SMEC98SP 加密芯片事例程序说明

本事例程序提供了两部分的代码：

1. SMEC98SP 加密芯片的代码；
2. 外部 MCU 的代码。

事例提供了 MCU 借助于加密芯片来保护方案不被外界破解的典型例子。开发者可以根据自己的需求，定义出适合自己的加密方案。

目录

功能说明.....2
1. 获取 SMEC98SP 的 UID 号.....2
2. 产生 MCU 及加密芯片的随机数.....2
3. 验证 PIN.....3
4. 内部认证.....3
5. 外部认证.....4
6. SHA1 哈希算法认证.....4
7. 关键算法放在加密芯片内.....5
8. 构造算法.....6
9. 密文读数据.....7
10. 读数据.....8
11. 写数据.....8
名词解释：.....8
DES 算法.....8
3DES 算法.....9
过程密钥.....10
密钥分散.....10
附件 1 – 例程密钥值.....10
附件 2 – SMEC98SP 样例指令.....11

功能说明

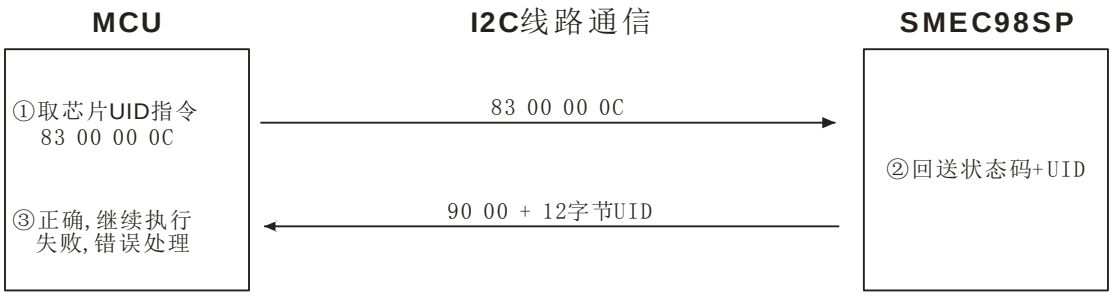
1. 获取 SMEC98SP 的 UID 号

事例程序提供了通过 I2C 通讯获取 SMEC98SP 加密芯片的 12 字节硬件 ID 号的方法。

利用此 ID 号，可以拓展一些应用，如：一卡一密等，即每个芯片可以设计成不一样的密钥，而只需一套主密钥。

一卡一密的设计，可借用金融 PBOC 卡的密钥分散概念来设计，即：利用主密钥，对芯片 ID 做 3DES 运算，并将运算结果作为该卡的密钥。

示意图：



获取SMEC98SP的UID号

2. 产生 MCU 及加密芯片的随机数

SMEC98SP 加密芯片具有硬件随机数发生器，本例程提供了获取 SMEC98SP 随机数的方法。

一般的单片机很少有硬件随机数发生器，所以我们针对不同类型的 MCU 设计了不同的随机数产生方式：

- 具有 A/D 数模转换的 MCU 随机数的产生：

利用 ADC 悬空引脚（通过读取悬空模拟针脚值）产生随机数种子，再将该随机数种子，与 MCU 的 UID 和加密芯片的 UID 作运算（异或运算），使得即使相同情况下，使用不同的 MCU 或加密芯片，其随机数种子也不相同。

3. 验证 PIN

PIN 码原理：MCU 和加密芯片分别存放着相同的 PIN 码，MCU 运行时，可以通过 I2C 发送 PIN 码给加密芯片进行验证，如果 PIN 码相同，且返回结果与预期一致，则认为加密芯片合法，否则认为加密芯片非法，而停止工作。

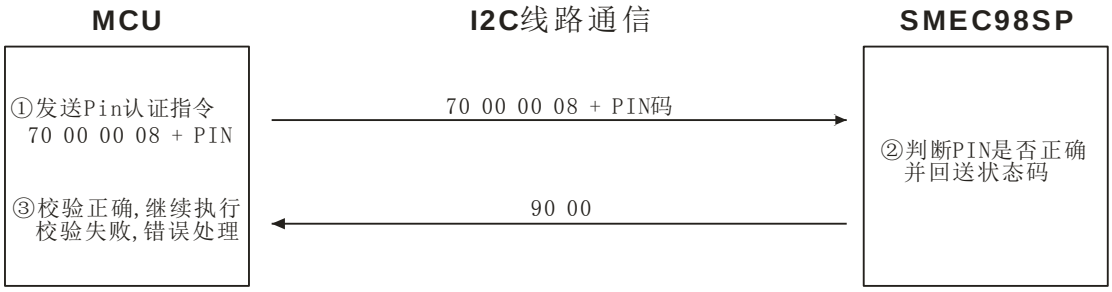
您可以利用 SMEC98SP 的 UID，将 PIN 码验证优化一下，如：可以将每个加密芯片的 PIN 码设计成跟 UID 相关，MCU 先获取 SMEC98SP 的 UID，然后计算出对应的 PIN 码值，再进行验证，再判断与预期是否一致。

优点：PIN 码验证实现简单，不需要添加额外的算法代码。

缺点：PIN 码会在 I2C 线路上传输，容易被攻击者监听并破解。也可被攻击者跳过“真值点”攻击。

安全程度：★★

示意图：



验证PIN

4. 内部认证

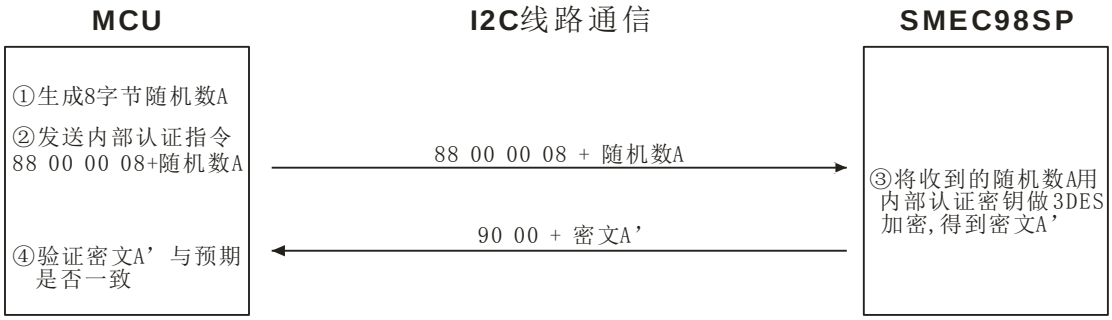
内部认证原理：MCU 向 SMEC98SP 发送 8 字节随机数, SMEC98SP 用内部认证密钥将随机数进行 3DES 加密后回送给 MCU, 由 MCU 判断回送数据的合法性。

优点：密钥不会在线路上传输，不怕攻击者监听 I2C 通讯数据。

缺点：密钥值会存放在 MCU 中，如果攻击者将 MCU 解密，并仿真调试，有可能找出该密钥。也可被攻击者跳过“真值点”攻击。

安全程度：★★★★

示意图：



内部认证

5. 外部认证

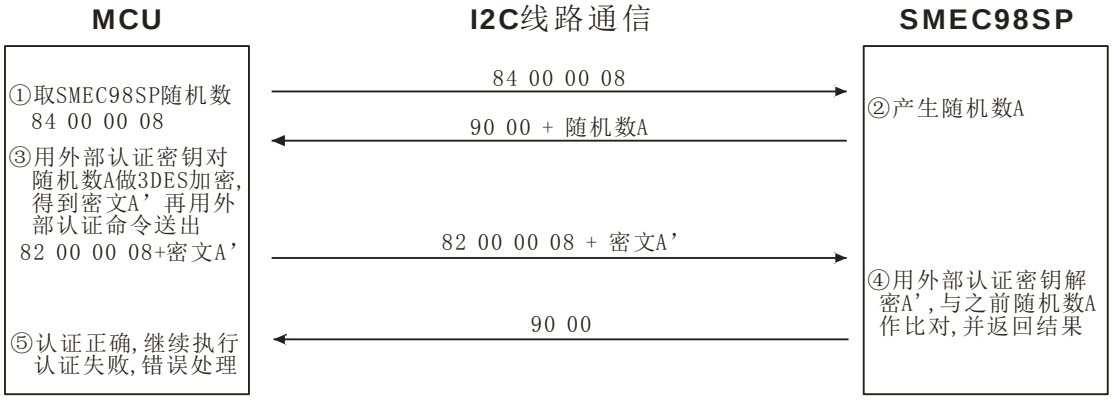
外部认证原理：MCU 先获取 SMEC98SP 的 8 字节随机数，然后 MCU 用外部认证密钥对随机数做 3DES 加密，再将密文送给 SMEC98SP，然后由 SMEC98SP 判断该密文的合法性。

优点：密钥不会在线路上传输，不怕攻击者监听 I2C 通讯数据。

缺点：密钥值会存放在 MCU 中，如果攻击者将 MCU 解密，并仿真调试，有可能找出该密钥。也可被攻击者跳过“真值点”攻击。

安全程度：★★★★

示意图：



外部认证

6. SHA1 哈希算法认证

SHA1 哈希算法认证原理：SHA1 为一种摘要算法，就是把任意长度的输入，通过散列算法，变换成固定长度（32 字节）的输出，该输出就是摘要值。

哈希算法具有单向性，即通过一组输入数据，可以得到一组固定长度的输出（摘要），

但通过输出，是不可以还原输入值。

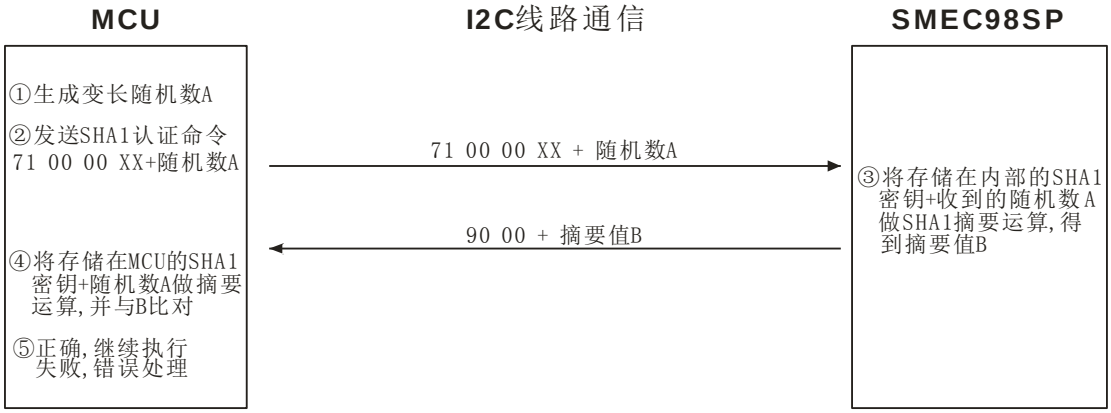
SHA1 利用这种特性，可以将输入数据前面的一部分作为密钥值，分别存放于 MCU 及加密芯片中。MCU 在做 SHA1 算法认证时，只需将部分数据传给加密芯片，由 SMEC98SP 内部再将“密钥值”+ “输入数据”一起做 SHA1 运算，并回送“摘要值”，MCU 再判断跟预期的值是否一致，实现 SHA1 算法认证。

优点：密钥不会在线路上传输，不怕攻击者监听 I2C 通讯数据。密钥长度及每次输入数据长度可以变化

缺点：密钥值会存放在 MCU 中，如果攻击者将 MCU 解密，并仿真调试，有可能找出该密钥。也可被攻击者跳过“真值点”攻击。

安全程度：★★★★

示意图：



SHA1哈希算法认证

7. 关键算法放在加密芯片内

将 MCU 中的一部分关键代码，放入加密芯片中运行，当需要用到 SMEC98SP 中的算法时，由 MCU 向 SMEC98SP 发送指令，SMEC98SP 根据指令，在内部运行，返回结果给 MCU。数据在 I2C 线路上传输,可以使用过程密钥加密的方式传输。

我们例程中写了一个算圆周长的简单示例，具体实现如下：

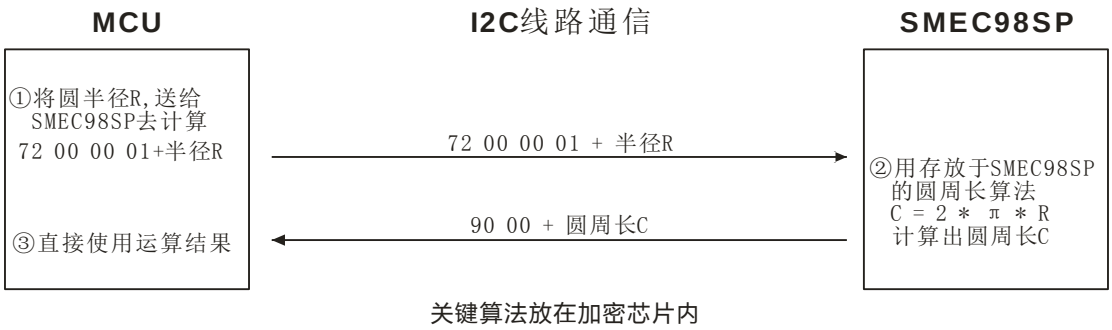
1. 加密芯片中存储算圆周长关键算法(周长 C = 2 * π * R)
2. 由 MCU 发送算圆周长指令：72 00 00 01 03 （R = 03）
3. 加密芯片根据 R 值，利用周长公式，算出周长 0x12，返回给 MCU。

优点：关键算法在加密芯片中，即使 MCU 被破解，并被理解反汇编代码，也无济于事。

缺点：暂无

安全程度：★★★★★

示意图：



8. 构造算法

针对很多控制类需求，没有"关键算法"可以存放在加密芯片中，例程中构造了一个算法：取 PA 端口数据 2 字节，用过程密钥加密后，送给 SMEC98SP，再由 SMEC98SP 解密后取反，再由过程密钥加密回送给 MCU。这样就“构造”出一个算法。

如 PA = 0x0000， 用过程密钥加密送给 SMEC98SP， SMEC98SP 解密后得到 0x0000， 取反后为 0xFFFF， 再用过程密钥加密给 MCU，主控 MCU 解密后得到 0xFFFF。这样， 判断 IO 口数据方式，只要跟之前相反就可以. 比如说 PA0 高电平才做的动作，调用了这个函数后，则判断 PA0 为低电平去做。

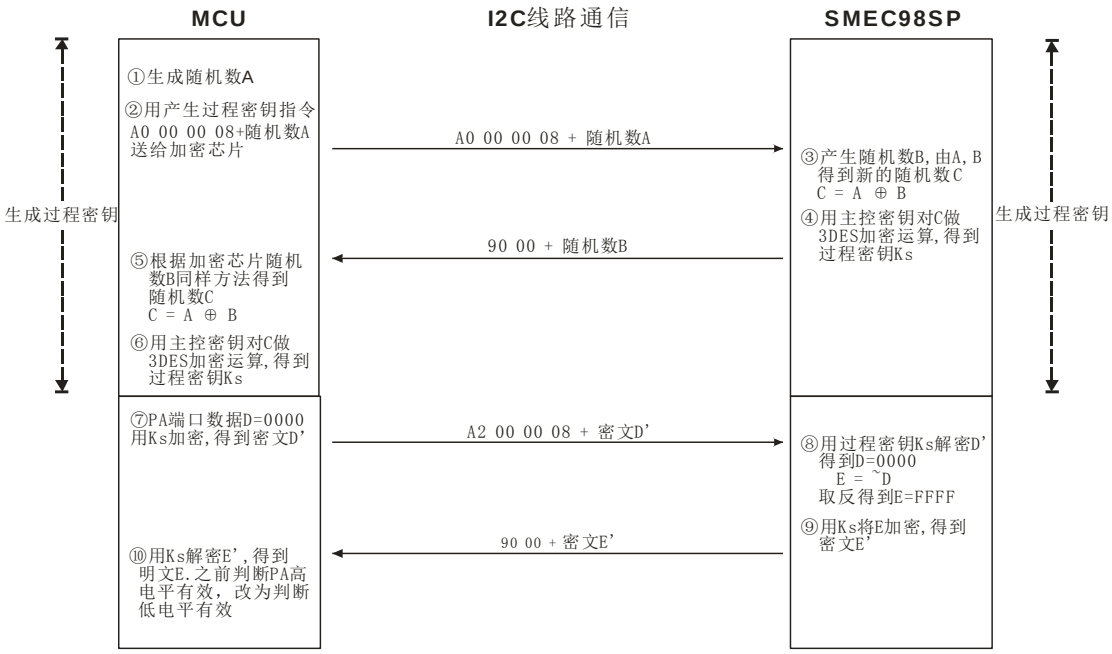
由于每次上电，过程密钥是临时产生的， 并且是变化的，这样即使 PA 口数据相同，在线路上通讯的数据也是不同的，而 MCU 程序又是基于"运算结果"而工作的，从而增加了破解难度，可以防止"真值点"攻击。

优点：数据被过程密钥加密传出，即使相同数据，每次在线路上传输的内容也不一样。

缺点：毕竟是“构造”出来的算法，算法复杂度不高。攻击者拿到 MCU 的汇编代码后，仿真调试，还是有可能将构造算法逆推。

安全程度：★★★★☆

示意图：



构造算法

9. 密文读数据

SMEC98SP 内部有 24K 字节程序区及 8K 字节数据区域。其中 8K 字节的数据区域可以设计成自由度写，也可以设计成需要一定权限（如验证 PIN，内部认证等）才能读写，还可以设计成密文方式读写。

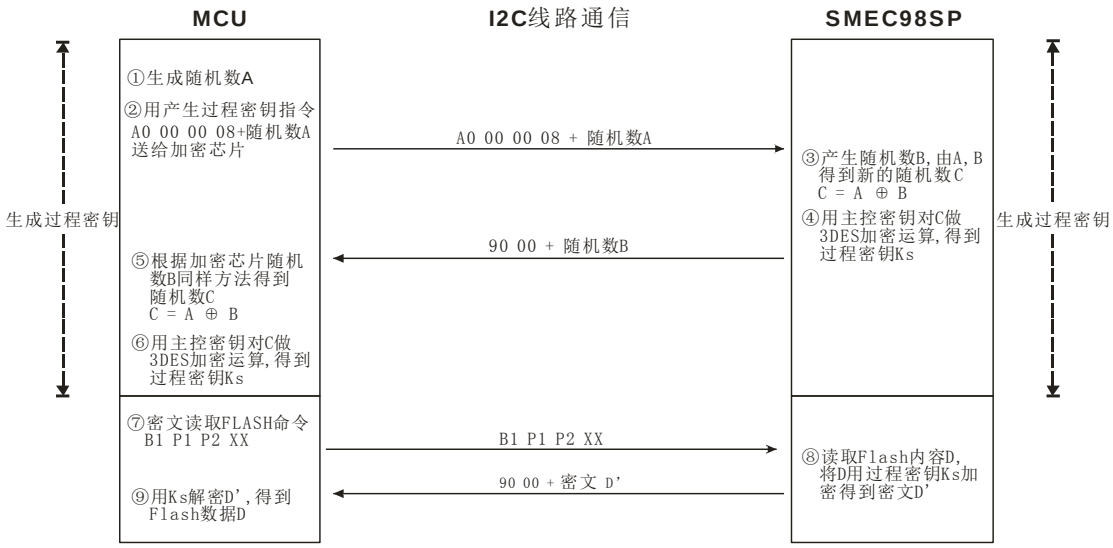
样例中，设计了利用过程密钥密文读取数据方法。由于过程密钥每次是变化的，这样即使是相同的数据，每次读出来，在线路上传输的内容也是不一样的。

优点：数据被过程密钥加密传出，即使相同数据，每次在线路上传输的内容也不一样。

缺点：密钥值会存放在 MCU 中，如果攻击者将 MCU 解密，并仿真调试，有可能找出该密钥。

安全程度：★★★★

示意图：



密文读数据

10.读数据

事例程序提供了明文读取 SMEC98SP 中 8K 数据区域的方法，样例中数据的起始地址是以字节为单位。开发者可以根据自己的情况，决定是否允许读取数据或者读取数据的相关权限。

11.写数据

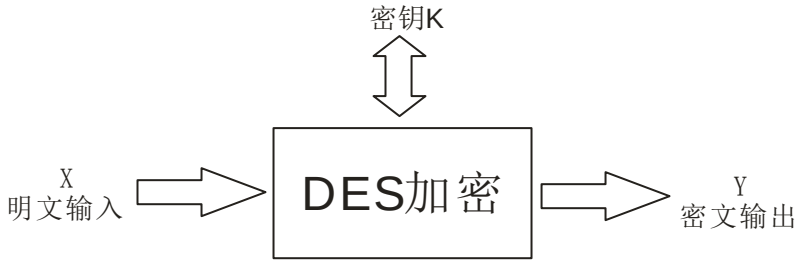
事例程序提供了明文写 SMEC98SP 数据区域的方法，样例中数据的起始地址是以字节为单位。开发者可以根据自己的情况，决定是否允许写入数据或者写入数据的相关权限。

名词解释：

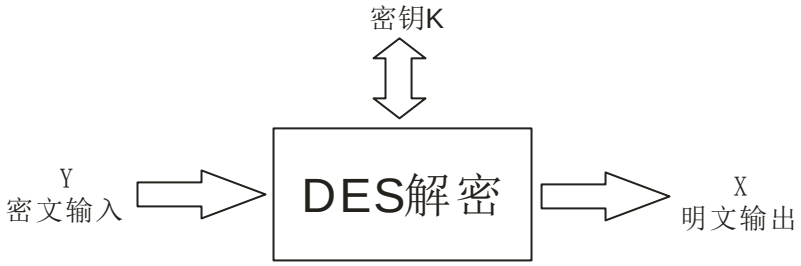
DES 算法

DES 算法是一种比较传统的加密方式，其加密运算、解密运算使用的是同样的密钥，信息的发送者和信息的接收者在进行信息的传输与处理时，必须共同持有该密码（称为对称密码），是一种对称加密算法。DES 是安全性比较高的一种算法，除了穷举外，没有其他方

法可破解。DES—密钥长度为 8 字节，数据为 8 字节。



DES加密算法: **Y=DES(K)[X]**

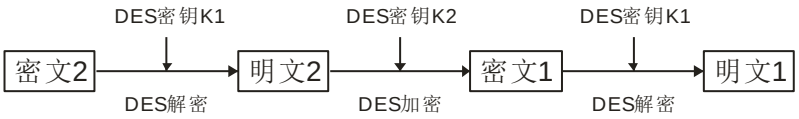


DES解密算法: **X=DES^{-1}(K)[Y]**

3DES 算法

3DES 是针对 DES 算法密钥过短、而改进的一个措施，被称为“3DES”。其实是通过执行 3 次 DES 来达到增加密钥长度和安全。3DES—密钥长度为 16 字节，数据为 8 字节。

3DES加密运算



3DES解密运算

过程密钥

过程密钥是一种会话密钥(session key)，是在使用过程中，基于主密钥而临时生成的加解密密钥，会话结束后，过程密钥失效。

密钥分散

密钥分散算法简称 Diversify，是指将一个双长度的密钥 MK，对分散数据(也叫分散因子，如：卡号等)进行处理，推导出一个双长度的密钥 DK。

推到 DK 左半部分的方法是：

1、将分散因子的最右 8 个字节作为输入数据；

2、将 MK 作为加密密钥；

3、用 MK 对输入数据进行 3DES 运算；

推到 DK 右半部分的方法是：

1、将分散因子的最右 8 个字节求反，作为输入数据；

2、将 MK 作为加密密钥；

3、用 MK 对输入数据进行 3DES 运算；

这样 MK 根据分散因子，可以得到分散密钥 DK。但根据 DK，却无法得到 MK。

附件 1 – 例程密钥值

事例程序中，MCU 及 SMEC98SP 加密芯片的各密钥值如下：

内部认证密钥：00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

外部认证密钥：10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F

SHA1 哈希算法认证密钥：20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F

产生过程密钥的主控密钥：30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F

PIN 码：55 66 77 88 99 AA BB CC

声明：本事例程序，只是提供了如何利用 SMEC98SP 保护方案不被破解的样例，开发者可以根据自己的需求，参照本例程，设计自己的加密保护方案。千万不要使用我们例程同样的密钥值！另外在 MCU 中，我们建议每组密钥值请打散后分开存放于不同地方，这样即使在攻击者把 MCU 解密后，也很难拼凑出完整的密钥值。

附件 2 – SMEC98SP 样例指令

```
;Pin 认证
--> 70 00 00 08 55 66 77 88 99 aa bb cc
<-- 9000

;哈希算法认证
--> 71 00 00 10 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
<-- 9000 3BD683BFC40F33254DE1BD6D0C2650BA1D07A56A

;实现一个简单算法，计算圆周长，外部输入的是半径，计算出周长。C = 2*3.14*R
--> 72 00 00 01 02
<-- 9000 0C

;外部认证,先取随机数,再做认证
--> 84 00 00 08
<-- 9000 D309685B156B398B          ;随机数每次不同
--> 82 00 00 08 7fa49dbf221a8751   ;根据随机数,外部认证密钥计算
<-- 9000

;取芯片唯一序列号
--> 83 00 00 0C
<-- 9000 33D60D08844C905016707726   ;每个加密芯片不同 ID 号

;取随机数
--> 84 00 00 08
<-- 9000 27F5EEDC149FE9FB

;内部认证
--> 88 00 00 08 11 22 33 44 55 66 77 88   ;假设主控芯片随机数 1122334455667788
```

<-- 9000 56BE32F01E736D0D

;产生过程密钥 先取 SMEC98SP 的随机数，再送入随机数，计算出过程密钥
--> A0 00 00 08 11 22 33 44 55 66 77 88 ;假设主控芯片随机数 1122334455667788
<-- 9000 2E2AB595F5186905 ;SMEC98SP 随机数每次不同
;过程密钥为:fede17ed454baa2e (303132333435363738393A3B3C3D3E3F =>(1122334455667788 ^ 2E2AB595F5186905)3DES 加密结果)

;端口数据运算
--> A2 00 00 08 df51daa6c1fdb929 ;假设端口数据为:0000，过程密钥:fede17ed454baa2e,(传入数据:df51daa6c1fdb929 为过程密钥 fede17ed454baa2e 对 0000000000000000(端口数据 0000+6 字节 00 填充)做 DES 加密)
<-- 9000 C0E7FECFABB13417 ;输出数据 C0E7FECFABB13417,用过程密钥解密后为 ffffffffffffffff，即为 0000000000000000 的取反"算法"，该"算法"可以根据需要修改

;读取 FLASH
--> B0 00 00 10
<-- 9000 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

;密文读取 FLASH
--> B1 00 00 10
<-- 9000 C0E7FECFABB13417C0E7FECFABB13417 ;用过程密钥 fede17ed454baa2e 解密后为 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

;写 FLASH
--> D6 00 00 10 0102030405060708090A0B0C0D0E0F10
<-- 9000