ALIENTEK

广州市星翼电子科技有限公司

修订历史

版本	日期	原因
V1.00	2019/08/15	首次发布
V1.01	2019/12/09	增加 NRF UID 自定义加密
V1.02	2019/04/09	增加 TI CC25XX 及新塘 51 单片机 UID 自定义加密

1 代码更新记录

P100 脱机下载器使用的是最新版本的 UID 加密代码，不受之前产品版本迭代的影响。
V1.2:

更新：

修正配置密钥长度为 4 字节、8 字节时，取用户自定义 ID 越界导致运算的密钥结果不正确的问题。

MINI-Pro 脱机下载器 V1.20B 版本固件后使用 V1.2 版本代码；

MINI 脱机下载器 V1.4A2 版本固件后使用 V1.2 版本代码；

影响：

前提条件：

(1).MINI 版脱机下载器在升级了固件到 V1.4A2 后，使得 MINI 脱机下载器内置的是 V1.2 版本代码;

(2).MINI-Pro 版脱机下载器在升级了固件到 V1.20B 后，使得 MINI 脱机下载器内置的是 V1.2 版本代码;

(3).密钥长度配置为 4 字节、8 字节的情况；

导致现象：

(1).由于修正了算法里密钥长度为 4/8 字节时的运算逻辑，会导致运算出的密钥结果（仅密钥长度为 4/8 字节受影响）

与 V1.1 版本前的代码运算结果不同！

解决办法：

(1).MINI-Pro 版脱机下载器已经升级到 V1.20B 版本固件的用户可以回退到旧版本固件继续使用。

(2).MINI 版脱机下载器已经升级到 V1.4A2 版本固件的用户可以回退到旧版本固件继续使用。

不受影响的情况：

(1).只要密钥长度为 12 字节均不受影响；

V1.1:(**此版本代码会对部分使用旧版本 MINI 脱机下载器的 UID 自定义加密功能的用户有影响，使用 MINI-Pro 脱机下载器的用户不受影响，请仔细查看更新内容**)

更新：

MINI 版脱机下载器 V1,3A5 版固件使用 V1.1 版本代码，V1.1 版本代码修复了 V1.0 版本调用 UID_Encryption_Key_Check(),UID_Encryption_Key_Calculate()使用大端模式有可能出现的内存越界问题。

影响：

前提条件：

(1).在升级了 MINI 版脱机下载器的固件到 V1.3A5 后，使用的是 MINI 脱机下载器内置的是 V1.1 版本代码;

(2).此前 使用 V1.3A4 及之前版本固件的 MINI 脱机下载器 配置为大端模式，且用户程序使用的是最初的 V1.0 版本代码；

导致现象：

(1).后续使用 V1.3A5 版本固件的 MINI 脱机下载器，开启了 UID 自定义加密功能，然后继续烧录包含了 V1.0 版本代码的用户程序，由于 V1.1 版本计算的大端模式密钥与 V1.0 版本计算的密钥不一致，因此会导致芯片中会出现会造成密钥验证不通过的问题！

解决办法：

(1).将用户程序中的 V1.0 版代码同步更新到 V1.1 版代码。

不受影响的情况：

(1).此前使用 小端模式 配置的用户产品不会受到 V1.1 版本代码更新以及 MINI 脱机下载器固件更新的影响；

(2).如果已经使用 V1.0 代码的产品后续使用 IAP 方式升级则不会受到影响；

(3).如果已经使用 V1.0 代码的产品后续使用 MINI 脱机下载器升级产品 且 不开启 UID 自定义加密功能重新更新密钥不会受到影响，但是如果使用脱机下载器升级产品的话，建议更新用户代码的了的 UID 自定义加密代码为 V1.1，同时开启 UID 自定义加密重新更新密钥；

(4)MINI-Pro 版脱机下载器首版固件中使用的是 V1.1 版本的代码，不受影响；

V1.0：

MINI 版脱机下载器 V1.3A4 版固件及之前的固件中均使用了 V1.0 版本的代码；

2 UID 自定义加密原理

本脱机下载器的 UID 自定义加密是基于芯片内的 UID（Unique ID，全球唯一 ID）通过特定算法进行加密的一种方法。具体原理如下：

一、用户通过配置软件对脱机下载器进行相关配置后，脱机下载器在烧录时获取目标芯片的 UID 及各项配置 通过特定算法为用户计算一个密钥存储在目标芯片中。

二、用户程序在上电后使用我们提供的代码的相关函数然后传入相同配置可以检查密钥的合法性；

三、由于每一颗芯片的 UID 都是唯一的，因此相同配置计算得到的密钥却不同。所以正常使用脱机下载器烧录的芯片 在使用【与脱机下载器相同配置+自身 UID】是能够通过 UID 合法性检查的。

四、当别人从正常产品上非法复制了用户代码到别的芯片使用，由于别的芯片的 UID 与原来的芯片不同，所以是无法程序中的密钥合法性检查的。所以一旦无法通过这个密钥合法性检查，程序便可以判定当前产品为盗版，从而选择运行相应针对盗版行为的代码，比如说让设备死机或功能失效等等，用户可自行发挥。

3 脱机下载器的 UID 自定义加密配置参数

可以在配置软件的“高级功能 -> UID 自定义加密”中进行配置，配置界面如图 1 所示：

图 1 UID 自定义加密配置界面

若要配置 UID 算法加密，首先要勾选“启用 UID 算法加密”，然后才能开始配置，各配置项描述如下：

- **密钥存储起始地址（Hex）**：指定计算得到的密钥的起始存储地址；
- **算法选择**：选择用于计算密钥的算法；
- **密钥字节长度**：指定计算与存储的密钥的长度；
- **端序模式**：设置密钥的存储方式按 字(4Byte)进行大端模式还是小端模式存储①。
- **自定义 ID（Hex）**：由用户妥善保存的 ID，参与密钥计算，其长度与密钥字节长度匹配。

注意：

①.举个例子，比如一个 12 字节长度的密钥：

0x11，0x22，0x33，0x44，0x55，0x66，0x77，0x88，0x99，0xAA，0xBB，0xCC，当以大端模式存储时，在存储器中的表现形式为：

11	22	33	44	55	66	77	88	99	AA	BB	CC

而以小端模式存储时，在存储器中的表现形式为：

44	33	22	11	88	77	66	55	CC	BB	AA	99

实际应用中，用户无需理会，只要统一用户程序中密钥验证的配置与脱机下载器的 UID 自定义加密配置一致便可；

4 UID 自定义加密代码介绍

要在产品上使用 UID 自定义加密功能，用户需在产品的代码中加入对应密钥合法性检查的功能，我们提供了与脱机下载器上的 UID 自定义加密功能完全匹配的代码 UID_Encryption.c 与 UID_Encryption.h（如图 2 所示），用户只需将这两个文件加入自己的工程，然后调用 UID_Encryption.h 中的函数进行密钥合法性检查便可。详细使用步骤情况第 4 节《UID 自定义加密使用演示》。源文件在附赠资料中有给出。

STM8 部分型号芯片具备 UID（具体型号请参考后文的芯片 UID 起始地址表，若表中没有展示，则不支持 UID 自定义加密功能），因此我们也为这部分信号的 STM8 芯片提供了 UID 自定义加密功能支持，我们提供的 STM8 与 STM32/GD32 的 UID 自定义加密功能的代码根据芯片的特性而有所差异，因此对不同的芯片类型，用户需要添加对应的 UID 自定义加密代码到自己的程序中。

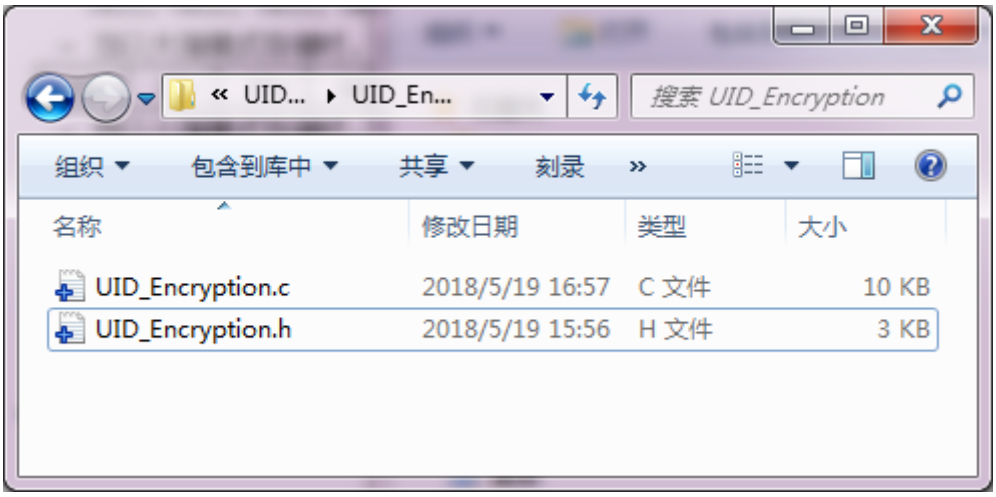

图 2　UID 自定义加密源文件

点击打开头文件 UID_Encryption.h，如图 3 所示：代码功能描述如下：

1. 第 4~10 行：　密钥长度取值枚举；
2. 第 12~17 行：　　大小端序选择取值枚举；
3. 第 20~28 行：　　算法选择取值枚举；
4. 第 47~52 行：　　UID 自定义加密密钥验证函数 UID_Encryption_Key_Check()；
5. 第 68~73 行：　　UID 自定义加密密钥计算函数 UID_Encryption_Key_Caculate()；

用户在使用时调用 UID_Encryption_Key_Check()函数、传入与脱机下载器同样的配置及芯片自身的 UID 地址（本文末尾附有各系列芯片 UID 起始地址），函数便可返回密钥检查结果。

```
#pragma once
#include "string.h"

typedef enum
{
    LENGTH_4 = 4,
    LENGTH_8 = 8,
    LENGTH_12 = 12

}eKeyLengthType;

typedef enum
{
    LITTLE_ENDIA = 0,
    BIG_ENDIA

}eEndiaType;


typedef enum
{
    ALGORITHM_0 = 0,
    ALGORITHM_1,
    ALGORITHM_2,
    ALGORITHM_3,
    ALGORITHM_4

}eAlgorithmType;


/****************** 调用以下函数完成密钥验证 ******************/
/*
char UID_Encryption_Key_Check(void *pKey,                       //[IN]
                              void *pUID,                       //[IN]
                              void *pCustomID,                  //[IN]
                              eKeyLengthType keyLength,         //[IN]
                              eEndiaType endiaType,             //[IN]
                              eAlgorithmType AlgorithmNum);     //[IN]
/****************** 调用以下函数完成密钥验证 ******************/
/*
void UID_Encryption_Key_Calculate(void *pKey,                   //[OUT]
                                  void *pUID,                   //[IN]
                                  void *pCustomID,              //[IN]
                                  eKeyLengthType keyLength,     //[IN]
                                  eEndiaType endiaType,         //[IN]
                                  eAlgorithmType AlgorithmNum);//[IN]
```

1 密钥字节长度枚举

2 端序枚举

3 算法枚举

4 密钥合法性检查函数

5 密钥计算函数

图 3 UID_Encryption.h 头文件代码

5 UID 自定义加密使用演示

此处使用 STM32 演示如何通过 UID 自定义加密算法功能对目标芯片进行加密，STM8 应用方法也是一致的。
演示平台：正点原子 Mini-STM32 开发板
芯片型号：STM32F103RCT6
使用代码：（附赠资料中均有给出）

①. 正点原子官方例程《ALIENTEK MINISTM32 实验 11 TFT LCD 液晶显示实验》;
②. 用于 UID 自定义加密的源文件 UID_Encryption.c 及其头文件 UID_Encryption.h。

具体应用过程如下：

一、首先打开《ALIENTEK MINISTM32 实验 11 TFT LCD 液晶显示实验》工程，添加如图 3-2 所示的源文件 UID_Encryption.c 到工程中,无需修改该文件中内容。如图 4 所示：

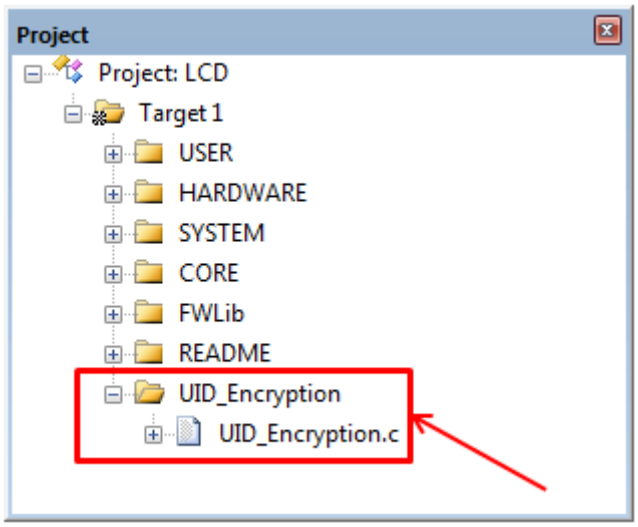

图 4 将 UID 自定义加密源文件加入工程

二、在程序 main.c 文件中增加以下代码：

1) 包含 UID 自定义加密函数声明的的头文件 UID_Encryption.h.如图 5 所示：

```
6   /* 包含UID加密头文件 */
7   #include "UID_Encryption.h"
```

图 5 包含 UID_Encryption.h

2) 定义必要的参数，UID 的地址，密钥存储的地址，自定义 ID，如图 6 所示：

```
/*
*密钥在当前芯片中的存储地址
*（注意：用户使用时必须按需修改该值，避免占用用户代码的位置及超过芯片flash的范围并且与上位机配置秘钥存储地址一致）
*/
#define KEY_LOCATION    0x08020000

/*
*当前芯片UID所在地址（注意：根据不同的芯片的地址各不一样,用户得按需修改）
*如果芯片的UID不连续的话，请用户将UID拼凑到一段连续的内存里面方可使用UID_Encryption_Key_Check()接口
*/
#define UID_LOCATION    0x1FFFF7E8

u8 myID[12] = {
                0x01, 0x02, 0x03, 0x04,
                0x05, 0x06, 0x07, 0x08,
                0x09, 0x0A, 0x0B, 0x0C};//用户自定义ID
```

图 6　定义必要参数

注意：这些参数在上位机配置 UID 加密时需要使用到。

3) 调用 UID_Encryption_Key_Check()验证密钥的合法性，如图 7 所示，详细情况请打开例程，参照代码。本例程的具体传入参数如下：

密钥位置：KEY_LOCATION = 0x08020000（该地址需根芯片及用户的需求合理分配，不能超过 flash 的地址，这里仅为示例参考）

UID 位置：UID_LOCATION = 0x1FFFF7E8（本文末尾附有各系列芯片 UID 起始地址）

自定义 ID：myID = {0x01, 0x02, 0x03, 0x04,
0x05, 0x06, 0x07, 0x08,
0x09, 0x0A, 0x0B, 0x0C}

端序模式：LITTLE_ENDIA = 小端序

密钥字节长度：LENGTH_12 = 12

算法选择：ALGORITHM_2 = 2

```
39  /**** 验证密钥是否有效 start ****/
40
41      checkResult = UID_Encryption_Key_Check(  (void*)KEY_LOCATION,
42                                               (void*)UID_LOCATION,
43                                                myID,
44                                                LENGTH_12,
45                                                LITTLE_ENDIA,
46                                                ALGORITHM_2);
47
48      if(checkResult != 0)//如果验证结果为非零值，则验证不通过
49      {
50          printf("\r\n密钥非法!\r\n");
51          LCD_ShowString(30,40,200,24,24,"Invalid Key !!!");
52          LED0=0;
53          while(1)
54          {
55              delay_ms(100);
56              LED0=!LED0;
57              LED1=!LED1;
58          }
59      }
60      printf("\r\n密钥合法!\r\n");
61
62  /**** 验证密钥是否有效 end ****/
```

验证密钥合法性

图 7　调用 UID_Encryption_Key_check()检查密钥

三、编译工程，得到固件后上传到配置软件中。其他的配置具体流程在《MINI-PRO 脱机下载器用户使用手册-通用功能介绍》中的第 2 节《简单使用步骤》已经演示过，这里主要演示如何配置软件上的 UID 算法加密功能。点击“高级功能 -> UID 算法加密”， 将上面的各项配置为 与 例程中传入给 UID_Encryption_Key_Check()的配置为完全一致（UID 地址不需要），然后点击“保存”如图 8 所示：

图 8 配置脱机下载器 UID 自定义加密功能

四、将所有功能项（包括基础功能项）配置完毕后，同样的点击开始编程将配置导入到脱机下载器，然后使用脱机下载器对 Mini-STM32 开发板进行烧录，当烧录完毕后开发板上的程序运行后我们可以看到串口打印输出显示“密钥合法”，如图 9 所示：

图 9 串口打印“密钥合法!”

五、上一步中可以看到通过脱机下载器使用相同配置的 UID 自定义加密算法对目标芯片进行加密后，程序能够验证密钥通过正常运行。接下来我们借助软件“STM32 ST-LINK Utility”与 ST-LINK 仿真器将开发板内的固件读出，然后使用“STM32 ST-LINK Utility”与 ST-LINK 仿真器烧录到另一个开发板（如果使用本脱机下载器来烧录，把 UID 算法加密功能关闭），当烧录完毕可以看到串口打印输出显示“密钥非法”，如图 10 所示：

图 10 串口打印“密钥非法!”

由此可得出结论：**正是由于前后两块开发板上的 STM32F103 芯片中的 UID 不一致，所用后一块开发板程序上电后算出的密钥与存储的密钥不一致， 导致密钥合法性检查无法通过。**因此通过这种方法就能够让用户的程序只能在指定 UID 的芯片运行，防止非法复制用户的劳动成果。

注意：

①.用户的配置不要泄露，其中的自定义 ID 实际上相当于一个用户自行持有的私钥。

②.用户配置参数可加密存储，使用时再解密传入给密钥验证函数；

③.用户可以对 UID_Encrption.c 的代码适当地进行一些删减，但是最好不要改动使用的算法，如果使用的算法被修改，则对密钥做合法性检查时计算得出的密钥将与脱机下载器烧录的密钥不一致，从而导致密钥合法性检查失败。

6 芯片 UID 起始地址表

芯片类型	芯片系列	芯片型号	UID 起始地址
STM32	STM32F0XX	-	0x1FFFF7AC
	STM32F1XX	-	0x1FFFF7E8
	STM32F2XX	-	0x1FFF7A10
	STM32F3XX	-	0x1FFFF7AC
	STM32F4XX	-	0x1FFF7A10
	STM32F7XX	-	0x1FF0F420 0x1FF07A10(F72xx,F73xx)
	STM32L0XX	-	0x1FF80050
	STM32L1XX	-	0x1FF80050 (cat.1/2) 0x1FF800D0(cat.3/4/5/6) Offset 0x00：UID[0:31] Offset 0x04：UID[32:63] Offset 0x14：UID[64:95]
	STM32L4XX	-	0x1FFF7590
GD32	GD32F10X	-	0x1FFFF7E8
	GD32F1X0	-	0x1FFFF7AC
	GD32F20X	-	0x1FFFF7E8
	GD32F30X	-	0x1FFFF7E8
	GD32F3X0	-	0x1FFFF7AC
	GD32F4XX	-	0x1FFF7A10
	GD32E103	-	0x1FFFF7E8
	GD32E230	-	0x1FFF F7AC
STM8	STM8S	STM8S003xx	0x00004865
		STM8S103xx	0x00004865
		STM8S105xx	0x000048CD
		STM8S207xx	0x000048CD
		STM8S208xx	0x000048CD
		STM8S903xx	0x00004865
	STM8L	STM8L001J3	0x00004925
		STM8L101xx	0x00004925
		STM8L15xxx	0x00004926
		STM8L16xxx	0x00004926
	STM8AL	STM8AL31xx/3Lxx	0x00004926
	STM8TL	STM8TL52x4/53x4	0x00004925
MM32	M32F00XX	-	0x1FFFF7E8
	M32F0XX	-	0x1FFFF7E8

	M32L0XX	-	0x1FFFF7E8
	M32L3XX	-	0x1FFFF7E8
	M32W0XX	-	0x1FFFF7E8
	M32W3XX	-	0x1FFFF7E8
	M32SPINXXX	-	0x1FFFF7E8
	M32F10XX	-	0x1FFFF7E8
Nordic	NRF51XX	-	0x10000060
	NRF52XX	-	0x10000060
TI	CC2540/41	-	0x780E
	CC2530/31/33	-	0x780C
NUVOTON	N76EXX	-	通过 IAP 对应命令读取
	MS51XX	-	通过 IAP 对应命令读取
	ML51XX		通过 IAP 对应命令读取

表 1 UID 起始地址表

表中数据由正点原子团队归纳得出，各芯片系列的 UID 地址通常在芯片厂商官方提供的参考手册中有描述，或者在库函数中有相应定义，如用户发现与官方描述不相符，还请指出，我们将及时修正。

7 联系我们

1、 官方店铺

官方店铺 1：http://shop62103354.taobao.com
官方店铺 2：http://shop62057469.taobao.com

2、 资料下载

资料链接：http://www.openedv.com/thread-300101-1-1.html

3、技术支持

技术论坛：www.openedv.com
官方网站：www.alientek.com
联系电话：020-38271790