GigaDevice Semiconductor Inc.

合理使用 UID 助力固件保护

应用笔记

AN073

目录

目录 2
图索引 3
表索引 4
1. 前言 5
2. 应用场景与能力分析 6
2.1. 应用场景 6
2.2. 代码需要的能力 6
3. 应对的方法 7
3.1. 概述 7
3.2. 使用特定外设 7
3.3. 改进设计方法 7
3.4. 增加冗余代码 10
3.4.1. 参考实现一 10
3.4.2. 参考实现二 11
3.5. 软件加密 13
3.6. 外接加密芯片 15
4. 版本历史 17

图索引

图 3-1. 改进设计方法原理图 7
图 3-2. Image 文件生成流程图 8
图 3-3. 程序运行流程图 9
图 3-4. 冗余代码的实现流程图 11
图 3-5. 漏洞和补漏洞原理图 12
图 3-6. 漏洞和补漏洞程序流程图 13
图 3-7. 软件加密原理图 14
图 3-8. 加密 Image 生成流程图 14
图 3-9. 运行加密 Image 流程图 15
图 3-10. MCU 外接 HASH 芯片示意图 16
图 3-11. 认证的流程图 16

表索引

表 3-1. 顺序执行代码单元格..... 9
表 3-2. 乱序处理后执行代码单元格..... 9
表 3-3. 特殊函数代码单元格.....10
表 3-4. 普通代码单元格.....10
表 3-5. 特殊函数处理代码单元格.....11
表 3-6. 漏洞代码代码单元格.....12
表 3-7. 补漏洞代码.....12
表 4-1. 版本历史.....17

1. 前言

本文是专为保护软件代码设计，旨在通过特定方法实现对软件代码的保护，防止软件代码被移植到其它平台或芯片。

本文总共分为两个部分来讲述，第一部分介绍本文的应用场景与代码所需的能力；第二部分介绍保护代码的方法。

本应用笔记理论上适用于全系列 GD32 MCU 产品。

2. 应用场景与代码能力分析

2.1. 应用场景

GD32 MCU 在硬件上提供了安全保护功能来阻止非法读取闪存，此功能可以很好的保护固件被盗取。但是在产品实际研发或量产中，有很多应用场景不需要开启读写保护，例如：

- 合作开发产品，多方开发者将各自开发的代码固化在flash中，以供后续开发或量产使用；
- 算法提供商将算法固化在微控制器内，以供客户使用；
- 用户需要根据使用场景设置参数，将参数保存至 Flash 中；
- 产品交付时软件功能还不稳定或不完善，需要根据用户使用反馈，进行更新迭代软件功能。

以上情况，MCU 通常不能开启读写保护，会导致微控制器内的固件可以被轻易的读取、移植或逆向。因此，为防止固件被盗取或移植到其它平台，需要采取其它方式保护固件。

2.2. 代码需要的能力

在不开启硬件读写保护的情况下，从代码设计的角度出发，其需要具备以下两个能力：

- 从某颗芯片中读出的代码，无法直接运行到其它芯片中。
- 代码设计具备一定的复杂度，第三方无法在短时间内分析掌握代码。

3. 应对的方法

3.1. 概述

对于同一系列的每颗 MCU 都拥有独有的 96 位 UID，将代码与 UID 绑定，能够做到不同芯片无法直接共用代码，此外，可以从代码设计、冗余设计、增加验证硬件和使用特定外设等方式，增加代码的复杂度。

本章后续阐述了使用特定外设、改进设计方法、增加冗余代码、软件加密和使用加密芯片五种方案增加代码复杂度，它们并不互斥，用户在使用时可以相互结合。

3.2. 使用特定外设

本节所述的方法旨在增加代码分析的难度，其基本思路是通过对 MCU 外设的使用，来增加代码分析的难度，如在读取 UID 时，不是直接通过寄存器读出后使用，而是通过 DMA 多次搬运之后才使用获取的 UID 值。类似的还可以使用 MPU 和特权级别来增加代码的复杂度。

此外，可以开启硬件的安全保护和读保护，来保护固件防止被读取、调试。

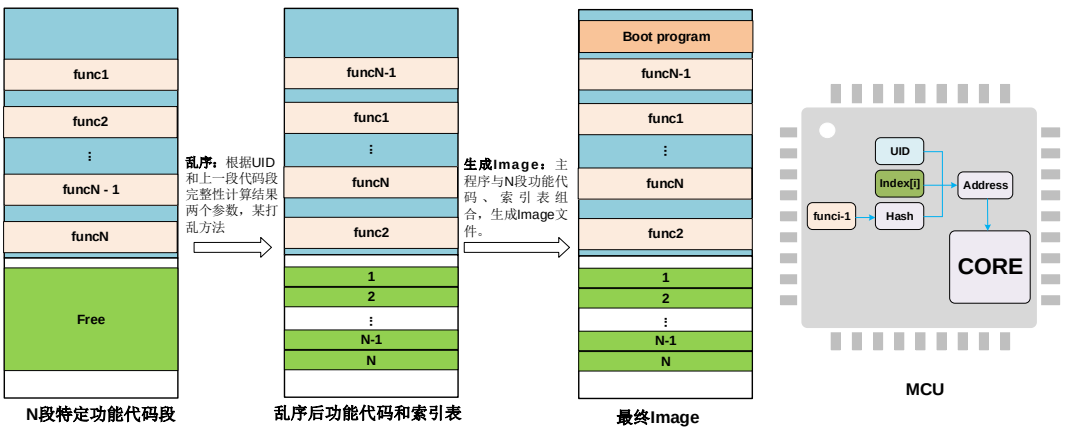
3.3. 改进设计方法

本节从改进代码设计方法的角度出发，将代码与 MCU 绑定。

这种方法如*图 3-1. 改进设计方法原理图*所示，在 IDE 中将 N 段功能代码分散加载至指定地址，每段功能代码的大小一致，上位机读取生成的 BIN 或 HEX 文件。上位机根据 MCU 的 UID 和上一段代码完整性计算结果两个参数，使用特定打乱算法，将功能代码的存放顺序打乱。功能代码分散加载可参考 *AN075《一种基于 MDK 的算法库调用方法及其实现》*。

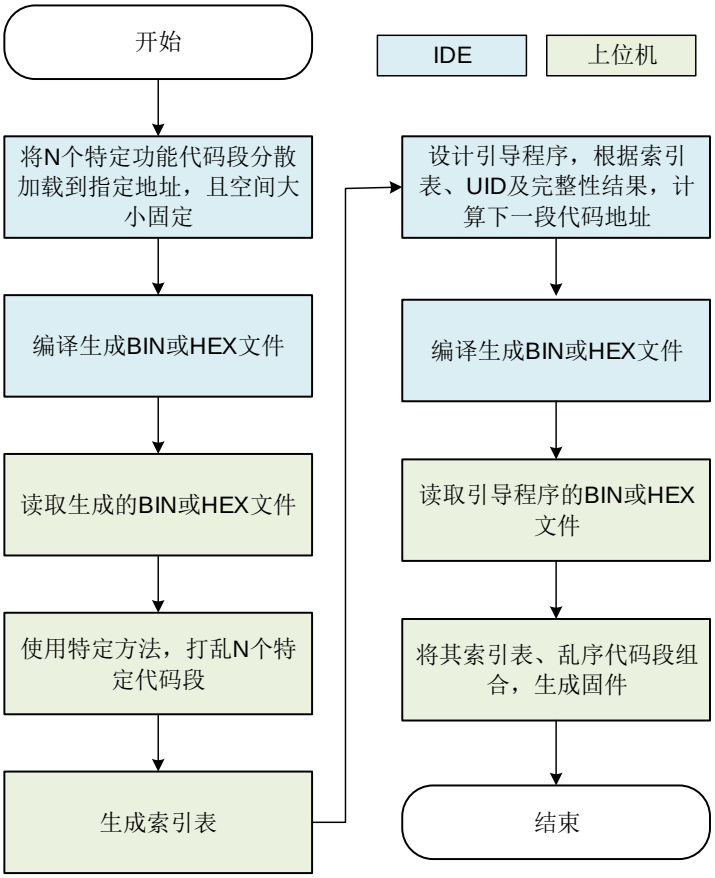
代码完整性的计算方法包括并不限于 hash、CRC、checksum 等，用户自定义打乱方法。当处理第一段代码时，将全部代码的完整性结果以及 UID 作为参数，确定其乱序后的地址。

图3-1. 改进设计方法原理图



为了确保运算地址具有可用性，需要在计算结果的基础上进行位置编码计算，计算的结果存放在索引表内。最终的 Image 包括索引表、引导程序和乱序后功能代码。Image 生成过程如*图 3-2. Image 文件生成流程图*所示。

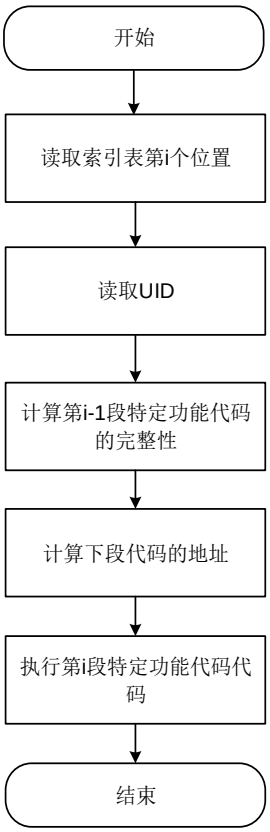
图3-2. Image文件生成流程图



在程序运行时，当需要执行某段功能代码时，根据 UID、前一段代码完整性结果以及索引表，使用与上位机相同的计算方法，计算出需要执行代码的地址。例如执行第 i 段关键代码的流程如

图 3-3. 程序运行流程图所示，若 UID 或者前一段代码更改，则会计算出错误地址，导致程序出现未知错误。

图 3-3. 程序运行流程图



当不对功能代码进行任何处理，功能代码顺序执行，程序逻辑容易理解，容易被抄袭。如下*表 3-1. 顺序执行代码单元格*所示。

表 3-1. 顺序执行代码单元格

```
/* execute functional code */
func1();
func2();
func3();
func4();
```

```
func5();
```

而当功能代码进行乱序处理后，在执行功能代码时，需要读取 UID 和检查前一段代码完整性，根据其计算出下一段代码的地址。这种方法确保代码不能够被修改，且不同的 MCU 无法共用代码。参考代码如下*表 3-2. 乱序处理后执行代码单元格*所示。

表 3-2. 乱序处理后执行代码单元格

```
/* define function pointer */
void (*func)(void);
/* define index address */
uint32 * index = (uint32 *)(0x0801F000);
/* calculate full code's completeness  w hen index = 0 */
uint32 * previous_ res = calculate_code_ completeness  (address_full_code,full_size);
/* calculate next functional code's address */
uint32 * address = calculate_address(getuid(),index[0],  previous_ res);
func = address;
/* execute functional code */
func();
for(i = 1 ; i < N; i ++ )
{
      /* calculate previous code's completeness  */
     previous_ res = calculate_code_ completeness  (address,0x1000);
     /* calculate next functional code's address */
     address = calculate_address(getuid(),index[i],  previous_ res);
     func = address;
     /* execute functional code */
     func();
}
```

3.4. 增加冗余代码

本节描述的方法通过增加和 UID 相关的冗余代码，来增加的代码的复杂度，冗余代码执行的结果会影响主代码的运行结果，如结果不正常会导致主代码运行异常。本节将列举 2 种参考的实现。

3.4.1. 参考实现一

在软件程序设计时增加特殊函数代码，例如下*表 3-3. 特殊函数代码单元格*所示。

表 3-3. 特殊函数代码单元格

函数	返回值	实现
fun0	0	读 flash 地址 0 与 UID 做运算
fun1	1	读 flash 地址 1 与 UID 做运算
fun2	2	读 flash 地址 2 与 UID 做运算

fun3	3	读 flash 地址 3 与 UID 做运算
….	…	…
funn	n	读 flash 地址 5 与 UID 做运算

函数返回值为特定的数值，这些函数读取芯片特定的 flash 位置，这些位置会在程序下载的时候根据芯片的 UID 写入，不同的芯片会写入不同的值，而在程序设计时需要调用数值的地方，通过调用这些函数间接获得，*图 3-4. 冗余代码的实现流程图*为设计的流程图。

在无特殊函数代码处理的情况下，正常代码逻辑简单，易被理解，参考代码如下*表 3-4. 普通代码单元格*所示。

表 3-4. 普通代码单元格

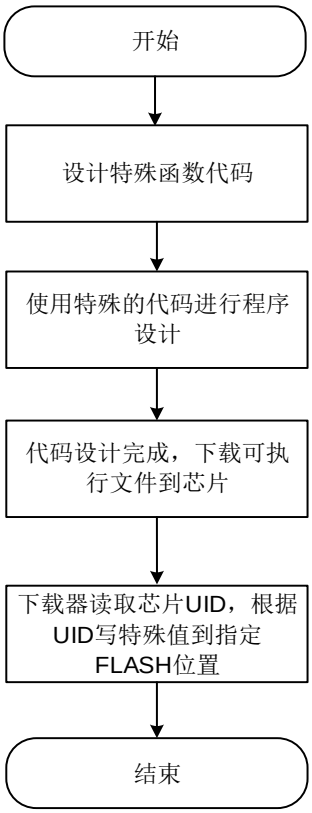
```
/* execute test code */
uint8_t *buf = NULL;
for ( i = 0; i < 10; i++ ){
    printf("%d\r\n", i );
}
buf = (uint8_t *)malloc(200);
```

在代码中添加特殊函数代码处理后，程序与芯片 UID 绑定，参考代码如下*表 3-5. 特殊函数处理代码单元格*所示，若芯片的 UID 更改，则会导致程序运行出错。

表 3-5. 特殊函数处理代码单元格

```
/* execute test code */
uint8_t *buf = NULL;
for ( i = 0; i < 10 * func1() ; i++ ){
    printf("%d\r\n", i );
}
buf = (uint8_t *)malloc(100*func2( ));
```

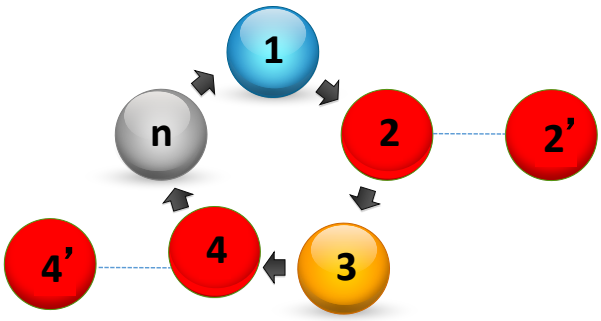
图 3-4. 冗余代码的实现流程图



3.4.2. 参考实现二

本节所述的参考实现，旨在通过在原有代码中增加漏洞和补漏洞代码对，来增加代码的复杂度，执行漏洞代码后，必须要执行补漏洞的代码，否则程序出现异常。根据芯片 UID 和随机数运行程序内的漏洞和补漏洞代码，具体原理如*图 3-5. 漏洞和补漏洞原理图*所示。

图 3-5. 漏洞和补漏洞原理图



漏洞代码参考*表 3-6. 漏洞代码代码单元格*。

表 3-6. 漏洞代码代码单元格

```
/* bug code */
int flag1 = 0;
void fun1(void)
{
    if (1 == flag1){
        flag1 = 0;
        *((uint32_t *)0) = 3;
        return;
    }
    if (0 == flag1){
        flag1 = 1;
    }
    else if (2 == flag1){
        flag1 = 0;
    }
}
```

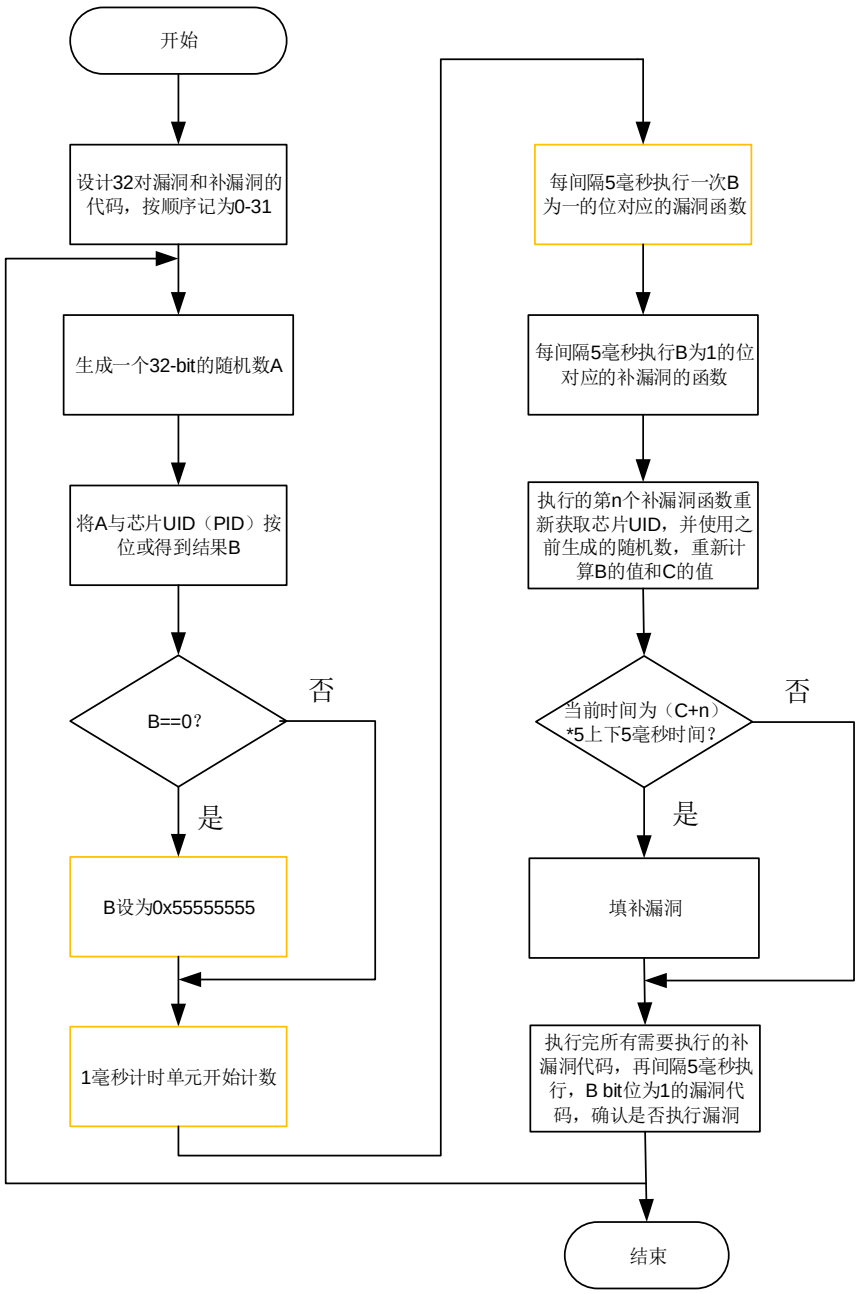
补漏洞代码参考*表 3-7. 补漏洞代码* 。

表 3-7. 补漏洞代码

```
/* fixs bug code */
void fun11(void)
{
   /* time check right */
    flag1 = 2;
}
```

在程序中添加漏洞和补漏洞的具体流程如*图 3-6. 漏洞和补漏洞程序流程图*所示。

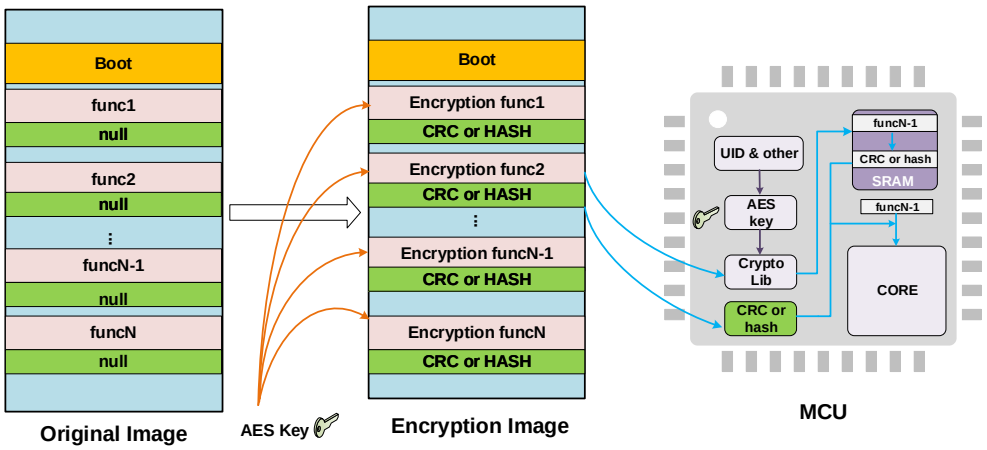
图 3-6. 漏洞和补漏洞程序流程图



3.5. 软件加密

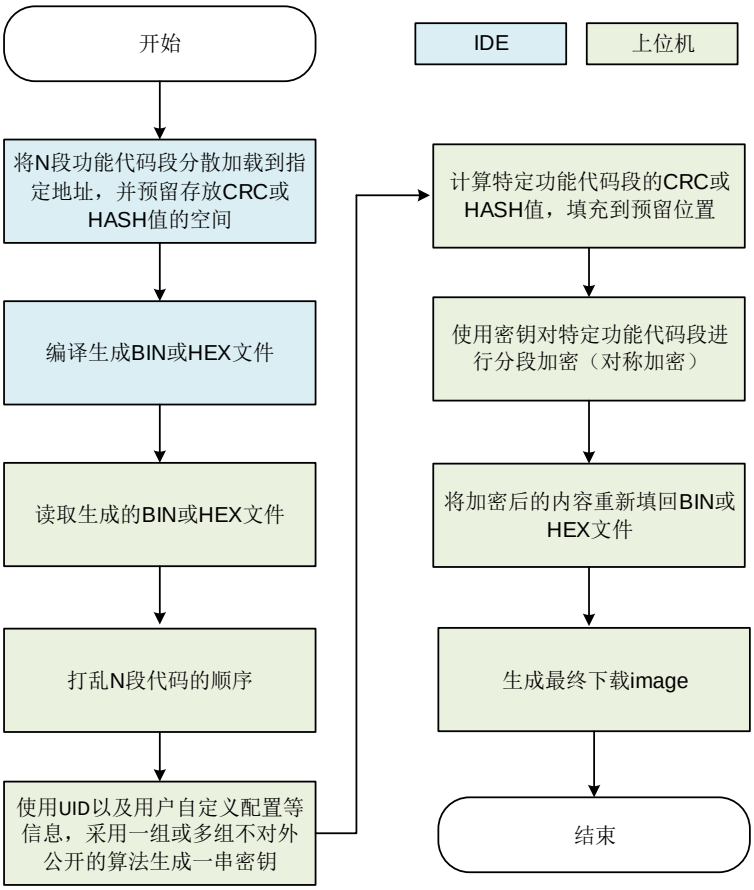
使用加解密算法将关键功能代码进行加密和解密，使用 UID 和用户自定义配置信息生成密钥。加解密算法可以使用开源算法库，如面向小型嵌入式设备的 mbedTLS 加密算法库。软件加密具体原理如*图 3-7. 软件加密原理图*所示。

图 3-7. 软件加密原理图



在上位机计算每段关键功能代码的 CRC 或 HASH 值，将其存放在预留空间，然后使用 AES 秘钥对每段代码进行对称加密，回填至 BIN 或 HEX 文件，将加密后的 Image 下载到芯片内。加密 Image 生成如*图 3-8. 加密 Image 生成流程图*所示。

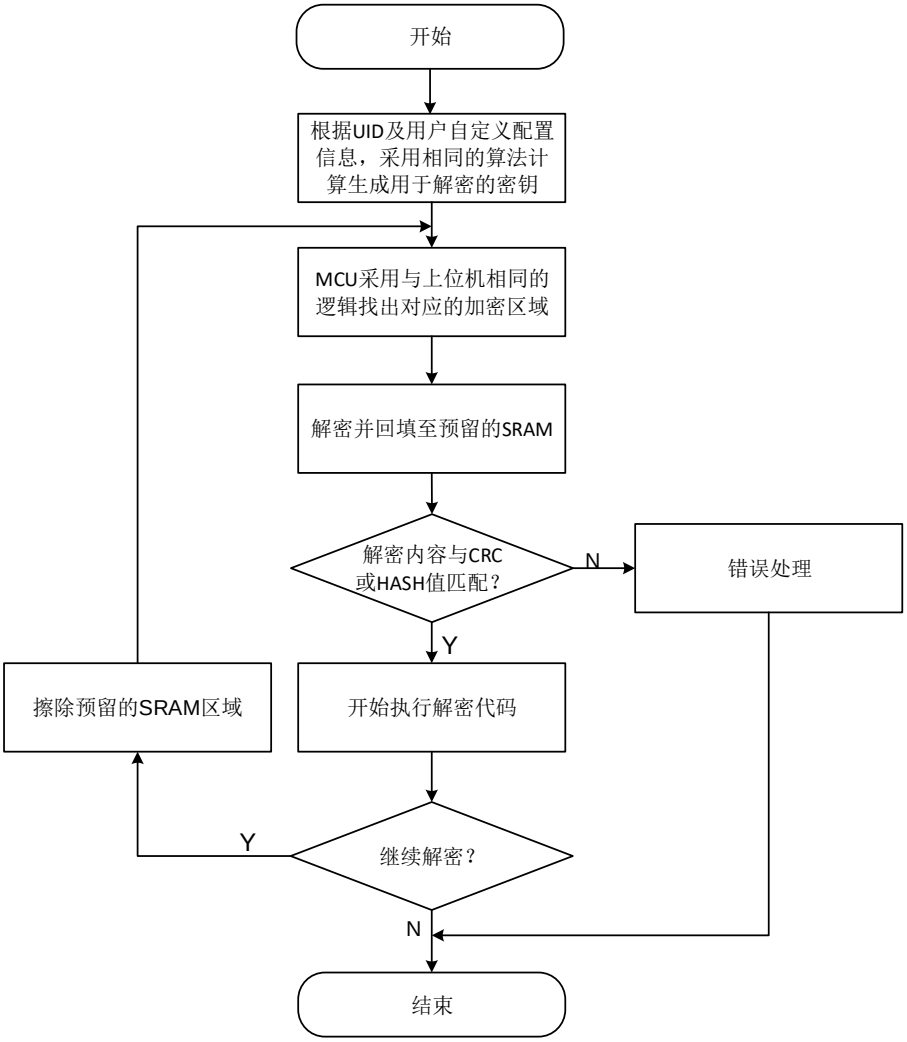
图 3-8. 加密 Image 生成流程图



在运行时，MCU 端 boot 段的代码，根据 UID 及用户自定义配置信息采用相同的算法计算生成用于解密的密钥，此处的操作虽然是未经加密的，但是由于采用了算法计算，会增加反汇编解析的难度。需要注意的是，MCU 计算出的密钥值在使用时可以通过 DMA 或中断的方式处理，

例如将计算好的密钥通过 DMA 传输到已经预先分配好的变量中去再使用，然后在 DMA 的传输完成中断中进行解密。运行过程如*图 3-9. 运行加密 Image 流程图*所示。

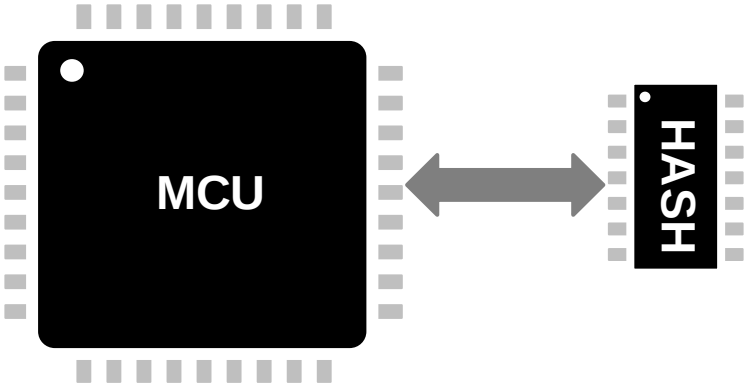
图 3-9. 运行加密 Image 流程图



3.6. 外接加密芯片

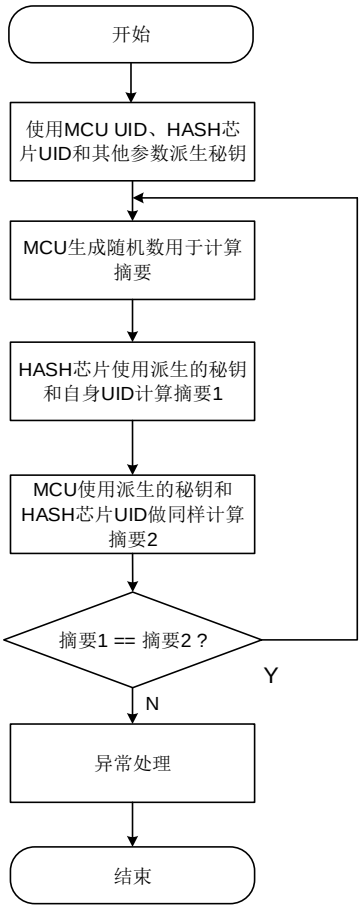
本节所述方法是使用外接安全芯片的方式，旨在增加代码的安全性。MCU 外接 HASH 芯片，利用了哈希芯片本身存储安全性的特点，来识别待认证区域的合法性。连接示意图如*图 3-10. MCU 外接 HASH 芯片示意图*所示。

图 3-10. MCU 外接 HASH 芯片示意图



具体认证过程如*图 3-11. 认证的流程图*所示。首先上位机端会使用 MCU UID 和 HASH 芯片的 UID 以及自定义的参数，派生用于计算摘要的秘钥，并将其写入到 HASH 芯片中；MCU 程序运行时会产生一个随机数，并将其发送给 HASH 芯片；HASH 芯片利用随机数、派生的秘钥以及自身的 UID 计算摘要 1；MCU 利用随机数、派生的秘钥以及 HASH 芯片的 UID 计算摘要 2；MCU 比对两个摘要，如果不同则会进入异常处理代码，该代码可以是让代码进入死循环或其它操作。由于 HASH 拥有写入后就不能更改的特性，所以可以做到一颗 HASH 芯片绑定一个设备的效果。

图 3-11. 认证的流程图



4. 版本历史

表 4-1. 版本历史

版本号.	说明	日期
1.0	首次发布	2022 年 10 月 28 日

Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the “Product”), is owned by the Company under the intellectual property laws and treaties of the People’s Republic of China and other jurisdictions worldwide. The Company reserves all rights under such laws and treaties and does not grant any license under its patents, copyrights, trademarks, or other intellectual property rights. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

The Company makes no warranty of any kind, express or implied, with regard to this document or any Product, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Company does not assume any liability arising out of the application or use of any Product described in this document. Any information provided in this document is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Except for customized products which has been expressly identified in the applicable agreement, the Products are designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only. The Products are not designed, intended, or authorized for use as components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, atomic energy control instruments, combustion control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or Product could cause personal injury, death, property or environmental damage ("Unintended Uses"). Customers shall take any and all actions to ensure using and selling the Products in accordance with the applicable laws and regulations. The Company is not liable, in whole or in part, and customers shall and hereby do release the Company as well as it’s suppliers and/or distributors from any claim, damage, or other liability arising from or related to all Unintended Uses of the Products. Customers shall indemnify and hold the Company as well as it’s suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Products.

Information in this document is provided solely in connection with the Products. The Company reserves the right to make changes, corrections, modifications or improvements to this document and Products and services described herein at any time, without notice.