STM32Cube 学习之十六： ID 加密

在单片机产品开发完成后，可能有人通过工具读取 MCU 上 FLASH 中的代码，窃取我们的程序。本文的通过一个完整的例程，演示如何利用 STM32 的 ID 对程序进行加密，使他人即使读到 FLASH 中的代码，也无法直接使用，从而保护自己的劳动成果。

先引用坛友“人生如梦 123” 2015 年 9 月 28 日在 ST 社区发布的帖子“[原创] STM32 防破解”

帖子链接：http://www.stmcu.org/module/forum/thread-603421-1-1.html

本人之前是做软加密破解的，后来良心发现，自觉做我们这行窃取他人劳动成果，和做贼没有区别，所以果断转行了，首选声明我绝不是高手，只是想将我做破解的时候的一些经验和大家分享，以便各位兄弟在做产品的时候重视加密，不要让别人轻而易举就窃取了您的劳动成果，

STM32 系列从问世以来就以优异的性能和便宜的价格深得人心，但是不幸的是树大招风,问世不就久就被解密公司破了，从 12 年的 12 万的解密价格，到 13 年的 6 万，14 年一万，再到现在的五千，相信不用两年就会像 51 一样沦为几百块的白菜价，所以软加密对未来的 STM32 的工程师来说异常重要，以下正式对各种方式的软加密和破解方法做一个总结，以便各位朋友在日后设计软加密的时候不要给破解的人留下漏洞

1.最简单的软加密不用反汇编，直接在机器码中就可以先到 FF1FE8F7,因为 STM32 机器码是小端格式，这个地址实际就是芯片 ID 地址，破解的人只需要在程序中找到一块空白的位置，然后将解密的那个芯片的 ID 复制到这里，再将程序中出现的那个 1FFFF7E8 改为存放母片的那个 ID 就破解了，这种方法和你程序采用的什么算法加密毫无关系，防破解处理方法是在程序加密的时候不要直接读芯片 ID,应采用几个变量运算合成 ID 地址再间接的去读，注意不能用立即数合成，因为那样编译器还是会给你优化成一个立即数的。

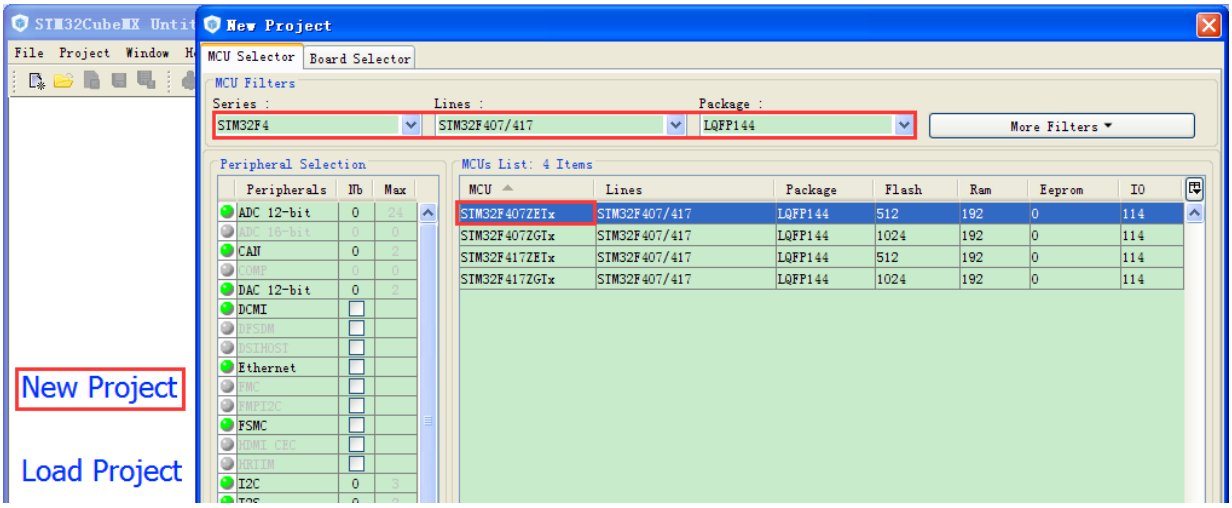
2.就算在程序中找不到明显的读 ID 指令也是可以破解的，方法就是仿真跟踪，仿真跟踪前需要反汇编，THUMB2 的文档中每条指令生成的机器码有详细的说明，随便都可以找个做上位机的写个自动反汇编工具，之后再人工修改下就可以了，其实还有一种更简单的方法，就是将机器码定义成 DCI XXXXH 这样的格式，导入 KEIL 编译能通过，然后仿真，KEIL 会自动的帮你反汇编，接下来就是单步执行，延时类函数跳过，这时候要密切注视 R0 到 R15,不管你用什么方法得到的 ID 地址，最终一定会出现再这几个寄存器中，防破解的方法一个是检验 ID 号的时候不要在开机就检验，要在特定的硬件条件下才检验 ID，然后如果不合法程序就自毁，这样就只能通过 JTAG 硬件仿真了。所以产品上市的时候切记将其它 IO 口转到 JTAG 口，这样就占用了 JTAG,仿真就不行了

3.是不是这样就安全了？不是的，你可以禁用 JTAG,人家同样可以修改指令开启 JTAG,最好的方式是在程序关键的代码块做 CRC 检验，这样只要关键指令被修改过，就可以发现，剩下的自己看着办……

4.其实没有破不了的软加密，只是一个时间和成本的问题，但是也不能让人家那么轻易的就破解了，除了上面的防破解的方法之外还可以在程序中特定条件下做多次检验，检验的时候不要用简单的判断真假跳转，应该用检验的结果做程序下一步执行的参数，这样别人破出来的产品原以为没问题了，但是用起来不稳定，或者性能差，或者老死机等。除了这些还可以外挂加密芯片，以增加破解的成本，更多防破解的方法欢迎加我 QQ254915501 讨论

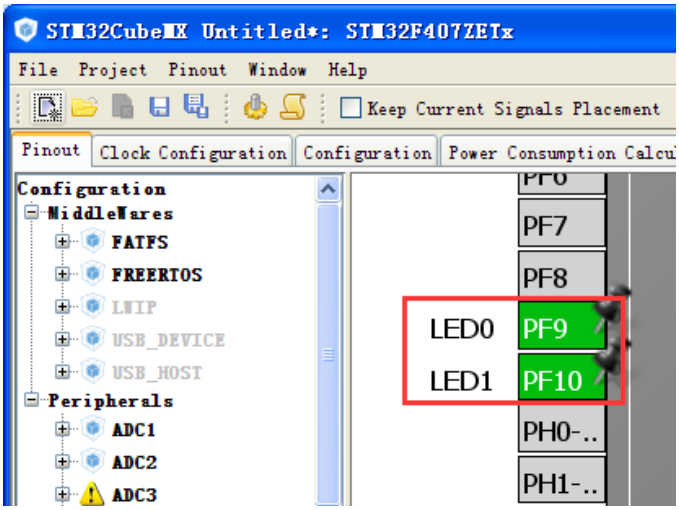
假设已经安装好 STM32CubeMX 和 STM32CubeF4 支持包。

Step1.打开 STM32CubeMX，点击“New Project”，选择芯片型号，STM32F407ZETx。

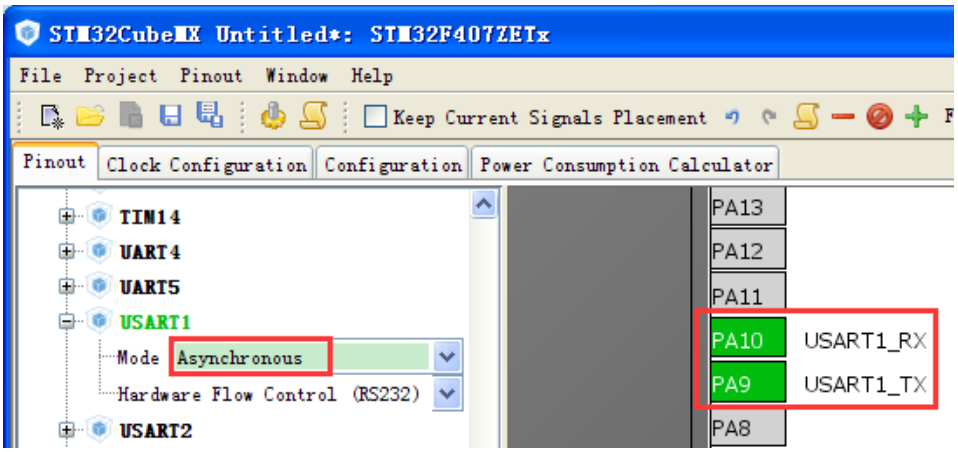


Step2. 在 Pinout 界面下配置引脚功能。

根据电路使用 PF9,PF10 作 LED 控制，将 PF9,PF10 的功能配置为 GPIO_Output，并把用户标签改为 LED0 和 LED1。

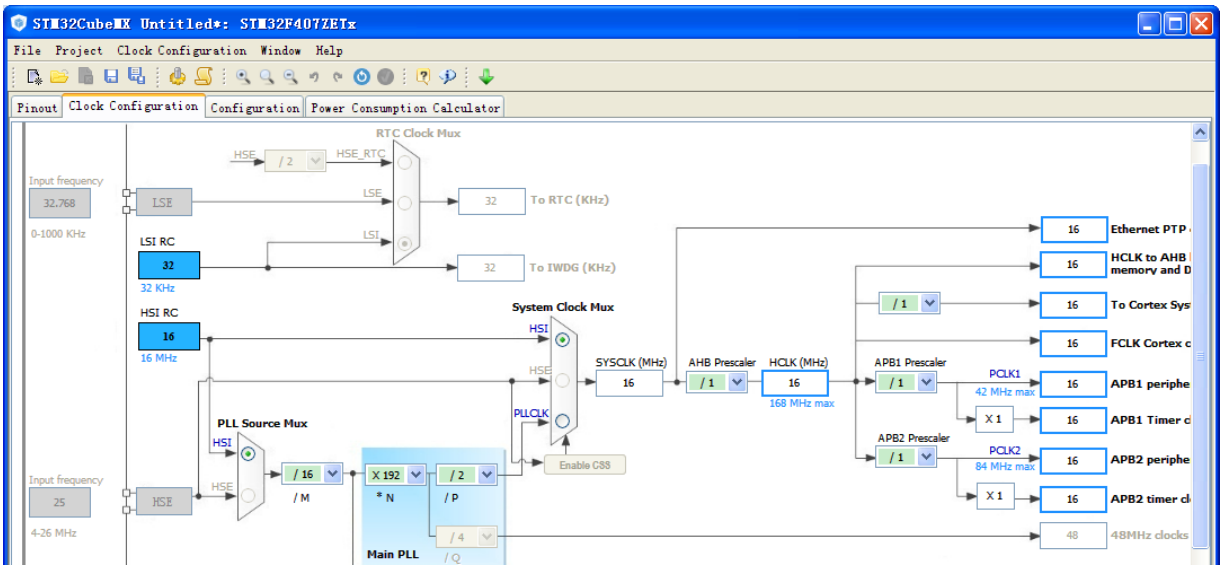


配置串口，作为信息输出接口。



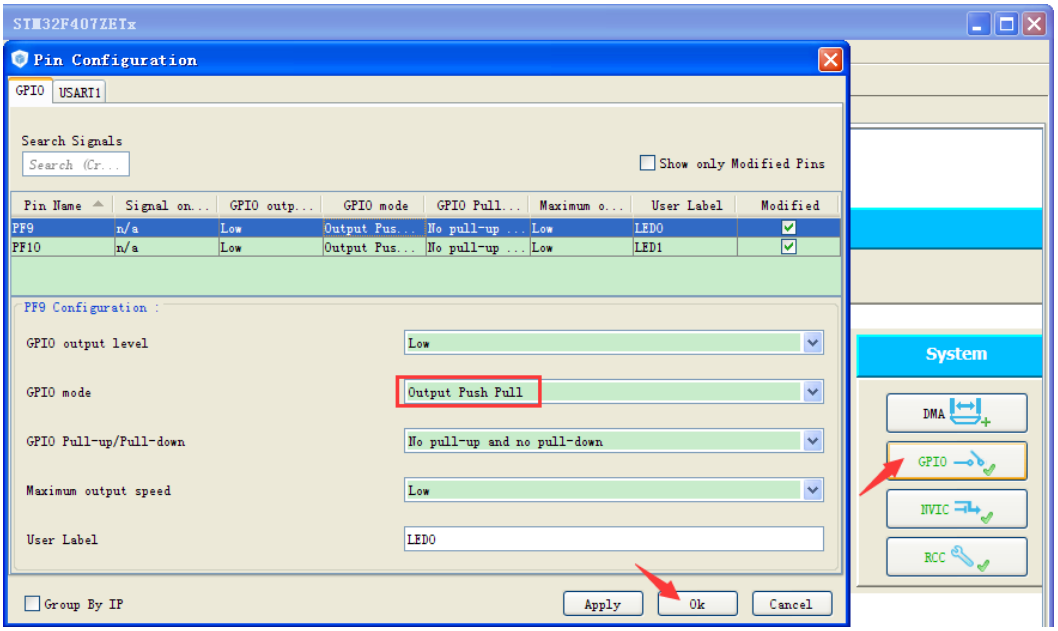
Step3.在 Clock Configuration 界面配置时钟源。

配置时钟树，在此使用默认值， 内部 16MHz。

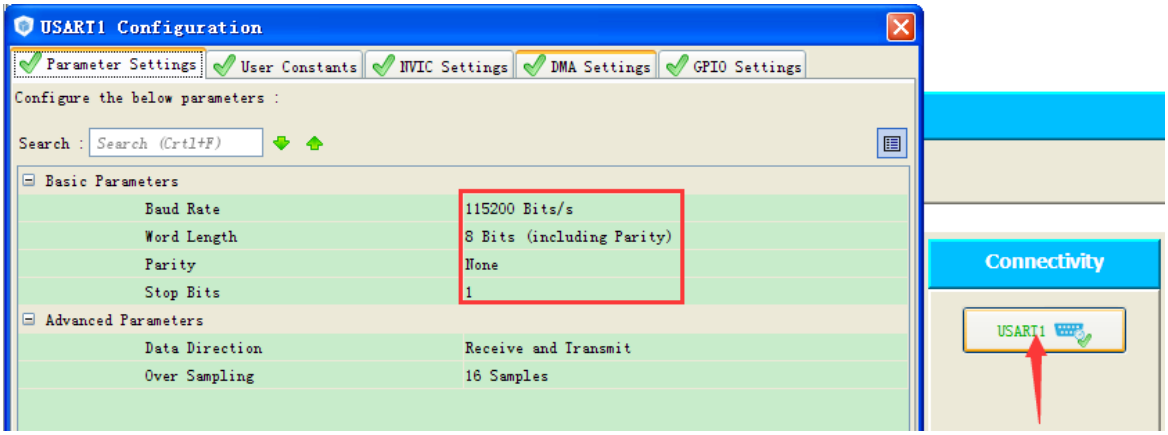


Step4.配置外设参数。

GPIO：使用默认的推挽即可。

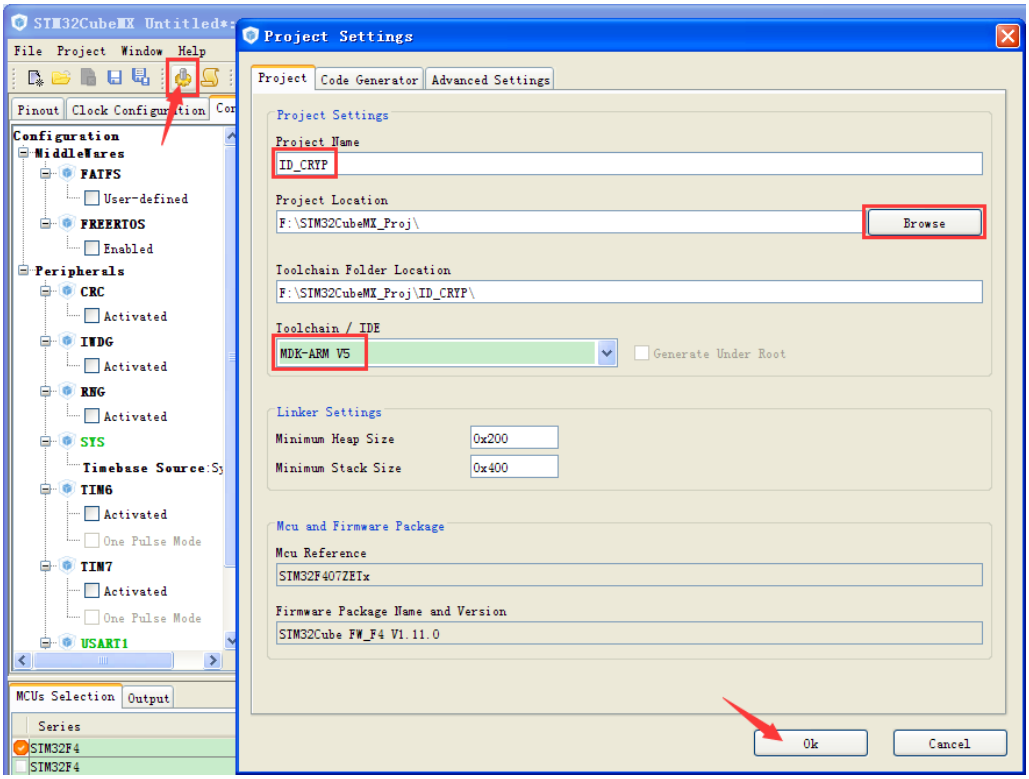


USART：串口参数默认即可。

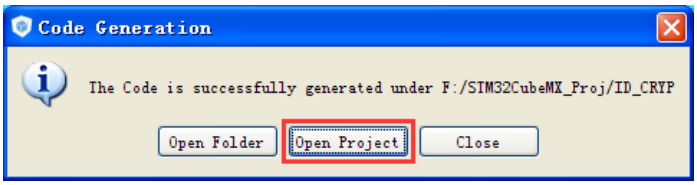


Step5.生成源代码。

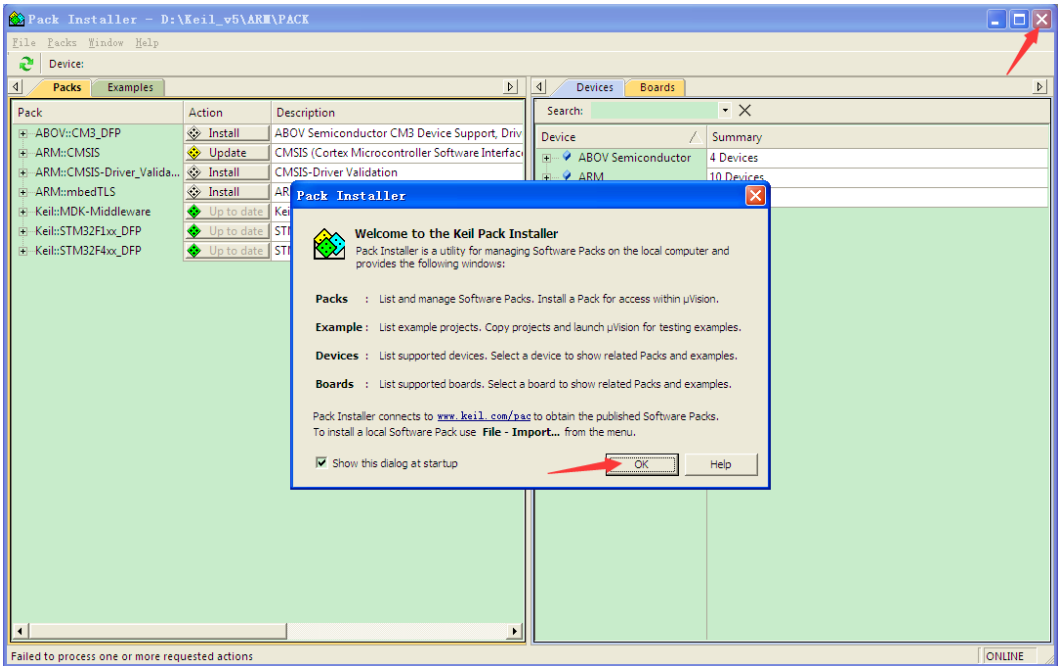
点击生成源代码按钮。在设置界面中输入工程名，保存路径，工程 IDE 类型，点 OK 即可。



生成代码完成后可直接打开工程。

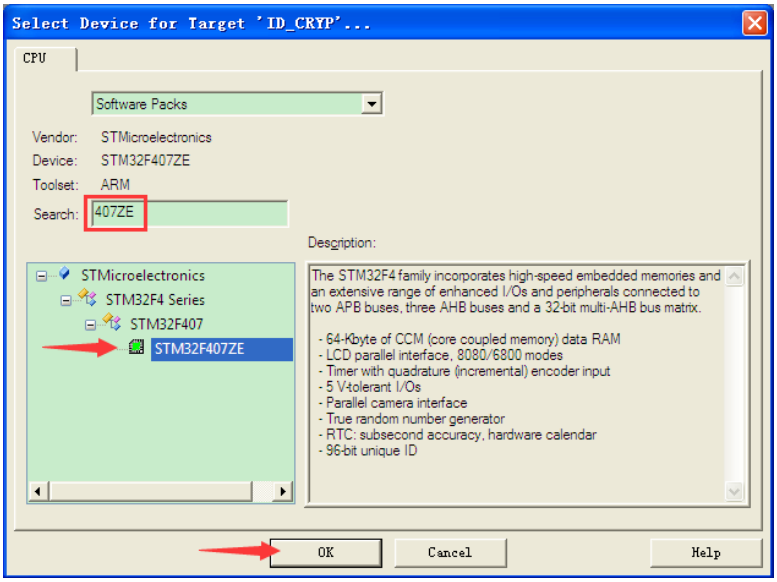


弹出如下对话框时，如果已经安装了 F4 的支持包，则点击 OK 关闭。如果没有安装，则点击界面中的 www.keil.com/...链接，找到芯片的支持包，然后安装。



关闭后面的界面。

点击“是”，然后选择芯片型号。可以在搜索框中输入关键字，加快选择速度。



Step6.添加功能代码。

先在 main.c 文件用户代码区输入包含标准输入输出头文件。

```
33  /* Includes ------------------------
34  #include "stm32f4xx_hal.h"
35
36  /* USER CODE BEGIN Includes */
37  #include <stdio.h>
38  /* USER CODE END Includes */
39
```

在用户代码区 4 实现标准输出 printf()的底层驱动函数 fputc()，功能是在 UART1 输出一个字符。

```
176 /* USER CODE BEGIN 4 */
177 int fputc(int ch, FILE *f)
178 {
179     HAL_UART_Transmit(&huart1, (uint8_t *)&ch, 1, 10);
180     return ch;
181 }
```

在 main 函数中定义一个变量。

```
66    /* USER CODE BEGIN 1 */
67    HAL_StatusTypeDef  res;
68    /* USER CODE END 1 */
```

在/* USER CODE BEGIN 2 */和/* USER CODE END 2 */用户代码，添加如下代码：

```
82  /* USER CODE BEGIN 2 */
83  res = check_ID();
84  if (HAL_OK == res) { // ID校验正确,则执行正常程序
85      while (1) { // 两个LED闪烁,每秒闪1次
86          HAL_GPIO_TogglePin(LED0_GPIO_Port, LED0_Pin);
87          HAL_GPIO_TogglePin(LED1_GPIO_Port, LED1_Pin);
88          printf("Hello! It's normal routine.\r\n");
89          HAL_Delay(500);
90      }
91  } else if (HAL_BUSY == res) {  // 第一次运行,使用通用密码
92      while (1) { // 只有一个LED闪烁,每秒闪1次
93          HAL_GPIO_TogglePin(LED0_GPIO_Port, LED0_Pin);
94          printf("It's running first time.\r\n");
95          HAL_Delay(500);
96      }
97  } else {  // ID校验错误,则执行异常程序,或者销毁程序
98      while (1) { // 两个LED闪烁,每秒闪2.5次
99          HAL_GPIO_TogglePin(LED0_GPIO_Port, LED0_Pin);
100         HAL_GPIO_TogglePin(LED1_GPIO_Port, LED1_Pin);
101         printf("It's abnormal routine.\r\n");
102         HAL_Delay(200);
103     }
104 }
105 /* USER CODE END 2 */
```

ID 校验函数 check_ID()在文件 encrypt.c 和 encrypt.h 中实现。

Step7.实现 ID 校验功能。

encrypt.h 文件内容：

```
#ifndef _ENCRYPT_H_
#define _ENCRYPT_H_

#include "stm32f4xx_hal.h"

#define FLASH_SECTOR3_SIZE          0x4000          /* Sector 3 size: 16 Kbytes */
#define ADDR_FLASH_SECTOR_3    ((uint32_t)0x0800C000) /* Base @ of Sector 3, 16 Kbytes */

// chipID 地址
#define UID0    0X1F
#define UID1    0XFF
#define UID2    0X7A
#define UID3    0X10
// chipID 地址在 crc_tmp_data 中的存储坐标
#define UID0_OFFSET    0x23
#define UID1_OFFSET    0x33
#define UID2_OFFSET    0x43
#define UID3_OFFSET_1  0x53
#define UID3_OFFSET_2  0x63
#define UID3_OFFSET_3  0x73

// 通用密码
#define eID0    0xEF
#define eID1    0x3E
#define eID2    0x05
#define eID3    0x69
#define eID4    0xAB
#define eID5    0xC7
#define eID6    0xE5
#define eID7    0x31
#define eID8    0x05
#define eID9    0x59
#define eID10   0xA3
#define eID11   0x07
```

```
// 通用密码在 crc_tmp_data 中的存储坐标
#define eID0_OFFSET    0x09
#define eID1_OFFSET    0x19
#define eID2_OFFSET    0x29
#define eID3_OFFSET    0x39
#define eID4_OFFSET    0x49
#define eID5_OFFSET    0x59
#define eID6_OFFSET    0x0A
#define eID7_OFFSET    0x1A
#define eID8_OFFSET    0x2A
#define eID9_OFFSET    0x3A
#define eID10_OFFSET   0x4A
#define eID11_OFFSET   0x5A

void calculate_eCODE(uint8_t *pUID, uint8_t *peCODE);
void encrypt_code_save(void);
void read_ID(uint32_t *pUID_dat32);
HAL_StatusTypeDef check_ID(void);

#endif
```

encrypt.c 文件内容：

```
#include "encrypt.h"

// 密码存储区
const unsigned char encrypt_code[FLASH_SECTOR3_SIZE] __attribute__((at(ADDR_FLASH_SECTOR_3))) = {
  eID0, eID1, eID2, eID3, eID4, eID5, eID6, eID7, eID8, eID9, eID10, eID11,
};

// 通用密码和 CRC 计算数据
const unsigned char crc_tmp_data[128]={
//    0   1   2   3    4   5   6   7       8   9   A   B    C   D   E   F
/*0*/ 0x39, 0x18, 0xbb, 0x5a, 0xdd, 0x21, 0xff, 0xee,    0x62, eID0, eID6, 0x14, 0x66, 0x77, 0x44, 0x55,
/*1*/ 0x2b, 0x90, 0x21, 0x32, 0x42, 0x54, 0x67, 0x76,    0xea, eID1, eID7, 0x89, 0xfe, 0xef, 0x2c, 0xcd,
/*2*/ 0x00, 0x21, 0x22, UID0, 0x44, 0x55, 0xc6, 0x77,    0x88, eID2, eID8, 0xbb, 0xcc, 0x3a, 0xce, 0x2c,
/*3*/ 0x31, 0x10, 0x32, UID1, 0x55, 0x44, 0xf7, 0x66,    0x34, eID3, eID9, 0x01, 0xc3, 0x67, 0x54, 0x05,
/*4*/ 0xaa, 0xbb, 0x88, UID2, 0xf7, 0xff, 0xcc, 0x8d,    0x33, eID4, eID10,0x00, 0xd7, 0x66, 0x55, 0xb4,
/*5*/ 0x5b, 0xaa, 0x99, UID3, 0xdf, 0xf7, 0xdd, 0xfe,    0xff, eID5, eID11,0xcf, 0xbb, 0x3a, 0x99, 0x78,
/*6*/ 0x88, 0xa9, 0xca,UID3+4,0xdc, 0x2d, 0xfe, 0x6f,    0x2c, 0xa1, 0x30, 0x20, 0x50, 0x25, 0x76, 0x3a,
/*7*/ 0x48, 0x55, 0x87,UID3+8,0x08, 0x61, 0x22, 0x23,        0x66, 0x2c, 0x4c, 0x55, 0x2c, 0x33, 0x00, 0x41,
};

// 本机 ID 存储变量
volatile union {
        uint32_t dat32[3];
        uint8_t dat8[12];
      }mcuID;
// 本机密码存储变量
volatile uint8_t eCODE[12];

/*
函数名称: calculate_eCODE
功   能: 计算本机密码
      (本函数当前没有进行复杂的算法设计,只简单地进行按位取反)
输   入: pUID,UID 数组指针
输   出: peCODE,本机密码
返   回: 无
```

```
*/
void calculate_eCODE(uint8_t *pUID, uint8_t *peCODE)
{
   peCODE[0] = ~pUID[0];
   peCODE[1] = ~pUID[1];
   peCODE[2] = ~pUID[2];
   peCODE[3] = ~pUID[3];
   peCODE[4] = ~pUID[4];
   peCODE[5] = ~pUID[5];
   peCODE[6] = ~pUID[6];
   peCODE[7] = ~pUID[7];
   peCODE[8] = ~pUID[8];
   peCODE[9] = ~pUID[9];
   peCODE[10]= ~pUID[10];
   peCODE[11]= ~pUID[11];
}

/*
函数名称: encrypt_code_save
功　能: 保存密码到 FLASH 中
输　入: 无
返　回: 无
*/
void encrypt_code_save(void)
{
	uint32_t PAGEError = 0;
	uint32_t write_addr;
	uint32_t i;
	uint8_t  *pbuf;
	FLASH_EraseInitTypeDef EraseInitStruct;

	EraseInitStruct.TypeErase   = FLASH_TYPEERASE_SECTORS;
	EraseInitStruct.Banks               = FLASH_BANK_1;
	EraseInitStruct.Sector        = FLASH_SECTOR_3;
	EraseInitStruct.NbSectors   = 1;
	EraseInitStruct.VoltageRange= FLASH_VOLTAGE_RANGE_3;

	HAL_FLASH_Unlock();
	if (HAL_OK == HAL_FLASHEx_Erase(&EraseInitStruct, &PAGEError)) {
		write_addr = ADDR_FLASH_SECTOR_3;
		pbuf = (uint8_t *)eCODE;
		for (i=0; i<12; i++) {
			HAL_FLASH_Program(FLASH_TYPEPROGRAM_BYTE, write_addr + i, pbuf[i]);
		}
	}
	HAL_FLASH_Lock();
}

/*
函数名称: read_ID
功　能: 读取 ID
输　入: 无
输　出: 读到的 UID 存入 pUID_dat32 指向的存储空间
返　回: 无
*/
void read_ID(uint32_t *pUID_dat32)
{
	uint32_t mcuID_add;
```

```
    uint32_t tmp[4];

    // 31:0
    // 为了更安全,不使用用立即数访问 UID,
    // 所以将 UID 的地址分散在 crc_tmp_data 中,然后通过读取 crc_tmp_data 合成 UID 地址
    tmp[0] = *(__IO uint8_t*)&crc_tmp_data[UID0_OFFSET];
    tmp[1] = *(__IO uint8_t*)&crc_tmp_data[UID1_OFFSET];
    tmp[2] = *(__IO uint8_t*)&crc_tmp_data[UID2_OFFSET];
    tmp[3] = *(__IO uint8_t*)&crc_tmp_data[UID3_OFFSET_1];
    mcuID_add = (tmp[0]<<24) + (tmp[1]<<16) + (tmp[2]<<8) + tmp[3]; // 合成 UID 地址

    pUID_dat32[0] = *(__IO uint32_t*)(mcuID_add); // 读取 UID[31:0]

    // 63:32
    tmp[0] = *(__IO uint8_t*)&crc_tmp_data[UID0_OFFSET];
    tmp[1] = *(__IO uint8_t*)&crc_tmp_data[UID1_OFFSET];
    tmp[2] = *(__IO uint8_t*)&crc_tmp_data[UID2_OFFSET];
    tmp[3] = *(__IO uint8_t*)&crc_tmp_data[UID3_OFFSET_2];
    mcuID_add = (tmp[0]<<24) + (tmp[1]<<16) + (tmp[2]<<8) + tmp[3];

    pUID_dat32[1] = *(__IO uint32_t*)(mcuID_add);

    // 95:64
    tmp[0] = *(__IO uint8_t*)&crc_tmp_data[UID0_OFFSET];
    tmp[1] = *(__IO uint8_t*)&crc_tmp_data[UID1_OFFSET];
    tmp[2] = *(__IO uint8_t*)&crc_tmp_data[UID2_OFFSET];
    tmp[3] = *(__IO uint8_t*)&crc_tmp_data[UID3_OFFSET_3];
    mcuID_add = (tmp[0]<<24) + (tmp[1]<<16) + (tmp[2]<<8) + tmp[3];

    pUID_dat32[2] = *(__IO uint32_t*)(mcuID_add);
}

/*
函数名称: check_ID
功　能: 校验 ID
输　入: 无
返　回: HAL_OK,校验成功;HAL_ERROR,失败
*/
HAL_StatusTypeDef check_ID(void)
{
    // Step1.读取本机 ID,存入 mcuID 联合体
    read_ID((uint32_t*)mcuID.dat32);
    // Step2.使用本机 ID 计算本机密码,存入 eCODE 数组
    calculate_eCODE((uint8_t*)mcuID.dat8, (uint8_t*)eCODE);
    // Step3.对比 FLASH 中存储的密码是否等于通用密码
    if((crc_tmp_data[eID0_OFFSET] == *(__IO uint8_t*)&encrypt_code[0])
    && (crc_tmp_data[eID1_OFFSET] == *(__IO uint8_t*)&encrypt_code[1])
    && (crc_tmp_data[eID2_OFFSET] == *(__IO uint8_t*)&encrypt_code[2])
    && (crc_tmp_data[eID3_OFFSET] == *(__IO uint8_t*)&encrypt_code[3])
    && (crc_tmp_data[eID4_OFFSET] == *(__IO uint8_t*)&encrypt_code[4])
    && (crc_tmp_data[eID5_OFFSET] == *(__IO uint8_t*)&encrypt_code[5])
    && (crc_tmp_data[eID6_OFFSET] == *(__IO uint8_t*)&encrypt_code[6])
    && (crc_tmp_data[eID7_OFFSET] == *(__IO uint8_t*)&encrypt_code[7])
    && (crc_tmp_data[eID8_OFFSET] == *(__IO uint8_t*)&encrypt_code[8])
    && (crc_tmp_data[eID9_OFFSET] == *(__IO uint8_t*)&encrypt_code[9])
    && (crc_tmp_data[eID10_OFFSET]== *(__IO uint8_t*)&encrypt_code[10])
    && (crc_tmp_data[eID11_OFFSET]== *(__IO uint8_t*)&encrypt_code[11])
    ) {
```

```
    // 如果密码等于通用密码,则将计算的结果写入 FLASH
    encrypt_code_save();
    return HAL_BUSY;
  } else
  // Step4.对比计算的本机密码是否等于 FLASH 中存储的密码
  if((eCODE[0] == *(__IO uint8_t*)&encrypt_code[0])
  && (eCODE[1] == *(__IO uint8_t*)&encrypt_code[1])
  && (eCODE[2] == *(__IO uint8_t*)&encrypt_code[2])
  && (eCODE[3] == *(__IO uint8_t*)&encrypt_code[3])
  && (eCODE[4] == *(__IO uint8_t*)&encrypt_code[4])
  && (eCODE[5] == *(__IO uint8_t*)&encrypt_code[5])
  && (eCODE[6] == *(__IO uint8_t*)&encrypt_code[6])
  && (eCODE[7] == *(__IO uint8_t*)&encrypt_code[7])
  && (eCODE[8] == *(__IO uint8_t*)&encrypt_code[8])
  && (eCODE[9] == *(__IO uint8_t*)&encrypt_code[9])
  && (eCODE[10]== *(__IO uint8_t*)&encrypt_code[10])
  && (eCODE[11]== *(__IO uint8_t*)&encrypt_code[11])
  ) {
    // 如果密码校验正确,则不进行任何操作
    return HAL_OK;
  } else {
    // 如果密码校验错位,则关机或者销毁程序
    return HAL_ERROR;
  }
}
```

注意以下几个关键的地方：

1.密码存储区。

该数组占用了 FLASH 的整个扇区 3，编程时其初始值存放的是通用 ID 校验码。check_ID()函数检测到是通用密码后，会将其修改为本机密码。

```
// 密码存储区
const unsigned char encrypt_code[FLASH_SECTOR3_SIZE] __attribute__((at(ADDR_FLASH_SECTOR_3))) = {
    eID0, eID1, eID2, eID3, eID4, eID5, eID6, eID7, eID8, eID9, eID10, eID11,
};
```

2.通用密码存放数组。

```
// 通用密码和CRC计算数据
const unsigned char crc_tmp_data[128]={
//      0    1     2     3     4     5     6     7          8    9     A     B     C     D     E     F
/*0*/ 0x39, 0x18, 0xbb, 0x5a, 0xdd, 0x21, 0xff, 0xee,     0x62, eID0, eID6, 0x14, 0x66, 0x77, 0x44, 0x55,
/*1*/ 0x2b, 0x90, 0x21, 0x32, 0x42, 0x54, 0x67, 0x76,     0xea, eID1, eID7, 0x89, 0xfe, 0xef, 0x2c, 0xcd,
/*2*/ 0x00, 0x21, 0x22, UID0, 0x44, 0x55, 0xc6, 0x77,     0x88, eID2, eID8, 0xbb, 0xcc, 0x3a, 0xce, 0x2c,
/*3*/ 0x31, 0x10, 0x32, UID1, 0x55, 0x44, 0xf7, 0x66,     0x34, eID3, eID9, 0x01, 0xc3, 0x67, 0x54, 0x05,
/*4*/ 0xaa, 0xbb, 0x88, UID2, 0xf7, 0xff, 0xcc, 0x8d,     0x33, eID4, eID10,0x00, 0xd7, 0x66, 0x55, 0xb4,
/*5*/ 0x5b, 0xaa, 0x99, UID3, 0xdf, 0xf7, 0xdd, 0xfe,     0xff, eID5, eID11,0xcf, 0xbb, 0x3a, 0x99, 0x78,
/*6*/ 0x88, 0xa9, 0xca,UID3+4,0xdc, 0x2d, 0xfe, 0x6f,     0x2c, 0xa1, 0x30, 0x20, 0x50, 0x25, 0x76, 0x3a,
/*7*/ 0x48, 0x55, 0x87,UID3+8,0x08, 0x61, 0x22, 0x23,     0x66, 0x2c, 0x4c, 0x55, 0x2c, 0x33, 0x00, 0x41,
};
```

crc_tmp_data[]数组，其中隐藏了两个重要信息：UID 在 FLASH 中的地址，通用的 ID 校验码。根据破解的理论，为了增加破解难度，在使用 ID 进行加密的时候，最好不要用立即数读取 UID。而是先用变量合成 UID 所在地址再读取。

STM32F4 的 UID 在 FLASH 中的地址是 0x1FFF7A10 为起始地址的 96bit(12 个字节)。本例程将 0x1FFF7A10 拆分成 4 个字节，放到 crc_tmp_data[]数组中，校验 ID 时先读取出来，合成该地址，再读取 UID。具体操作详见 read_ID()函数。

作为演示例程，程序中 UID 和通用校验码的信息在 crc_tmp_data[]数组中的位置比较整齐有规律。在实际应用中可以随机放入，修改相应的位置坐标宏定义 UIDn_OFFSET 和 eIDn_OFFSET 即可。

3.比较 FLASH 的中数据应该注意的问题。

在 check_ID()函数的 Step3 比较 FLASH 存储的密码和通用密码，Step4 比较 FLASH 存储的密码和本机密码。这两个步骤中，比较的两个数据都来自 FLASH，在编译过程中，可能会被编译器优化成立即数。因为 encrypt_code[]数组在本例中是有可能被修改的，所以，下图的写法，不能得到预想的比较结果。因为在编译时，使用的是通用密码，编译器优化之后，认为 Step3 的条件永远成立，不能实现加密效果。

```
// Step3.对比FLASH中存储的密码是否等于通用密码
if((crc_tmp_data[eID0_OFFSET] == encrypt_code[0])
&& (crc_tmp_data[eID1_OFFSET] == encrypt_code[1])
&& (crc_tmp_data[eID2_OFFSET] == encrypt_code[2])
&& (crc_tmp_data[eID3_OFFSET] == encrypt_code[3])
&& (crc_tmp_data[eID4_OFFSET] == encrypt_code[4])
&& (crc_tmp_data[eID5_OFFSET] == encrypt_code[5])
&& (crc_tmp_data[eID6_OFFSET] == encrypt_code[6])
&& (crc_tmp_data[eID7_OFFSET] == encrypt_code[7])
&& (crc_tmp_data[eID8_OFFSET] == encrypt_code[8])
&& (crc_tmp_data[eID9_OFFSET] == encrypt_code[9])
&& (crc_tmp_data[eID10_OFFSET]== encrypt_code[10])
&& (crc_tmp_data[eID11_OFFSET]== encrypt_code[11])
) {
    // 如果密码等于通用密码,则将计算的结果写入FLASH
    encrypt_code_save();
    return HAL_BUSY;
} else
```

应该改为如下写法，这样可以避免编译器将 encrypt_code[]的访问优化成立即数。Step4 也是如此。

```
// Step3.对比FLASH中存储的密码是否等于通用密码
if((crc_tmp_data[eID0_OFFSET] == *(__IO uint8_t*)&encrypt_code[0])
&& (crc_tmp_data[eID1_OFFSET] == *(__IO uint8_t*)&encrypt_code[1])
&& (crc_tmp_data[eID2_OFFSET] == *(__IO uint8_t*)&encrypt_code[2])
&& (crc_tmp_data[eID3_OFFSET] == *(__IO uint8_t*)&encrypt_code[3])
&& (crc_tmp_data[eID4_OFFSET] == *(__IO uint8_t*)&encrypt_code[4])
&& (crc_tmp_data[eID5_OFFSET] == *(__IO uint8_t*)&encrypt_code[5])
&& (crc_tmp_data[eID6_OFFSET] == *(__IO uint8_t*)&encrypt_code[6])
&& (crc_tmp_data[eID7_OFFSET] == *(__IO uint8_t*)&encrypt_code[7])
&& (crc_tmp_data[eID8_OFFSET] == *(__IO uint8_t*)&encrypt_code[8])
&& (crc_tmp_data[eID9_OFFSET] == *(__IO uint8_t*)&encrypt_code[9])
&& (crc_tmp_data[eID10_OFFSET]== *(__IO uint8_t*)&encrypt_code[10])
&& (crc_tmp_data[eID11_OFFSET]== *(__IO uint8_t*)&encrypt_code[11])
) {
    // 如果密码等于通用密码,则将计算的结果写入FLASH
    encrypt_code_save();
    return HAL_BUSY;
} else
```

根据下图的主函数流程执行，

```
82   /* USER CODE BEGIN 2 */
83   res = check_ID();
84   if (HAL_OK == res) { // ID校验正确,则执行正常程序
85       while (1) { // 两个LED闪烁,每秒闪1次
86           HAL_GPIO_TogglePin(LED0_GPIO_Port, LED0_Pin);
87           HAL_GPIO_TogglePin(LED1_GPIO_Port, LED1_Pin);
88           printf("Hello! It's normal routine.\r\n");
89           HAL_Delay(500);
90       }
91   } else if (HAL_BUSY == res) {  // 第一次运行,使用通用密码
92       while (1) { // 只有一个LED闪烁,每秒闪1次
93           HAL_GPIO_TogglePin(LED0_GPIO_Port, LED0_Pin);
94           printf("It's running first time.\r\n");
95           HAL_Delay(500);
96       }
97   } else {  // ID校验错误,则执行异常程序,或者销毁程序
98       while (1) { // 两个LED闪烁,每秒闪2.5次
99           HAL_GPIO_TogglePin(LED0_GPIO_Port, LED0_Pin);
100          HAL_GPIO_TogglePin(LED1_GPIO_Port, LED1_Pin);
101          printf("It's abnormal routine.\r\n");
102          HAL_Delay(200);
103      }
104  }
105  /* USER CODE END 2 */
```

运行结果是，第一次运行将只有一个 LED 闪烁，每秒闪烁一次，并从串口输出“It’s running first time.”。之后再次复位或者重新上电，则有两个 LED 闪烁，每秒闪烁一次，并输出“Hello！It’s normal routine.”。如果通过 J-Link 等工具读出 FLASH 中的代码，然后下载到另一片 STM32F407 上，运行的结果是：两个 LED 闪烁，每秒闪烁 2.5 次，并输出“It’s abnormal routine.”。

在实际应用中，将上图的 98~103 行代码换成销毁程序的功能，就实现了用 ID 加密程序的目的。

可以在程序的多个地方插入密码验证的代码。如果设计的产品使用了其他带有唯一 ID 的芯片(如 DS18B20 等)，可以把加密的安全等级提到更高水平。那就是使用外部芯片的 ID 再进行一次加密。据说，有的 MCU 的 ID 是可能被修改和重置的。因此，使用外部芯片 ID 加密可多一层保护，加大破解难度。

S.D.Lu 于 深圳
2016 年 10 月