOTDEFx 字段是 Zx-BOOTDEF-LOW/Zx-BOOTDEF-HIGH CPU1 USER OTP 存储器位置的一部分。对于 CPU2/CM，BOOTMODE 字段是 CPU1TOCPU2IPCBOOTMODE/CPU1TOCMIPCBOOTMODE 寄存器的一部分，分别由 CPU1 应用程序代码填充。
(3) 安全启动特性仅适用于 Zone1，因此所选闪存扇区必须配置为 Zone1-EXEONLY。此外，无论扇区大小如何，“主要安全闪存启动”仅基于所选扇区的前 16KB 运行。例如，扇区 4 和扇区 8 各为 64KB，但仅考虑前 16KB。

English Document: SPRACT3
Copyright © 2026 Texas Instruments Incorporated

5 安全闪存启动流程

在器件上实现安全闪存启动的过程包含两步：

1. **生成身份验证标签** – 此过程是在器件外部创建映像期间进行的。
   a. C2000™ 或 Arm® 十六进制实用程序使用输入 CMACKEY 以及能够为黄金 CMAC 身份验证标签保留存储空间的 CMAC 应用程序数据结构来对闪存启动代码映像运行 CMAC 算法。更多有关十六进制实用程序的详细信息，请参阅 [3] 和 [4]。
   b. 生成的黄金 CMAC 标签嵌入在十六进制文件中由表 4-1 指定的位置。
   c. 十六进制映像（现在包含黄金 CMAC 标签）被编程到闪存的相应扇区中。
   d. 相应的安全闪存启动模式应根据表 4-1 进行选择并编程到 CPU1 USER OTP 中。
2. **对闪存中的应用程序启动代码进行身份验证** – 此过程在器件内部执行安全闪存启动时进行。
   a. BOOTDEFx/BOOTPINCONFIG 字段被配置为根据表 4-1 选择安全闪存启动选项（在复位时，器件会启动）并对指定的闪存扇区执行 CMAC 算法。
   b. 将由 CMAC 算法生成的标签与位于预编程位置的黄金 CMAC 标签进行比较。
   c. 标签匹配成功后，启动过程将分支到经过身份验证的闪存代码并开始执行。
   d. 如果标签匹配失败，则会对 CPU1/CPU2/CM 采取不同的操作：
      i. 如果是 CPU1，则会将器件复位（代码保持在循环中，并且在看门狗到期时会自动发出 XRSn）。
      ii. 如果是 CPU2，则会在 CPU2TOCPU1IPCBOOTSTS 寄存器中设置安全启动失败标志，将 IPC 命令与安全闪存 CMAC 错误代码一起发送到 CPU1，并且 CPU2 启动代码会循环等待 CPU1 以采取必要操作。还会在 CPU2 的 0x0000 0002 地址位置捕获 CPU2TOCPU1IPCBOOTSTS 寄存器的副本。
      iii. 如果是 CM，则会在 CMTOCPU1IPCBOOTSTS 寄存器中设置安全启动失败标志，将 IPC 命令与安全闪存 CMAC 错误代码一起发送到 CPU1，并且 CM 启动代码会循环等待 CPU1 以采取必要操作。还会在 CM 的 0x2000 0000 地址位置捕获 CMTOCPU1IPCBOOTSTS 寄存器的副本。

备注

在计算映像上的身份验证标签以及对映像进行身份验证时，CMAC 算法会将包含黄金标签的存储器地址视为全 1。

English Document: SPRACT3
Copyright © 2026 Texas Instruments Incorporated

6 C2000Ware 示例详细信息

C2000Ware [2] 通过一个示例来说明应用程序的安全闪存启动设置。该示例包括针对每个内核的安全闪存启动应用程序工程。该示例还详细说明了如何对安全闪存启动入口地址 + 16KB 之外的闪存代码进行身份验证。节 7 中提供了关于此定制闪存范围身份验证功能的更多详细信息。该示例假设要进行身份验证的闪存扇区已预先配置为 Zone 1 EXEONLY 并使用默认 CMACKEY 进行身份验证。有关在 CPU1 USER OTP 中对定制 CMACKEY 和其他 DCSM 设置进行编程的详细信息，请参阅 [1] 和 [2]。

C2000Ware 位置：<C2000Ware_安装目录>/driverlib/f2838x/examples/c28x/boot

工程名称：

- boot_ex1_cpu1_cpu2_cm_secure_flash_cpu1
- boot_ex1_cpu1_cpu2_cm_secure_flash_cpu2
- boot_ex1_cpu1_cpu2_cm_secure_flash_cm

包含的文件：

- 源文件 – 包含主应用程序代码
  - 示例：boot_ex1_cpu1_cpu2_cm_secure_flash_cpu1.c
- 十六进制链接器命令文件 – 向 c2000 或 arm hex 实用程序提供整个闪存长度的详细信息
  - 示例：boot_ex1_flash_hex_lnk_cpu1.cmd
- CMAC 密钥文本文件 – 为 c2000 或 arm hex 实用程序提供用户 CMAC 密钥
  - 示例：boot_ex1_user_cmac_key.txt
  - 更多有关 cmac_key 格式的详细信息，请参阅 [3] 和 [4]。

如何运行该示例：

1. 将应用程序加载到 CPU1 闪存中（以及 CPU2 和 CM 应用程序）。
   a. 加载 *.hex 文件，而不是 *.out 文件
2. 断开连接并仅重新连接到 CPU1。
3. 要将器件配置为在启动时执行安全闪存启动：
   a. 仿真启动（建议用于示例/开发）
      i. 在 CCS 存储器窗口中，将 BOOTPINCONFIG 位置 (0x0D00) 设置为 0x5AFFFFFF，并将 BOOTDEF 位置 (0x0D04) 设置为 0x0000000A
   b. 独立启动（建议用于部署）
      i. 对与 BOOTPINCONFIG 和 BOOTDEF 相对应的 CPU1 USER OTP 位置进行编程。若要了解更多信息，请参阅 [1]。
4. 通过 CCS 重置 CPU1，然后点击“Resume”。
5. 观察 controlCARD 上的 LED 指示灯是否指示已成功。
   a. 当所有三个内核都成功地安全启动并对它们的完整闪存内容进行身份验证时，三个 LED 指示灯（每个内核一个）将闪烁。
6. 对于不使用 controlCARD 的情况，请注意以下基于 CPU 子系统的 GPIO 切换开关：
   a. CPU1 – GPIO31
   b. CPU2 – GPIO34
   c. CM – GPIO145

English Document: SPRACT3
Copyright © 2026 Texas Instruments Incorporated

生成黄金 CMAC 标签的应用程序代码要求：

用于安全闪存启动选项 0 的 CPU1/CPU2 黄金 CMAC 标签存储器分配

```
// CPU1/CPU2 的实现情况
#pragma RETAIN(cmac_sb_1)
#pragma LOCATION (cmac_sb_1, 0x080002)
const char cmac_sb_1[8] = {0};
```

用于安全闪存启动选项 0 的 CM 黄金 CMAC 标签存储器分配

```
#pragma RETAIN(cmac_sb_1)
#pragma LOCATION (cmac_sb_1, 0x00200004)
const uint8_t cmac_sb_1[16] = {0};
```

常量字符/无符号整数定义会为黄金 CMAC 标签分配存储器。如需更多信息，请参阅节 6 中的示例：

- 变量命名必须为以下之一：cmac_sb_1、cmac_sb_2、cmac_sb_3、cmac_sb_4
  - 在《*TMS320C28x 汇编语言工具用户指南*》[3] 中可找到更多有关 C28x 的详细信息
  - 在《*ARM 汇编语言工具用户指南*》[4] 中可找到更多有关 CM 的详细信息
- 使用 LOCATION pragma 为 CMAC 黄金标签指定预期身份验证范围内的地址。对于 CPU1/CPU2，此地址必须为入口地址 + 2，对于 CM，此地址必须为入口地址 + 4。
- 保持变量初始化为零。

使用十六进制实用程序时的应用程序设置：

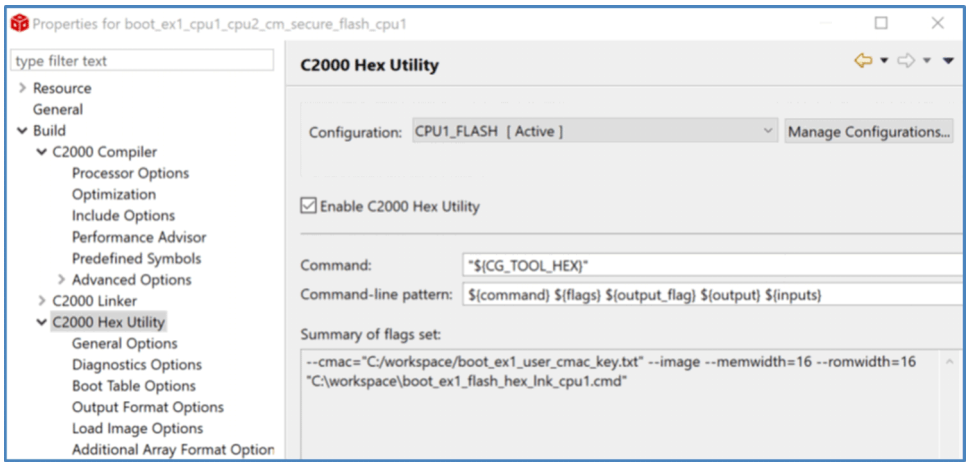

图 6-1. 为 CMAC 启用十六进制实用程序后的 CPU1 示例属性

每个内核工程都使十六进制实用程序能够生成黄金 CMAC 标签（请参阅图 6-1）。标签包括：

- “--cmac”，提供用户 CMAC 密钥文本文件的路径
- “--image”、“--memwidth”和“--romwidth”，其中设置了 mem/rom。CPU1/CPU2 的此设置应为 16，CM 的此设置应为 8
- 对应内核的闪存十六进制链接器命令文件的路径。

English Document: SPRACT3
Copyright © 2026 Texas Instruments Incorporated

7 对超过 16KB 的闪存代码进行身份验证

“安全闪存启动”仅对从入口地址开始的前 16KB 闪存扇区 进行身份验证。为了对闪存的其他扇区进行身份验证，用户应用程序必须直接调用安全闪存启动 CMAC API。

通过使用十六进制实用程序，它支持为四个闪存入口地址中的每一个 + 16KB 和 1 个定制闪存范围生成黄金 CMAC 标签。定制闪存范围可配置为能够在定制地址范围内进行 CMAC 身份验证。如有需要，此范围可以是所有闪存扇区的长度。

用于定制闪存范围身份验证的 CPU1/CPU2 应用程序 CMAC 结构

```
struct CMAC_TAG
{
   char tag[8];
   uint32_t start;
   uint32_t end;
}
```

用于完整闪存范围身份验证的 CPU1/CPU2 黄金 CMAC 标签存储器分配

```
#pragma RETAIN(cmac_all)
#pragma LOCATION (cmac_all, 0x087002)
const struct OMAC_TAG cmac_all = {{0}, 0x0, 0x0};
```

用于定制闪存范围身份验证的 CM 应用程序 CMAC 结构

```
struct CMAC_TAG
{
   uint8_t tag[16];
   uint32_t start;
   uint32_t end;
}
```

用于完整闪存范围身份验证的 CM 黄金 CMAC 标签存储器分配

```
#pragma RETAIN(cmac_all)
#pragma LOCATION (cmac_all, 0x00204004)
const struct OMAC_TAG cmac_all = {{0}, 0x0, 0x0};
```

创建一个名为“cmac_all”的结构，用于创建定制 CMAC 身份验证范围。

- 使用 LOCATION pragma 为 CMAC 黄金标签指定预期身份验证范围内的任何地址。（CMAC 黄金标签越接近身份验证起始地址，CMAC 算法的执行速度越快）。
- 让“tag”结构元素始终初始化为零。
- 初始化“start”和“end”结构元素以设置定制范围。如果两者都为零，则器件内核闪存的全部长度都将接受身份验证，包括“主要安全启动”的 16KB 存储器范围。
- 更多有关应用程序 CMAC 变量/结构的详细信息，请参阅 [3] 和 [4]。

CPU1 安全闪存 CMAC 身份验证 API

```
applicationCMACStatus = CPU1BROM_calculateCMAC (CMAC_AUTH_START_ADDRESS,
                                                CMAC_AUTH_END_ADDRESS,
                                                CMAC_AUTH_TAG_ADDRESS);
```

在应用程序中，将 F2838x 安全区域代码符号库包含到工程中（位于 C2000Ware [2] 中的 <C2000Ware_安装目录>/libraries/boot_rom/f2838x 位置），并调用相应内核的安全闪存启动 CMAC API。以上示例显示了在 CPU1 上进行的示例 API 调用。

- CMAC_AUTH_TAG_ADDRESS 应该与提供给 LOCATION pragma 的内容匹配
- CMAC_AUTH_START_ADDRESS 和 CMAC_AUTH_END_ADDRESS 应该与提供给“cmac_all”结构的内容匹配。此外，需要注意起始地址和结束地址应对齐到 128 位，这很重要。
- 例如，用于对 F2838x CPU1 闪存全部长度进行身份验证的起始地址和结束地址为：
  - 起始：0x00080000

English Document: SPRACT3
Copyright © 2026 Texas Instruments Incorporated

  - 结束：0x000C0000
- 然后应在用户应用程序中相应地检查和处理从安全闪存启动 CMAC API 返回的状态。
- 更多有关安全闪存 CMAC 身份验证 API 的详细信息，请参阅 [1] 中的 *ROM 代码和外设启动* 一章 。

备注

即使以上示例中的结束地址实际为 0x000BFFFF，也必须提供 0x000C0000，以便结束地址对齐到 128 位。

8 调试资源

表 8-1. 安全闪存启动调试场景

场景	行为
CPU1 中的安全启动失败	独立启动：器件复位。 仿真启动：CPU1 在地址范围 0x3FB13C – 0x3FB142 内停止
CPU2 中的安全启动失败	CPU2TOCPU1IPCBOOTSTS (1) 寄存器的 Bit21 将被设置。 CPU2 向 CPU1 发送带有安全闪存 CMAC 错误代码的 IPC 命令。
CM 中的安全启动失败	CMTOCPU1IPCBOOTSTS (2) 寄存器的 Bit21 将被设置。CM 向 CPU1 发送带有安全闪存 CMAC 错误代码的 IPC 命令。
CPU1 的安全启动是否成功运行？	位于 0x0000 0002 地址位置的 CPU1 BootROM 状态 Bits7:0 将反映 0x3。
CPU2 的安全启动是否成功运行？	CPU2TOCPU1IPCBOOTSTS (1) 寄存器的 Bits7:0 将反映 0x3。
CM 的安全启动是否成功运行？	CMTOCPU1IPCBOOTSTS (2) 寄存器的 Bits7:0 将反映 0x3。

1. 在位于 CPU2 的 0x0000 0002 地址位置也捕获了相同的信息。
2. 在位于 CM 的 0x2000 0000 地址位置也捕获了相同的信息。

9 其他信息和注意事项

- 虽然不推荐，但如果对 CPU2/CM 而不是对 CPU1 执行安全闪存启动，那么在将 CPU2/CM 解除复位之前，必须从 CPU1 为 Z1 OTP CMACKEY 执行虚拟加载。虚拟加载是通过读取 CPU1 USER OTP 中的 0x78018-0x7801F 位置来完成的。
- 同样，如果在没有对 CPU1 运行安全闪存启动模式的情况下使用 CMAC 身份验证 API，那么在调用这些 API 之前，必须从 CPU1 为 Z1 OTP CMACKEY 执行虚拟加载。
- 在对 CPU1 使用安全闪存启动时，建议不要对 BOOTDEF 表中配置的同一扇区使用正常（非安全）闪存启动模式。
- 若要对超过 16KB 的闪存代码进行身份验证：
  - 128 位黄金 CMAC 标签必须存储在执行计算的存储器地址范围内。
  - 黄金 CMAC 标签的起始地址必须与 32 位边界对齐。
- 启动模式设置位于一次性可编程 (OTP) 闪存中，因此建议在冻结与启动相关的配置之前使用仿真启动模式进行试验。更多有关仿真启动的信息，请参阅 *《TMS320F2838x 技术参考手册》* [1] 中的 *ROM 代码和外设启动* 一章。

English Document: SPRACT3
Copyright © 2026 Texas Instruments Incorporated

10 C2000 CMAC 算法与 OpenSSL 的对齐

C2000™ 十六进制实用程序 (hex2000) 会对消息数据（在 CMAC 计算之前）以及生成的身份验证标签（在嵌入闪存映像之前）应用一种专有的字节重排变换，本文档通篇将其称为“swapwords”。节 10.2 对此进行了详细介绍。

由于 OpenSSL 的 AES-128-CMAC 实现方案直接操作未经修改的连续字节流，不进行任何此类重排，因此在原始二进制文件上直接调用 *openssl dgst -mac* cmac 命令所产生的输出将与 C2000 生成的金标 CMAC 标签不匹配。本节介绍了使 OpenSSL 输出与 C2000 对齐所需的精确操作序列，使开发人员能够使用标准的开源加密工具独立验证或复现金标 CMAC 标签。

请注意、此对齐过程旨在用于开发和验证工作流程，并非替代 hex2000 实用程序来生成生产映像。有关 *hex2000 --cmc* 标志和密钥文件格式的详细信息，请参阅 [3] 和 [4]。

10.1 C28x 存储器和二进制文件字节顺序

C28x CPU 采用 16 位字寻址（不可按字节寻址）。每个 C28x 字为 16 位宽。在二进制文件中，每个 16 位字以小端字节序存储（低字节位于文件偏移较小处）。

32 位值在二进制文件中的显示方式

一个 C28x uint32_t 值 0xAABBCCDD 存储在字地址 N 处：

```
C28x word address N:   value = 0xCCDD  (low 16 bits)
C28x word address N+1: value = 0xAABB  (high 16 bits)

Binary file bytes:
  offset 2k+0: 0xDD  ← low  byte of word[N]
  offset 2k+1: 0xCC  ← high byte of word[N]
  offset 2k+2: 0xBB  ← low  byte of word[N+1]
  offset 2k+3: 0xAA  ← high byte of word[N+1]
```

因此 uint32_t 0xAABBCCDD 在二进制文件中显示为： DD CC BB AA

示例：codestart 分支指令

位于 C28x 地址 0x080000 的“LB 0x081BC8”指令：

```
C28x word 0x080000: 0x4800  → binary bytes: 00 48
C28x word 0x080001: 0x1BC8  → binary bytes: C8 1B

Binary file at offset 0: 00 48 C8 1B
```

10.2 闪存二进制字节顺序

hex2000 实用程序并非直接将闪存二进制文件提供给 AES-128-CMAC 算法。在计算身份验证标签之前，它会先应用 *swapwords* 变换，即交换二进制数据中每个 32 位组内的两个 16 位字。

对于二进制文件中每组 4 个连续字节 [b0, b1, b2, b3]，其中：

- [b0, b1] = C28x 字地址 N 处的 [低字节, 高字节]
- [b2, b3] = C28x 字地址 N+1 处的 [低字节, 高字节]

```
swapwords([b0, b1, b2, b3]) = [b2, b3, b0, b1]
```

此变换过程交换每个 32 位组内的两个 16 位字，同时保持每个字的字节按其自然（小端字节序）顺序排列。这相当于按逆序读取每对 C28x 字（地址 N+1 处的高位字在前，地址 N 处的低位字在后）。该操作是自逆的，即对任意输入应用该变换两次将得到原始数据。

对变换后的数据计算 AES-128-CMAC 标签后，hex2000 将同样的 swapwords 变换再一次应用到 16 字节的标签本身，然后将结果嵌入闪存映像中的金标 CMAC 标签位置处（从闪存组入口地址起的字节偏移 +0x4，对应 C28x

English Document: SPRACT3
Copyright © 2026 Texas Instruments Incorporated

字地址 +0x2 ）。为了生成与 hex2000 所产生标签逐字节完全相同的存储标签，在使用 OpenSSL 时必须复现这两次变换。

10.3 CMAC 密钥字节顺序

128 位 CMAC 密钥以四个 32 位寄存器（DCSM_Z1_CMACKEY0 至 DCSM_Z1_CMACKEY3）的形式预置在 DCSM Zone 1 OTP 中，其中 CMACKEY0 保存最高有效 32 位，CMACKEY3 保存最低有效 32 位。

hex2000 密钥文本文件格式按最高有效位到最低有效位的顺序连接这四个值，并添加“0x”前缀。OpenSSL 的 *-macopt hexkey:<32-hex-key>* 选项直接接受相同的 32 字符十六进制字符串，无需对字节进行任何额外重排。表 10-1 展示了 OTP 寄存器值、hex2000 密钥文件格式和 OpenSSL hexkey 参数之间的对应关系。

表 10-1. CMAC 密钥字节顺序对应关系

OTP 寄存器	值（示例）	角色
DCSM_Z1_CMACKEY0	0x2b7e1516	密钥位 [127:96] (MSW)
DCSM_Z1_CMACKEY1	0x28aed2a6	密钥位 [95:64]
DCSM_Z1_CMACKEY2	0xabf71588	密钥位 [63:32]
DCSM_Z1_CMACKEY3	0x09cf4f3c	密钥位 [31:0] (LSW)
hex2000 密钥文件	0x2b7e151628aed2a6abf7158809cf4f3c	单行，带 0x 前缀
OpenSSL hexkey	2b7e151628aed2a6abf7158809cf4f3c	相同值，无 0x 前缀

10.4 CMAC 输出对齐流程

使用 OpenSSL 复现 hex2000 金标 CMAC 标签是一个五步流程。各步骤如下所述。

第 1 步 — 获取签名前的二进制文件。

从 CMAC 签名前的 16KB（0x4000 字节）闪存二进制文件开始。在此状态下，字节偏移 0x4 处的 16 字节金标 CMAC 标签占位符 (cmac_sb_1) 必须初始化为全 0，这与 节 6 中描述的应用程序源代码要求一致。16KB 区域的其余部分应反映预期的闪存内容，未编程区域填充 0xFF。

第 2 步 — 屏蔽金标 CMAC 标签区域。

CMAC 金标签名存储在 CMAC 所覆盖的 16KB 区域内。为避免循环依赖（标签依赖于自身），在 CMAC 计算期间会将签名区域屏蔽为全 1 (0xFFFF)。C28x 器件上已擦除的闪存读取为全 1。通过屏蔽签名区域，CMAC 计算会将该签名位置视为已擦除的闪存，这与编程前闪存的初始状态一致。

因此，在计算 CMAC 之前，需将字节 0x4 至 0x13（含）替换为值 0xFF。无论使用 hex2000 还是 OpenSSL 进行计算，都必须执行此屏蔽步骤，这与 节 5 中描述的行为一致。

第 3 步 — 应用 swapwords 变换来生成 OpenSSL 输入。

对屏蔽后的整个 16KB 二进制文件应用 swapwords 字节重排。对于每组四个连续字节 [b0, b1, b2, b3]，将其替换为 [b2, b3, b0, b1]。将生成的 16KB 字节序列写入一个临时文件（例如 input_swapped.bin）。这个变换后的文件构成了 OpenSSL CMAC 命令的直接输入。

第 4 步 — 使用 OpenSSL 计算 AES-128-CMAC。

按如下方式调用 OpenSSL，并替换为相应的 32 字符十六进制密钥字符串：

用于对变换后的输入计算 AES-128-CMAC 的 OpenSSL 命令

```
openssl dgst -mac cmac \
             -macopt cipher:AES-128-CBC \
             -macopt hexkey:<32-hex-key> \
             -binary input_swapped.bin > aes_tag.bin
```

输出文件 aes_tag.bin 将恰好包含 16 个字节，代表原始的 AES-128-CMAC 标签。该值与 hex2000 应用其最终变换之前在内部计算出的标签相对应。

English Document: SPRACT3
Copyright © 2026 Texas Instruments Incorporated

第 5 步 — 对标签应用 swapwords 以获得存储的格式。

对第 4 步生成的 16 字节 AES 标签应用相同的 swapwords 变换。得到的 16 个字节即构成金标 CMAC 标签，采用 hex2000 所使用且引导 ROM 所需的存储格式。该值应与 hex2000 在已签名闪存映像的字节偏移 0x4 处写入的字节相匹配，并且是引导 ROM 在身份验证期间从 0x00080002（C28x 字地址）处的金标 CMAC 标签位置读取的值。

请注意，由于 swapwords 是自逆的，引导 ROM 只需读取存储的标签并再次应用 swapwords，即可恢复原始 AES 标签，然后再将其与对屏蔽及转换后的闪存内容重新计算出的标签进行比较。

10.5 工作示例

以下示例使用 NIST AES-128 测试密钥 2b7e151628aed2a6abf7158809cf4f3c，将其应用于一个极小的闪存映像。第一个应用程序字（偏移 0x0）是“LB 0x081BC8”分支指令，汇编为两个 C28x 字“0x4800”和“0x1BC8”。偏移 0x4 处的金标 CMAC 标签占位符初始化为 0。偏移 0x14 及之后的字段处于其擦除状态 (0xFF)。16KB 区域的其余部分填充 0xFF。

签名前的二进制文件（显示前 24 字节；其余为 0xFF）：

```
Offset 0x0000:  00 48 C8 1B   (codestart: LB 0x081BC8)
Offset 0x0004:  00 00 00 00   (cmac_sb_1, initialized to zero)
Offset 0x0008:  00 00 00 00   (cmac_sb_1, continued)
Offset 0x000C:  00 00 00 00   (cmac_sb_1, continued)
Offset 0x0010:  00 00 00 00   (cmac_sb_1, continued)
Offset 0x0014:  FF FF FF FF   (erased contents)
```

第 2 步之后 — 签名区域屏蔽为 0xFF：

```
Offset 0x0000:  00 48 C8 1B   (unchanged)
Offset 0x0004:  FF FF FF FF   (masked)
Offset 0x0008:  FF FF FF FF   (masked)
Offset 0x000C:  FF FF FF FF   (masked)
Offset 0x0010:  FF FF FF FF   (masked)
Offset 0x0014:  FF FF FF FF   (unchanged)
```

第 3 步之后 — 已应用 swapwords（显示前 8 字节）：

```
Offset 0x0000:  C8 1B 00 48   ([b2,b3,b0,b1] of original offset 0x0000)
Offset 0x0004:  FF FF FF FF   (unchanged; all bytes identical)
```

第 4 步 — OpenSSL AES-128-CMAC 输出（aes_tag.bin，16 字节）：

```
f2 70 99 41  e1 cf 3c cf  f1 36 7a a3  c4 3e f6 f7
```

第 5 步之后 — 已对标签应用 swapwords（存储的格式）：

```
99 41 f2 70  3c cf e1 cf  7a a3 f1 36  f6 f7 c4 3e
```

最终的已签名二进制文件（前 32 字节）：

```
Offset 0x0000:  00 48 C8 1B
Offset 0x0004:  99 41 F2 70   (stored tag bytes 0-3)
Offset 0x0008:  3C CF E1 CF   (stored tag bytes 4-7)
Offset 0x000C:  7A A3 F1 36   (stored tag bytes 8-11)
Offset 0x0010:  F6 F7 C4 3E   (stored tag bytes 12-15)
Offset 0x0014:  FF FF FF FF
```

上述存储的标签与使用相同密钥对相同二进制文件应用 *hex2000 --cmac* 的输出相匹配。经验证，该结果与 hex2000 输出逐字节完全相同。

English Document: SPRACT3
Copyright © 2026 Texas Instruments Incorporated

10.6 差异总结

hex2000 --cmac 签名流程与直接调用 OpenSSL AES-128-CMAC 命令之间仅存在唯一的行为差异，即围绕 AES 操作的两次 swapwords 变换。在两种情况下，底层密码算法、密钥编排和初始化向量完全相同。表 10-2 总结了两种流程之间的完整对应关系，以供参考。

表 10-2. C2000 与 OpenSSL CMAC 流程对比

方面	C2000	OpenSSL
消息预处理	在 AES 之前应用 swapwords(masked_binary)	原始字节流；不应用变换
签名区域屏蔽	字节偏移 0x4 – 0x13 替换为 0xFF	调用 OpenSSL 前必须手动应用
密钥字节顺序	MSB 在前 (CMACKEY0 || ... || CMACKEY3)	MSB 在前；顺序相同，无需转换
AES-128-CMAC 算法	NIST SP 800-38B (OMAC1)	NIST SP 800-38B (OMAC1)；完全相同
初始化向量	全 0（在 CMAC 标准中隐式定义）	全 0（在 CMAC 标准中隐式定义）
标签后处理	嵌入前应用 swapwords(aes_tag)	原始 AES 标签输出；必须手动应用 swapwords
输出	嵌入了所存储标签的已签名二进制文件	16 字节原始 AES 标签；需进一步处理

备注

swapwords 变换以 32 位组为单位操作，因此要求身份验证区域长度为 4 字节的倍数。16KB（0x4000 字节）的 C28x 主安全启动区域可满足此要求。当按照 节 7 所述对自定义闪存范围执行扩展身份验证时，起始和结束地址同样必须对齐到 128 位，才能使 CMAC 算法在无填充的情况下正确运行。

11 参考文献

1. 德州仪器 (TI)：*《TMS320F2838x 实时微控制器技术参考手册》*
2. 适用于 C2000 实时 MCU 的 C2000Ware
3. 德州仪器 (TI)：*《TMS320C28x 汇编语言工具 v20.2.0.LTS 用户指南》*
4. 德州仪器 (TI)：*《Arm 汇编语言工具 v20.2.0.LTS 用户指南》*
5. TMS320F28338D 产品页面

English Document: SPRACT3
Copyright © 2026 Texas Instruments Incorporated

12 修订历史记录

Changes from Revision * (September 2020) to Revision A (June 2026) **Page**

- 更新了新版 C2000 器件上的安全启动支持信息。 1
- 更新了整个文档中的表格、图和交叉参考的编号格式 2
- 新增一节，详述 C2000 实现与 OpenSSL 的对齐方式。 9

English Document: SPRACT3
Copyright © 2026 Texas Instruments Incorporated

重要通知和免责声明

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、与某特定用途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安全、安保法规或其他要求。

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 产品的相关应用。严禁以其他方式对这些资源进行复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。对于因您对这些资源的使用而对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，您将全额赔偿，TI 对此概不负责。

TI 提供的产品受 TI 销售条款)、TI 通用质量指南 或 ti.com 上其他适用条款或 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。 除非德州仪器 (TI) 明确将某产品指定为定制产品或客户特定产品，否则其产品均为按确定价格收入目录的标准通用器件。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

版权所有 © 2026，德州仪器 (TI) 公司

最后更新日期：2025 年 10 月