

# 利用ROCKEY6 SMART 轻松实现软件保护

飞天ROCKEY6 SMART加密锁是第六代可编程加密锁，硬件上使用32位智能卡芯片，软件上集成了COS和虚拟机技术。开发商可以把程序中的一部分代码移植到加密锁中，使之在加密锁中运行，这样这部分代码运行时就不会在计算机的内存中出现，也就杜绝了程序被跟踪调试和转储的可能，因此，加密锁中的代码越复杂，其破解的可能性就越微乎其微，再利用ROCKEY6 SMART系列产品的远程管理机制，可以对软件经常升级，软件的加密机制更加可靠，因此可以称之为“不可破解的加密锁”。

虽然理论上利用ROCKEY6 SMART我们可以达到不可破解，但我们如果加密方式做的不好，那么同样会存在安全隐患。下面我们先介绍几种不好加密方式：

## 简单查找模式

这种方法只是简单判断一下加密锁是否存在，如果存在则继续执行，不存在则返回错误。

## 存储器读数判别

在软件开发商使用存储型加密锁，多会出现这种不良加密方式。相应的，在其它允许用户存贮数据的加密锁的使用中，这种不良加密方案也时常会被使用。在这种加密方式中，软件开发商只是简单的读取加密锁中事先存贮的数据，然后与程序中的数据进行比较，或者在程序中直接使用这些数据。这种使用方式，因为与加密锁的交互接口非常明显，很容易就被找到，从而找到加密数据。一旦找到所有数据，解密者就可能模拟它们，而破解软件。

我们建议这样使用具有存储功能的加密锁，要被加密程序的“写入数据与读取数据相分离”，也就是说把存储型加密锁做为程序运行空间的一个特殊内存来使用，这块内存可以在程序进程间共享数据，在程序中一个地方写入，在另一个地方使用它。比如，我们在程序一开始就写入一个随机数，程序中的任何数据

交互都要去取这个随机数，然后经过它的处理，再传递。这样就分离了加密锁的数据读写。还有这样做的好处就是即使解密者知道了这样的加密方式，要解密也是很困难的，因为要处理很多地方的代码，很容易破坏程序的地址偏移，使解密失败。

## 封装接口过于明显

软件开发商在使用加密锁中，为了在其它编译器代码模块，如VB、Dephi中也能调用加密锁验证代码，常常把加密锁验证封装在一个DLL中，并且导出一个验证函数。

## 过分依赖外壳加密

软件开发商在应用加密锁时，总是喜欢用加密锁提供的外壳加密工具，直接对生成的可执行程序加密，认为这样即快捷，又安全。但事实却不是这样的。

外壳加密直接对程序的二进制可执行文件进行加密，这样是给加密带来很大的方便，因为在程序设计中，可以不用考虑加密代码了。但是由于外壳是一个相对比较标准的技术，因为它是标准的技术，就有标准的解密方法。所以就有成系列的脱壳工具，这种工具可以很快脱掉只进行外壳加密软件的壳。

加密锁公司提供的外壳工具，是为了和加密锁API配合使用的，在程序用加密锁API加密后，再用外壳加密程序打壳，就会隐藏程序中对加密锁API的调用，使解密者不易分析，从而达到更好的加密效果。

所以单纯使用外壳加密，就不能起到有效保护软件的效果。建议加密锁用户把加密锁API和加密锁外壳配合使用。

## 规律IO算法

软件开发商在使用加密锁时，时常会有与加密锁中算法进行交互的情况发生。给加密锁算法一定的输入，就得到相应的输出，看起来算法不在主机中出现，程序非常安全。但是如果算法输入输出情况

不多，一定的输入就有一定的输出，就容易被找出该算法的规律，从而被简单模拟，破解软件。

### 算法不良使用

这种不良加密方案是指用户在加密锁构造的算法，同时也在程序空间内出现。常表现为用加密锁中的算法加密，然后再用PC机上的算法解密，但使用的是对称算法，只要破解者明白了这个流程，就很容易破解。

由于ROCKEY6 SMART系列产品是可编程的加密锁，因此其开发模式需要两部分，一部分是加密锁内的程序，另一部分是程序与加密锁通讯的程序。在安装完软硬件后，就可以按照下面的步骤进行一个简单的保护程序开发了。

### Step1: 选择程序中的核心算法

ROCKEY6 SMART加密锁主要采用C语言进行开发，对于熟悉C语言的开发人员只需要学习一下ROCKEY6 SMART提供的系统调用函数就可以进行加密锁相关的开发。

加密锁开发流程：

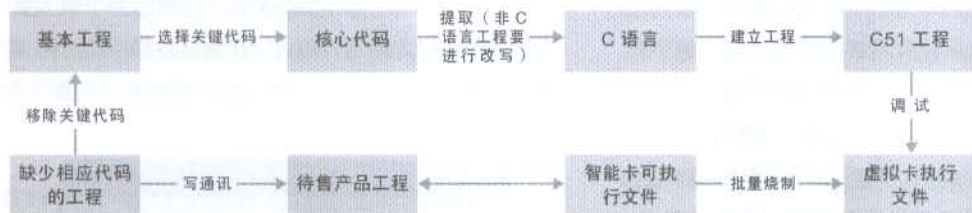


图 3-1 开发流程

考虑如下算法：输入15个字节身份证号码的字符数组，输出为18个字节的升位号码，这一个简单的身份证升位运算，假设这是个程序的重要组成部分，我们就可以将这段代码放入到加密锁中进行保护。C语言的源程序如下：

```
1 unsigned char Wi[18] = {7,9,10,5,8,4,2,
2 1,6,3,7,9,10,5,8,4,2,1};
3 char Ai[11]={'1','0','x','9','8','7','6',
4 '5','4','3','2'};
```

```
5 void ConvertID(char ID[15],char newID[18])
6 {
7     int i,j,s;
8     s=0;
9     memcpy(newID,ID,6);
10    newID[6]='1';
11    newID[7]='9';
12    memcpy(newID+8,ID+6,9);
13    for(i=0;i<17;i++)
14    {
15        j=(newID[i]-48)*Wi[i];
16        s+=j;
17    }
18    s%=11;
19    newID[17]=Ai[s];
20 }
```

下面我们就把这段程序转换成C51中的C语言。转换后成的C语言如下所示：

```
1. unsigned char Wi[18] = {7,9,10,5,8,4,2,
2 1,6,3,7,9,10,5,8,4,2,1};
3 char Ai
4 [11]={'1','0',
5 'x','9','8',
6 '7','6','5',
7 '4','3','2'};
8 void
9 main(void)
10 {
11     int i,j,s;
12     byte ID[15],newID[18];
13     s=0;
14     get_input(ID,0,0,15); // 输入原身份证号码
15     memcpy(newID,ID,6);
16     newID[6]='1';
17     newID[7]='9';
18     memcpy(newID+8,ID+6,9);
```



文件，运行此文件，可以根据程序的输入数据进行输入，运行完毕后返回程序的输出结果。

文件(目录)名	ID	类别	大小	序号	属性	安全
	2F01	FF	15	00	系统文件	0
2000	2000	FF	122	00	可执行	0

图 3-4-3 真实锁中的文件

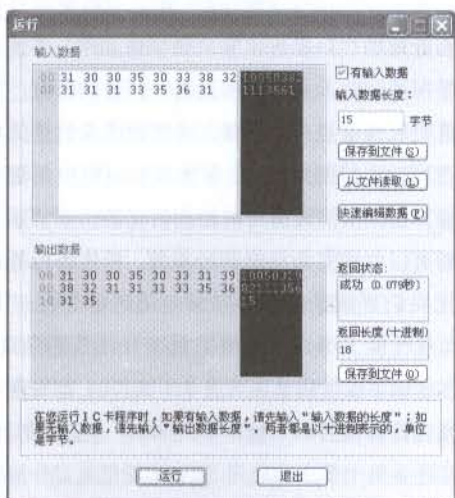


图 3-4-5 仿真运行

这时候点击“浏览真实设备”工具条按钮，可以看见该文件，这样，核心代码的移植工作就算完成了。剩下的工作就是在应用程序中写一个通讯模块来和锁进行通讯以完成保护的工作。

#### Step4: 建立主程序和算法文件的连接

运行一个加密锁文件的步骤:



图 3-5 运行文件步骤

下面是应用程序与ROCKEY6 SMART通讯和设置的API:

```

EXTERN_C int WINAPI DIC_Find(DWORD UID);

```

该函数根据用户号查找当前连接到计算机的加密锁，并返回数量。

```

EXTERN_C int WINAPI DIC_Open(int hic, char*
reader_name);

```

根据一个索引打开一个加密锁，并返回加密锁的句柄。

```

EXTERN_C int WINAPI DIC_Command(int hic,
int cmd, void* cmddata);

```

向指定的加密锁中发送命令。第一个参数是一个打开的加密锁的句柄，第二个参数是命令宏，第三个参数是根据命令宏传入传出的结构，具体每个宏的用法及其对应的结构请参考《用户手册》。

```

EXTERN_C int WINAPI DIC_Set(void* xdata,
int p1, int p2, int p3, char* buffer);

```

帮助用户向 DIC\_Command 命令中的 data 中设置数据。

```

EXTERN_C int WINAPI DIC_Get(void* xdata,
int p1, int p2, char* buffer);

```

帮助用户从 DIC\_Command 命令中的 data 中提取数据的函数。

DIC\_Set和DIC\_Get这两个函数的用法在用户手册中已做了详细的说明，需要注意的是这两个函数并不对加密锁进行任何操作，而是设置一个缓冲区，这个缓冲区将传递给DIC\_Command的第3个参数中，也就是所需要的结构，对于C类支持结构体的语言也可以不使用这两个函数，直接按照DIC\_Command函数的定义，根据不同的宏填充不同的结构体即可。这两个函数的目的就是帮助用户去设置复杂的结构体数据，尤其是那些不支持结构体的语言，如VB等。

下面我们就把原来的 ConvertID 封装成和加密锁通讯的函数

```

1 void ConvertID(const
  char ID[15],char newID[18])
  2 {
  3     DICST_Before_Run_Data *bD=
    (DICST_Before_Run_Data *)new char[48];
  4     DICST_After_Run_Data *aD=
    (DICST_After_Run_Data *)bD;
  5     int count=DIC_Find();// 查找加密锁

```

```

6     int hic=0;
7     for(int i=0;i<count;i++)
8     {
9         if((hic=DIC_Open(i,NULL))>
=0)
10            break;
11    }
12    if(hic==count)
13        return;// 未找到卡
14    bD->RunID=0x2000;//设置可执行文件
名称
15    bD->ParaSize=15;// 输入缓冲区的长
度
16    memcpy(bD->Para,ID,15);// 输入缓冲
区
17    int ret=DIC_Command(hic,RUN,bD);/
/运行文件并等待返回数据
18    memcpy(newID,aD->Result,aD->
ResultSize);// 取回返回数据
19 }

```

通过上面的四个步骤，我们就完成了整个工程的保护工作，单独提炼出来的模块和原来的接口一样，不需要更改程序的其他部分就可以照常运行。

### 核心代码的选择

在以上的例子中，我们介绍了一个基本的软件加密保护方案，由于这个例子的目的是让用户对加密锁进行加密的一个简单演示，实例本身过于简单，建议用户不要直接套用或使用类似简单的实例作为自己程序的保护。据我们调查，有不少用户在使用加密锁保护软件的时候，只是简单查找打开锁或者仅执行简单的计算，而且这段计算代码在整个程序中可有可无。这种加密保护的软件对于破解人员来说，破解也就是几分钟的事。建议用户在保护自己产品的时候一定要多花一些时间来做一些保护工作。下面我们就来探讨一下选择核心代码的几个原则和方法。

1. 代码一定要在整个程序中举足轻重。首先这段代码被下载到加密锁中，破解者无法得到这段代码，

如果这段没有这段代码，程序的一部分功能就不能实现，所以即使破解者即使绕过了这段代码，得到的也会是一个残缺不全的程序。

2. 尽量选择与自己行业相关的专业代码。使用通用的算法，破解者很容易“猜”出源代码的功能然后自己实现这个代码来达到其破解的目的。各位朋友大多都是做某一特定行业软件的，所以可以把自己行业的某些处理的代码提炼出来放到加密锁中，破解者虽然水平很高，但不大可能精通这一行业的算法，因此破解这类软件更是难上加难。比如做图像处理的可以使用图形处理的部分代码，做游戏的可以使用部分人工智能算法，做机械的可以使用特定的力学算法，做数学的可以使用复杂的微积分等等。相信各行各业的朋友比我们更清楚自己的哪部分代码更专业。

3. 从性能上考虑，不要把加密代码放到瓶颈处。因为加密锁中的代码是在智能卡中执行的，而智能卡的CPU要比计算机的CPU慢许多，因此在一定程度上，加密锁往往是整个算法的瓶颈。例如不要把这部分放到游戏中人物移动、界面程序的ONMOUSEMOVE事件等等。尽可能将代码放到某一复杂操作上，即使时间稍微延长一点，也不会造成软件使用的不便。

本文通过ROCKEY6 SMART,我们讨论了使用加密锁进行软件保护的基本原理和方法以及如何轻松的使用ROCKEY6 SMART进行软件保护。讨论了如何把代码分离出来并转换成C语言，同时讨论了如何编写主程序和加密锁的通讯模块和如何选择关键代码。

为了避免用户的软件成了盗版的牺牲品，请务必多花些时间来做软件保护的工作。

