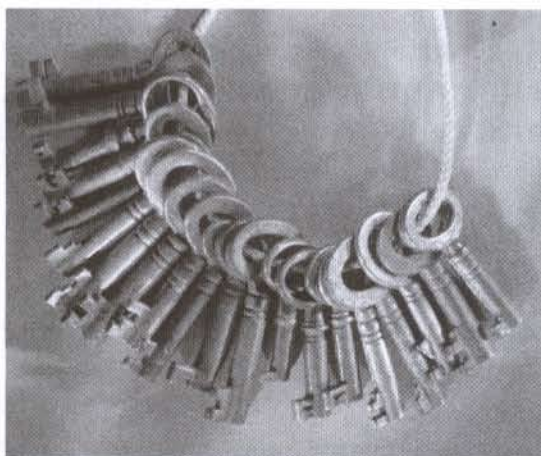


加密锁的使用技巧



在使用硬件加密时，加密的强度主要依靠两个方面：一是加密锁本身的设计结构，二是开发商使用加密锁的技巧。在使用同一种加密锁时，加密强度可能有很大的差别，因为很多开发商不是很熟悉加密锁的使用方式，没有充分利用加密锁提供的加密方式，给破解者留下了漏洞。在应用加密锁开发时加密的强度可以从两个途径提高：一是增加破解的工作量，二是增加加密点的复杂度。

一、增加工作量的基本技巧：

1、增加加密点的数量，在程序的多个位置，多次检验加密锁的存在，存取数据，破解者必须识别并修改每个加密点，只要有漏掉的加密点，程序就不能正常执行。

2、增加复杂的循环或死循环等耗时操作，让破解者难以跟踪程序的执行。例如：

```
RetCode = RY2_Find();
```

... .. // 加入多条语句，可以是与操作狗有关的，或者其他复杂的语句结构。后面的例子如无特别说明，此标记都代表这个意思

while(RetCode<=0)// 如果没有找到锁，则进入死循环

```
{
    int j = abs(random()/10);
    ... ..
```

```
RetCode = RecCode-j;
... ..
}
```

例 1.1

3、提高效率验和读写加密锁的随机性，使破解者难以理解程序的结构和执行规律。随机性是加密的重要思想，合理应用应用随机性还可以增加程序的复杂度，有效对抗分析软件和加密锁模拟器。例如

```
int j = abs(random()/100);
... ..
if(j%17==0)
{
    ... ..
    RetCode = RY2_Find();
    ... ..
    if(RetCode==0) {} // 未找到锁，
    出错处理
    else
    {
        RetCode = RY2_Open(...);//打
        开指定UID 的加密锁
        if(RetCode==0) {} // 打开失败，
        出错处理
    }
}
```

例 1.2

二、增加加密点的复杂度的基本方法是增加迷惑语句，并根据加密锁本身的特点使用一些编程技巧，迷惑就是增加破解者找到加密点的难度，使程序难于理解。常见的有以下几种方式：

1、不显示出错提示信息：这个技巧很重要，尽可能少地给用户提示信息，因为这些蛛丝马迹都可能导致解密者直接深入到保护的核心。比如，当

检测到破解企图之后，不要立即给用户提示信息，而是在系统的某个地方做一个记号，随机地过一段时间后使软件停止工作，或者装作正常工作但实际上却在所处理的数据中加入了一些垃圾。例如可以改写上面的示例中 `if(RetCode==0){}` // 未找到锁，出错处理 这句：

```
int flag = 1;
...// 此部分同例 1.2
    if(RetCode==0) // 未找到锁，出错处理
    {
        Flag=
    }
...// 此部分同例 1.2
if(flag==0)
{
    ... ..
    // 这里可以采用三种处理方法
    1. 普通的：未找到加密锁，退出程序
    2. 迷惑性处理：启动定时器，过一定的时间后结束程序
    3. 迷惑性处理：破坏掉锁内的数据，继续运行程序。
    ... ..
}
```

例 2.1

2、加入复杂的循环和死循环，让破解者难以跟踪，发现加密点。这个方法与例 1.1 类似：

```
for ( ;; )
{
    int j = random() %1000;
    if(j%917==0)
    {
        RetCode = RY2_Find();
        if(RetCode == 0)
        {
        }
        else
        {
        }
    }
}
```

```
}
}
```

例 2.2

3、随机的读写和效验加密锁，在随机的位置读写。这条技巧前面已经多次应用，下面举例说说随机位置的读和写：

```
int i = abs(random()/5);
... ..
char buffer[512];
... ..
retcode = RY2_Write(handle, i, buffer);//
/ 随机写入到某一块
... ..
retcode = RY2_Read(handle, i, buffer);//
读出刚才写入的那一块
```

例 2.3

4、把操作加密锁和最终判断程序执行是否合法的语句分开。这个技巧前面的例子里面都用到了，就是不要在得到操作加密锁的状态后立即判断状态是否正常，而是在操作和判断之间加入一定的代码。

5、加入无用的读写，效验锁的语句或其他无效代码。

6、在加密锁内写入读出无效数据。当然不要应用真的应用这些数据。2.4、2.5、2.6 常结合使用。

三、下面结合 rokey2 的特点说说综合的编程技巧。

1、使用加密锁传递变量，把加密锁内的存储区当作临时变量，变量需要赋值时，向锁内写入数据，需要应用变量的值时读取加密锁。

a. 传递局部或全局变量举例

```
void func()
{
    int i;// 这个变量的值将被写入加密锁，
        // 锁内值的更改和使用代替了变量 i
    本身
    ... ..
    char buffer[512];
    memcpy(buffer,&i,sizeof(i))
```

```

retcode = RY2_Write(handle, 0, buffer);
... ..
while(...)
{
... ..
if(...)
{
    int j
    retcode = RY2_Read(handle, 0, buffer);

    memcpy(&j,buffer,sizeof(i));
    ... ..//原来需要用变量i的现在改用
    临时变量j,j的值是从加密锁内读出来的

    retcode = RY2_Write(handle, 0,
buffer);
//原来把变量j的当前值写入加密锁,相当于变量i
的值发生了变化
}
}
}

```

例 3.1

b. 传递函数的参数:

普通的编程思路:

```

Func1(int i)
{}
Func2()
{
    int j;
    ...
    Func1(j)
}

```

改写为应用加密锁的方式:

```

Func1()
{
    int j;
    retcode = RY2_Read(handle, 0, buffer);
    memcpy(&j,buffer,sizeof(int));//这个过
程相当于参数传递
... ..

```

```

}
Func2()
{
    char buffer[512];
    int i;//这个是要传递的参数,
    ... ..
    memcpy(buffer,&i,sizeof(i))
    retcode = RY2_Write(handle, 0, buffer);
    ... ..
    Func1();
}

```

例 3.2

2、把需要保存在磁盘上的数据全部或部分保存到加密锁内。Rockey2 提供了多达 2560 字节的存储空间,可以把本来需要保存在磁盘上的重要的数据保存在加密锁内,比如程序的配置信息,上次运行时得到的下次需要应用的数据等等

3、使用多线程技术。利用多线程调试的复杂性可以有效迷惑破解者和破解软件。可在一个线程中寻找加密锁,随机读写数据,在另外的线程中应用这些信息。多线程技术也是加密锁应用的重要技巧。

