1、RSA算法

它是第一个既能用于数据加密也能用于数字签名的算法。它易于理解和操作，也很流行。算法的名字以发明者的名字命名：Ron Rivest, Adi Shamir 和 Leonard Adleman。但 RSA 的安全性一直未能得到理论上的证明。它经历了各种攻击，至今未被完全攻破。

一、RSA算法 :

首先, 找出三个数, p, q, r,
其中 p, q 是两个相异的质数, r 是与 (p-1)(q-1) 互质的数.....
p, q, r 这三个数便是 private key

接著, 找出 m, 使得 rm == 1 mod (p-1)(q-1).....
这个 m 一定存在, 因为 r 与 (p-1)(q-1) 互质, 用辗转相除法就可以得到了.....
再来, 计算 n = pq.....
m, n 这两个数便是 public key

编码过程是, 若资料为 a, 将其看成是一个大整数, 假设 a < n....
如果 a >= n 的话, 就将 a 表成 s 进位 (s <= n, 通常取 s = 2^t),
则每一位数均小於 n, 然後分段编码.....
接下来, 计算 b == a^m mod n, (0 <= b < n),
b 就是编码後的资料.....

解码的过程是, 计算 c == b^r mod pq (0 <= c < pq),
於是乎, 解码完毕..... 等会会证明 c 和 a 其实是相等的 :)

如果第三者进行窃听时, 他会得到几个数: m, n(=pq), b.....
他如果要解码的话, 必须想办法得到 r.....
所以, 他必须先对 n 作质因数分解.....
要防止他分解, 最有效的方法是找两个非常的大质数 p, q,
使第三者作因数分解时发生困难.....

<定理>
若 p, q 是相异质数, rm == 1 mod (p-1)(q-1),
a 是任意一个正整数, b == a^m mod pq, c == b^r mod pq,
则 c == a mod pq

证明的过程, 会用到费马小定理, 叙述如下:
m 是任一质数, n 是任一整数, 则 n^m == n mod m
(换另一句话说, 如果 n 和 m 互质, 则 n^(m-1) == 1 mod m)
运用一些基本的群论的知识, 就可以很容易地证出费马小定理的.....

<证明>
因为 rm == 1 mod (p-1)(q-1), 所以 rm = k(p-1)(q-1) + 1, 其中 k 是整数

因为在 modulo 中是 preserve 乘法的
(x == y mod z and u == v mod z => xu == yv mod z),
所以, c == b^r == (a^m)^r == a^(rm) == a^(k(p-1)(q-1)+1) mod pq

1. 如果 a 不是 p 的倍数, 也不是 q 的倍数时,
  则 a^(p-1) == 1 mod p (费马小定理) => a^(k(p-1)(q-1)) == 1 mod p
    a^(q-1) == 1 mod q (费马小定理) => a^(k(p-1)(q-1)) == 1 mod q
  所以 p, q 均能整除 a^(k(p-1)(q-1)) - 1 => pq | a^(k(p-1)(q-1)) - 1
  即 a^(k(p-1)(q-1)) == 1 mod pq
  => c == a^(k(p-1)(q-1)+1) == a mod pq

2. 如果 a 是 p 的倍数, 但不是 q 的倍数时,
  则 a^(q-1) == 1 mod q (费马小定理)
  => a^(k(p-1)(q-1)) == 1 mod q
  => c == a^(k(p-1)(q-1)+1) == a mod q
  => q | c - a
  因 p | a
  => c == a^(k(p-1)(q-1)+1) == 0 mod p
  => p | c - a
  所以, pq | c - a => c == a mod pq

3. 如果 a 是 q 的倍数, 但不是 p 的倍数时, 证明同上

4. 如果 a 同时是 p 和 q 的倍数时,
  则 pq | a
  => c == a^(k(p-1)(q-1)+1) == 0 mod pq
  => pq | c - a
  => c == a mod pq
                Q.E.D.

这个定理说明 a 经过编码为 b 再经过解码为 c 时, a == c mod n (n = pq)....
但我们在做编码解码时, 限制 0 <= a < n, 0 <= c < n,
所以这就是说 a 等於 c, 所以这个过程确实能做到编码解码的功能.....

二、RSA 的安全性

RSA 的安全性依赖于大数分解，但是否等同于大数分解一直未能得到理论上的证明，因为没有证明破解 RSA 就一定需要作大数分解。假设存在一种无须分解大数的算法，那它肯定可以修改成为大数分解算法。目前， RSA 的一些变种算法已被证明等价于大数分解。不管怎样，分解 n 是最显然的攻击方法。现在，人们已能分解多个十进制位的大素数。因此，模数 n 必须选大一些，因具体适用情况而定。

三、RSA的速度

由于进行的都是大数计算，使得RSA最快的情况也比DES慢上倍，无论是软件还是硬件实现。速度一直是RSA的缺陷。一般来说只用于少量数据加密。

四、RSA的选择密文攻击

RSA在选择密文攻击面前很脆弱。一般攻击者是将某一信息作一下伪装(Blind)，让拥有私钥的实体签署。然后，经过计算就可得到它所想要的信息。实际上，攻击利用的都是同一个弱点，即存在这样一个事实：乘幂保留了输入的乘法结构：

(XM)^d = X^d *M^d mod n

前面已经提到，这个固有的问题来自于公钥密码系统的最有用的特征--每个人都能使用公钥。但从算法上无法解决这一问题，主要措施有两条：一条是采用好的公钥协议，保证工作过程中实体不对其他实体任意产生的信息解密，不对自己一无所知的信息签名；另一条是决不对陌生人送来的随机文档签名，签名时首先使用One-Way HashFunction 对文档作HASH处理，或同时使用不同的签名算法。在中提到了几种不同类型的攻击方法。

五、RSA 的公共模数攻击

若系统中共有一个模数，只是不同的人拥有不同的e和d，系统将是危险的。最普遍的情况是同一信息用不同的公钥加密，这些公钥共模而且互质，那末该信息无需私钥就可得到恢复。设P为信息明文，两个加密密钥为e1 和e2，公共模数是n，则：

C1 = P^e1 mod n

C2 = P^e2 mod n

密码分析者知道n、e1、e2、C1 和C2，就能得到P。

因为e1 和e2 互质，故用Euclidean算法能找到r和s，满足：

r * e1 + s * e2 = 1

假设r为负数，需再用Euclidean算法计算C1^(-1)，则

(C1^(-1))^(-r) * C2^s = P mod n

另外，还有其它几种利用公共模数攻击的方法。总之，如果知道给定模数的一对e和d，一是有利于攻击者分解模数，一是有利于攻击者计算出其它成对的e’和d’，而无需分解模数。解决办法只有一个，那就是不要共享模数n。

RSA的小指数攻击。 有一种提高 RSA速度的建议是使公钥e取较小的值，这样会使加密变得易于实现，速度有
所提高。但这样作是不安全的，对付办法就是e和d都取较大的值。

RSA算法是第一个能同时用于加密和数字签名的算法，也易于理解和操作。RSA是被研究得最广泛的公钥算法，从提出到现在已近二十年，经历了各种攻击的考验，逐渐为人们接受，普遍认为是目前最优秀的公钥方案之一。RSA的安全性依赖于大数的因子分解，但并没有从理论上证明破译RSA的难度与大数分解难度等价。即RSA的重大缺陷是无法从理论上把握它的保密性能如何，而且密码学界多数人士倾向于因子分解不是NPC问题。 RSA的缺点主要有：A)产生密钥很麻烦，受到素数产生技术的限制，因而难以做到一次一密。B)分组长度太大，为保证安全性，n 至少也要 600 bits 以上，使运算代价很高，尤其是速度较慢，较对称密码算法慢几个数量级；且随着大数分解技术的发展，这个长度还在增加，不利于数据格式的标准化。目前，SET(Secure Electronic Transaction)协议中要求CA采用比特长的密钥，其他实体使用比特的密钥。

2、DES 算法

一、DES 算法

美国国家标准局 1973 年开始研究除国防部外的其它部门的计算机系统的数据加密标准，于 1973 年 5 月 15 日和 1974 年 8 月 27 日先后两次向公众发出了征求加密算法的公告。加密算法要达到的目的（通常称为 DES 密码算法要求）主要为以下四点：

☆提供高质量的数据保护，防止数据未经授权的泄露和未被察觉的修改；

☆具有相当高的复杂性，使得破译的开销超过可能获得的利益，同时又要便于理解和掌握；

☆DES 密码体制的安全性应该不依赖于算法的保密，其安全性仅以加密密钥的保密为基础；

☆实现经济，运行有效，并且适用于多种完全不同的应用。

1977 年 1 月，美国政府颁布：采纳 IBM 公司设计的方案作为非机密数据的正式数据加密标准（DES 棗 Data Encryption Standard）。

目前在国内，随着三金工程尤其是金卡工程的启动，DES 算法在 POS、ATM、磁卡及智能卡（IC 卡）、加油站、高速公路收费站等领域被广泛应用，以此来实现关键数据的保密，如信用卡持卡人的 PIN 的加密传输，IC 卡与 POS 间的双向认证、金融交易数据包的 MAC 校验等，均用到 DES 算法。

DES 算法的入口参数有三个：Key、Data、Mode。其中 Key 为 8 个字节共 64 位，是 DES 算法的工作密钥；Data 也为 8 个字节 64 位，是要被加密或被解密的数据；Mode 为 DES 的工作方式，有两种：加密或解密。

DES 算法是这样工作的：如 Mode 为加密，则用 Key 去把数据 Data 进行加密， 生成 Data 的密码形式（64 位）作为 DES 的输出结果；如 Mode 为解密，则用 Key 去把密码形式的数据 Data 解密，还原为 Data 的明码形式（64 位）作为 DES 的输出结果。在通信网络的两端，双方约定一致的 Key，在通信的源点用 Key 对核心数据进行 DES 加密，然后以密码形式在公共通信网（如电话网）中传输到通信网络的终点，数据到达目的地后，用同样的 Key 对密码数据进行解密，便再现了明码形式的核心数据。这样，便保证了核心数据（如 PIN、MAC 等）在公共通信网中传输的安全性和可靠性。

通过定期在通信网络的源端和目的端同时改用新的 Key，便能更进一步提高数据的保密性，这正是现在金融交易网络的流行做法。

DES 算法详述

DES 算法把 64 位的明文输入块变为 64 位的密文输出块，它所使用的密钥也是 64 位，整个算法的主流程图如下：

其功能是把输入的 64 位数据块按位重新组合，并把输出分为 L0、R0 两部分，每部分各长 32 位，其置换规则见下表：

58,50,12,34,26,18,10,2,60,52,44,36,28,20,12,4,

62,54,46,38,30,22,14,6,64,56,48,40,32,24,16,8,

57,49,41,33,25,17, 9,1,59,51,43,35,27,19,11,3,

61,53,45,37,29,21,13,5,63,55,47,39,31,23,15,7,

即将输入的第 58 位换到第一位，第 50 位换到第 2 位，...，依此类推，最后一位是原来的第 7 位。L0、R0 则是换位输出后的两部分，L0 是输出的左 32 位，R0 是右 32 位，例：设置换前的输入值为 D1D2D3.....D64，则经过初始置换后的结果为：L0=D58D50...D8；R0=D57D49...D7。

经过 16 次迭代运算后。得到 L16、R16，将此作为输入，进行逆置换，即得到密文输出。逆置换正好是初始置的逆运算，例如，第 1 位经过初始置换后，处于第 40 位，而通过逆置换，又将第 40 位换回到第 1 位，其逆置换规则如下表所示：

40,8,48,16,56,24,64,32,39,7,47,15,55,23,63,31,

38,6,46,14,54,22,62,30,37,5,45,13,53,21,61,29,

36,4,44,12,52,20,60,28,35,3,43,11,51,19,59,27,

34,2,42,10,50,18,58 26,33,1,41, 9,49,17,57,25,

放大换位表

32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9, 8, 9, 10,11,

12,13,12,13,14,15,16,17,16,17,18,19,20,21,20,21,

22,23,24,25,24,25,26,27,28,29,28,29,30,31,32, 1,

单纯换位表

16,7,20,21,29,12,28,17, 1,15,23,26, 5,18,31,10,

2,8,24,14,32,27, 3, 9,19,13,30, 6,22,11, 4,25,

在 f(Ri,Ki)算法描述图中，S1,S2...S8 为选择函数，其功能是把 6bit 数据变为 4bit 数据。下面给出选择函数 Si(i=1,2.....8)的功能表：

选择函数 Si

S1:

14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7,

0,15,7,4,14,2,13,1,10,6,12,11,9,5,3,8,

4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0,

15,12,8,2,4,9,1,7,5,11,3,14,10,0,6,13,

S2:

15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10,

3,13,4,7,15,2,8,14,12,0,1,10,6,9,11,5,

0,14,7,11,10,4,13,1,5,8,12,6,9,3,2,15,

13,8,10,1,3,15,4,2,11,6,7,12,0,5,14,9,

S3:

10,0,9,14,6,3,15,5,1,13,12,7,11,4,2,8,

13,7,0,9,3,4,6,10,2,8,5,14,12,11,15,1,

13,6,4,9,8,15,3,0,11,1,2,12,5,10,14,7,

1,10,13,0,6,9,8,7,4,15,14,3,11,5,2,12,

S4:

7,13,14,3,0,6,9,10,1,2,8,5,11,12,4,15,

13,8,11,5,6,15,0,3,4,7,2,12,1,10,14,9,

10,6,9,0,12,11,7,13,15,1,3,14,5,2,8,4,

3,15,0,6,10,1,13,8,9,4,5,11,12,7,2,14,

S5:

2,12,4,1,7,10,11,6,8,5,3,15,13,0,14,9,

14,11,2,12,4,7,13,1,5,0,15,10,3,9,8,6,

4,2,1,11,10,13,7,8,15,9,12,5,6,3,0,14,

11,8,12,7,1,14,2,13,6,15,0,9,10,4,5,3,

S6:

12,1,10,15,9,2,6,8,0,13,3,4,14,7,5,11,

10,15,4,2,7,12,9,5,6,1,13,14,0,11,3,8,

9,14,15,5,2,8,12,3,7,0,4,10,1,13,11,6,

4,3,2,12,9,5,15,10,11,14,1,7,6,0,8,13,

S7:

4,11,2,14,15,0,8,13,3,12,9,7,5,10,6,1,

13,0,11,7,4,9,1,10,14,3,5,12,2,15,8,6,

1,4,11,13,12,3,7,14,10,15,6,8,0,5,9,2,

6,11,13,8,1,4,10,7,9,5,0,15,14,2,3,12,

S8:

13,2,8,4,6,15,11,1,10,9,3,14,5,0,12,7,

1,15,13,8,10,3,7,4,12,5,6,11,0,14,9,2,

7,11,4,1,9,12,14,2,0,6,10,13,15,3,5,8,

2,1,14,7,4,10,8,13,15,12,9,0,3,5,6,11,

在此以 S1 为例说明其功能，我们可以看到：在 S1 中，共有 4 行数据，命名为 0，1、2、3 行；每行有 16 列，命名为 0、1、2、3，.....，14、15 列。

现设输入为： D＝D1D2D3D4D5D6

令：列＝D2D3D4D5

行＝D1D6

然后在 S1 表中查得对应的数，以 4 位二进制表示，此即为选择函数 S1 的输出。下面给出子密钥 Ki(48bit) 的生成算法

从子密钥 Ki 的生成算法描述图中我们可以看到：初始 Key 值为 64 位，但 DES 算法规定，其中第 8、16、.....64 位是奇偶校验位，不参与 DES 运算。故 Key 实际可用位数便只有 56 位。即：经过缩小选择换位表 1 的变换后，Key 的位数由 64 位变成了 56 位，此 56 位分为 C0、D0 两部分，各 28 位，然后分别进行第 1 次循环左移，得到 C1、D1，将 C1（28 位）、D1（28 位）合并得到 56 位，再经过缩小选择换位 2，从而便得到了密钥 K0（48 位）。依此类推，便可得到 K1、K2、.....、K15，不过需要注意的是，16 次循环左移对应的左移位数要依据下述规则进行：

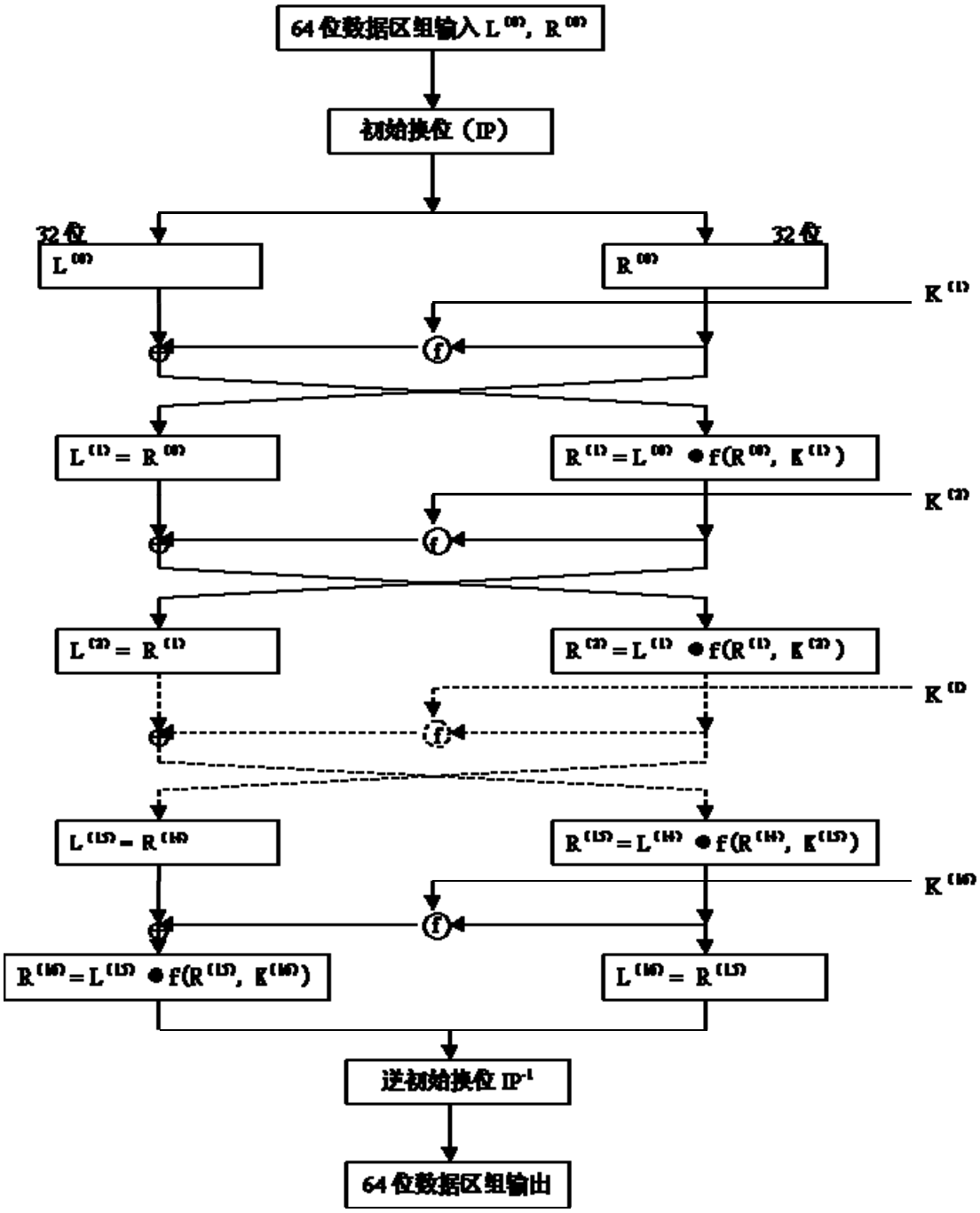
循环左移位数

1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1

以上介绍了 DES 算法的加密过程。DES 算法的解密过程是一样的，区别仅仅在于第一次迭代时用子密钥 K15，第二次 K14、.....，最后一次用 K0，算法本身并没有任何变化。

二、DES 算法理论图解

DES 的算法是对称的，既可用于加密又可用于解密。下图是它的算法粗框图。其具体运算过程有如下七步。



三、DES 算法的应用误区

DES 算法具有极高安全性，到目前为止，除了用穷举搜索法对 DES 算法进行攻击外，还没有发现更有效的办法。而 56 位长的密钥的穷举空间为 256，这意味着如果一台计算机的速度是每一秒种检测一百万个密钥，则它搜索完全部密钥就需要将近 2285 年的时间，可见，这是难以实现的，当然，随着科学技术的发展，当出现超高速计算机后，我们可考虑把 DES 密钥的长度再增长一些，以此来达到更高的保密程度。

由上述 DES 算法介绍我们可以看到：DES 算法中只用到 64 位密钥中的其中 56 位，而第 8、16、24、.....64 位 8 个位并未参与 DES 运算，这一点，向我们提出了一个应用上的要求，即 DES 的安全性是基于除了 8，16，24，.....64 位外的其余 56 位的组合变化 256 才得以保证的。因此，在实际应用中，我们应避开使用第 8，16，24，.....64 位作为有效数据位，而使用其它的 56 位作为有效数据位，才能保证 DES 算法安全可靠

地发挥作用。如果不了解这一点，把密钥 Key 的 8，16，24，..... .64 位作为有效数据使用，将不能保证 DES 加密数据的安全性，对运用 DES 来达到保密作用的系统产生数据被破译的危险，这正是 DES 算法在应用上的误区，留下了被人攻击、被人破译的极大隐患。

3、ElGamal算法

ElGamal算法既能用于数据加密也能用于数字签名，其安全性依赖于计算有限域上离散对数这一难题。密钥对产生办法。首先选择一个素数p，两个随机数, g 和x，g, x < p, 计算 y = g^x (mod p)，则其公钥为 y, g 和p。私钥是x。g和p可由一组用户共享。
ElGamal用于数字签名。被签信息为M，首先选择一个随机数k, k与 p - 1 互质，计算

a = g^k (mod p)
再用扩展 Euclidean 算法对下面方程求解b：

M = xa + kb (mod p - 1)

签名就是(a, b)。随机数k须丢弃。
验证时要验证下式：

y^a * a^b (mod p) = g^M (mod p)

同时一定要检验是否满足 1<= a < p。否则签名容易伪造。
ElGamal用于加密。被加密信息为M，首先选择一个随机数k，k与 p - 1 互质，计算

a = g^k (mod p)
b = y^k M (mod p)

(a, b)为密文，是明文的两倍长。解密时计算

M = b / a^x (mod p)

ElGamal签名的安全性依赖于乘法群(IFp)* 上的离散对数计算。素数p必须足够大，且p-1 至少包含一个大素数
因子以抵抗Pohlig & Hellman算法的攻击。M一般都应采用信息的HASH值(如SHA算法)。ElGamal的安全性主要依赖于p和g，若选取不当则签名容易伪造，应保证g对于p-1 的大素数因子不可约。
D.Bleichenbache“GeneratingElGamal Signatures Without Knowing the Secret Key”中提到了一些攻击方法和对策。ElGamal的一个不足之处是它的密文成倍扩张。

美国的DSS(Digital Signature Standard)的DSA(Digital Signature Algorithm)算法是经ElGamal算法演变而来。

4、DSA算法

Digital Signature Algorithm (DSA)是Schnorr和ElGamal签名算法的变种，被美国NIST作为DSS(DigitalSignature Standard)。算法中应用了下述参数：

p：L bits长的素数。L是 64 的倍数，范围是 512 到 1024；
q：p - 1 的 160bits的素因子；
g：g = h^((p-1)/q) mod p，h满足h < p - 1, h^((p-1)/q) mod p > 1；
x：x < q，x为私钥 ；
y：y = g^x mod p ，(p, q, g, y)为公钥；
H(x)：One-Way Hash函数。DSS中选用SHA(Secure Hash Algorithm)。
p, q, g可由一组用户共享，但在实际应用中，使用公共模数可能会带来一定的威胁。签名及验证协议如下：

1. P产生随机数k，k < q；
2. P计算 r = (g^k mod p) mod q
s = (k^(-1) (H(m) + xr)) mod q
签名结果是(m, r, s)。
3. 验证时计算 w = s^(-1)mod q
u1 = (H(m) * w) mod q
u2 = (r * w) mod q
v = ((g^u1 * y^u2) mod p) mod q
若v = r，则认为签名有效。

DSA是基于整数有限域离散对数难题的，其安全性与RSA相比差不多。DSA的一个重要特点是两个素数公开，这
样，当使用别人的p和q时，即使不知道私钥，你也能确认它们是否是随机产生的，还是作了手脚。RSA算法却作不到。

5、MD5 算法

在一些初始化处理后，MD5 以 512 位分组来处理输入文本，每一分组又划分为 16 个 32 位子分组。算法的输出由四个 32 位分组组成，将它们级联形成一个 128 位散列值。
首先填充消息使其长度恰好为一个比 512 位的倍数仅小 64 位的数。填充方法是附一个 1 在消息后面，后接所要求的多个 0，然后在其后附上 64 位的消息长度（填充前）。这两步的作用是使消息长度恰好是 512 位的整数倍（算法的其余部分要求如此），同时确保不同的消息在填充后不相同。
四个 32 位变量初始化为：
A=0x01234567
B=0x89abcdef
C=0xfedcba98
D=0x76543210
它们称为链接变量（chaining variable）
接着进行算法的主循环，循环的次数是消息中 512 位消息分组的数目。
将上面四个变量复制到别外的变量中：A 到 a，B 到 b，C 到 c，D 到 d。
主循环有四轮（MD4 只有三轮），每轮很相拟。第一轮进行 16 次操作。每次操作对 a，b，c 和 d 中的其

中三个作一次非线性函数运算，然后将所得结果加上第四个变量，文本的一个子分组和一个常数。再将所得结果向右环移一个不定的数，并加上 a，b，c 或 d 中之一。最后用该结果取代 a，b，c 或 d 中之一。

以一下是每次操作中用到的四个非线性函数（每轮一个）。

F(X,Y,Z)=(X&Y)|((~X)&Z)

G(X,Y,Z)=(X&Z)|(Y&(~Z))

H(X,Y,Z)=X^Y^Z

I(X,Y,Z)=Y^(X|(~Z))

(&是与,|是或,~是非,^是异或)

这些函数是这样设计的：如果 X、Y 和 Z 的对应位是独立和均匀的，那么结果的每一位也应是独立和均匀的。

函数 F 是按逐位方式操作：如果 X，那么 Y，否则 Z。函数 H 是逐位奇偶操作符。

设 Mj 表示消息的第 j 个子分组（从 0 到 15），<<<s 表示循环左移 s 位，则四种操作为：

FF(a,b,c,d,Mj,s,ti)表示 a=b+((a+(F(b,c,d)+Mj+ti)<<<s)

GG(a,b,c,d,Mj,s,ti)表示 a=b+((a+(G(b,c,d)+Mj+ti)<<<s)

HH(a,b,c,d,Mj,s,ti)表示 a=b+((a+(H(b,c,d)+Mj+ti)<<<s)

II(a,b,c,d,Mj,s,ti)表示 a=b+((a+(I(b,c,d)+Mj+ti)<<<s)

这四轮（64 步）是：

第一轮

FF(a,b,c,d,M0,7,0xd76aa478)

FF(d,a,b,c,M1,12,0xe8c7b756)

FF(c,d,a,b,M2,17,0x242070db)

FF(b,c,d,a,M3,22,0xc1bdceee)

FF(a,b,c,d,M4,7,0xf57c0faf)

FF(d,a,b,c,M5,12,0x4787c62a)

FF(c,d,a,b,M6,17,0xa8304613)

FF(b,c,d,a,M7,22,0xfd469501)

FF(a,b,c,d,M8,7,0x698098d8)

FF(d,a,b,c,M9,12,0x8b44f7af)

FF(c,d,a,b,M10,17,0xffff5bb1)

FF(b,c,d,a,M11,22,0x895cd7be)

FF(a,b,c,d,M12,7,0x6b901122)

FF(d,a,b,c,M13,12,0xfd987193)

FF(c,d,a,b,M14,17,0xa679438e)

FF(b,c,d,a,M15,22,0x49b40821)

第二轮

GG(a,b,c,d,M1,5,0xf61e2562)

GG(d,a,b,c,M6,9,0xc040b340)

GG(c,d,a,b,M11,14,0x265e5a51)

GG(b,c,d,a,M0,20,0xe9b6c7aa)

GG(a,b,c,d,M5,5,0xd62f105d)

GG(d,a,b,c,M10,9,0x02441453)

GG(c,d,a,b,M15,14,0xd8a1e681)

GG(b,c,d,a,M4,20,0xe7d3fbc8)

GG(a,b,c,d,M9,5,0x21e1cde6)

GG(d,a,b,c,M14,9,0xc33707d6)

GG(c,d,a,b,M3,14,0xf4d50d87)

GG(b,c,d,a,M8,20,0x455a14ed)

GG(a,b,c,d,M13,5,0xa9e3e905)

GG(d,a,b,c,M2,9,0xfcefa3f8)

GG(c,d,a,b,M7,14,0x676f02d9)

GG(b,c,d,a,M12,20,0x8d2a4c8a)

第三轮

HH(a,b,c,d,M5,4,0xfffa3942)

HH(d,a,b,c,M8,11,0x8771f681)

HH(c,d,a,b,M11,16,0x6d9d6122)

HH(b,c,d,a,M14,23,0xfde5380c)

HH(a,b,c,d,M1,4,0xa4beea44)

HH(d,a,b,c,M4,11,0x4bdecfa9)

HH(c,d,a,b,M7,16,0xf6bb4b60)

HH(b,c,d,a,M10,23,0xbebfbc70)

HH(a,b,c,d,M13,4,0x289b7ec6)

HH(d,a,b,c,M0,11,0xeaa127fa)

HH(c,d,a,b,M3,16,0xd4ef3085)

HH(b,c,d,a,M6,23,0x04881d05)

HH(a,b,c,d,M9,4,0xd9d4d039)

HH(d,a,b,c,M12,11,0xe6db99e5)

HH(c,d,a,b,M15,16,0x1fa27cf8)

HH(b,c,d,a,M2,23,0xc4ac5665)

第四轮

II(a,b,c,d,M0,6,0xf4292244)

II(d,a,b,c,M7,10,0x432aff97)

II(c,d,a,b,M14,15,0xab9423a7)

II(b,c,d,a,M5,21,0xfc93a039)

II(a,b,c,d,M12,6,0x655b59c3)

II(d,a,b,c,M3,10,0x8f0ccc92)

II(c,d,a,b,M10,15,0xffeff47d)

II(b,c,d,a,M1,21,0x85845dd1)

II(a,b,c,d,M8,6,0x6fa87e4f)

II(d,a,b,c,M15,10,0xfe2ce6e0)

II(c,d,a,b,M6,15,0xa3014314)

II(b,c,d,a,M13,21,0x4e0811a1)

II(a,b,c,d,M4,6,0xf7537e82)

II(d,a,b,c,M11,10,0xbd3af235)

II(c,d,a,b,M2,15,0x2ad7d2bb)

II(b,c,d,a,M9,21,0xeb86d391)

常数 ti 可以如下选择：

在第 i 步中，ti 是 4294967296*abs(sin(i))的整数部分,i 的单位是弧度。

(2 的 32 次方)

所有这些完成之后，将 A，B，C，D 分别加上 a，b，c，d。然后用下一分组数据继续运行算法，最后的输出是 A，B，C 和 D 的级联。

MD5 的安全性

MD5 相对 MD4 所作的改进：
1.增加了第四轮.
2.每一步均有唯一的加法常数.
3.为减弱第二轮中函数 G 的对称性从(X&Y)|(X&Z)|(Y&Z)变为(X&Z)|(Y&(~Z))
4.第一步加上了上一步的结果,这将引起更快的雪崩效应.
5.改变了第二轮和第三轮中访问消息子分组的次序,使其更不相似.
6.近似优化了每一轮中的循环左移位移量以实现更快的雪崩效应.各轮的位移量互不相同.

6、BLOWFISH 算法

作　者：夜月
联　系：luoyi_ly1@sina.com
时　间：2001 年 10 月 6 日
范　例：BlowFish's CrackMe1
注册机：Bfkeygen

一、BlowFish 算法说明（文中数据类型以 Tc2.0 为准）

BlowFish 算法用来加密 64Bit 长度的字符串。

BlowFish 算法使用两个“盒”——ungigned long pbox[18]和 unsigned long sbox[4,256]。

BlowFish 算法中，有一个核心加密函数:BF_En(后文详细介绍）。该函数输入 64 位信息，运算后， 以 64 位密文的形式输出。 用 BlowFish 算法加密信息，需要两个过程：

1.密钥预处理
2.信息加密

分别说明如下：
密钥预处理：

BlowFish 算法的源密钥——pbox 和 sbox 是固定的。我们要加密一个信息，需要自己选择一个 key， 用这个 key 对 pbox 和 sbox 进行变换，得到下一步信息加密所要用的 key_pbox 和 key_sbox。具体的变化算法如下：

1)用 sbox 填充 key_sbox
2)用自己选择的 key8 个一组地去异或 pbox，用异或的结果填充 key_pbox。key 可以循环使用。
比如说：选的 key 是"abcdefghijklmn"。则异或过程为：
key_pbox[0]=pbox[0]^abcdefgh

key_pbox[1]=pbox[1]^ijklmnab

……

……

如此循环，直到 key_box 填充完毕。

3)用 BF_En 加密一个全 0 的 64 位信息，用输出的结果替换 key_pbox[0]和 key_pbox[1]。i=0

4)用 BF_En 加密替换后的 key_pbox[i],key_pbox[i+1],用输出替代 key_pbox[i+2]和 key_pbox[i+3]

5)i+2，继续第 4 步，直到 key_pbox 全部被替换

6)用 key_pbox[16]和 key_pbox[17]做首次输入（相当于上面的全 0 的输入），用类似的方法，替换 key_sbox 信息加密。信息加密就是用函数把待加密信息 x 分成 32 位的两部分:xL,xR BF_En 对输入信息进行变换，BF_En 函数详细过程如下：

对于 i=1 至 16

xL=xL^Pi

xR=F(xL)^xR

交换 xL 和 xR(最后一轮取消该运算)

xR=xR^P17

xL=xL^P18

重新合并 xL 和 xR

函数 F 见下图:

```
        8 位                32 位
    |-----------S 盒 1-----------
    |                          |加
    |  8 位                32 位  |----
    |-----------S 盒 2-----------  |
    |                            |
    |                            |异或----
32 位-|                            |      |
    |  8 位                32 位      |      |
    |-----------S 盒 3---------------       |加
    |                                      |-----------------32 位
    |                                      |
    |                                      |
    |  8 位                32 位               |
    |-----------S 盒 4----------------------
```

把 xL 分成 4 个 8 位分组:a,b,c 和 d

输出为：F(xL)=((((S[1,a]+S[2,b])MOD 4294967296)^s[3,c])+S[4,d])MOD 4294967296

(2 的 32 次方)　　(2 的 32 次方)

重新合并后输出的结果就是我们需要的密文。

用 BlowFish 算法解密，同样也需要两个过程。

1.密钥预处理

2.信息解密

密钥预处理的过程与加密时完全相同

信息解密的过程就是把信息加密过程的 key_pbox 逆序使用即可。

可以看出，选择不同的 key，用 BlowFish 算法加密同样的信息，可以得出不同的结果。

要破解 BlowFish 算法，就是要得到 BlowFish 算法的 key。所以，使用 BlowFish 算法进行加密，最重要的也就是 key 的选择以及 key 的保密。其中 key 的选择可以使用 bf_sdk 中的_WeakKey 函数进行检验。以下是该函数的说明：

源文：

_WeakKey

Function : Test if the generated Boxes are weak

Argument : none

Return : AX = Status (1=weak, 0=good)

Affects : AX, BX, CX, DX, SI, DI, direction Flag

Description: After "_InitCrypt" you should test the Boxes with this function.
If they provide a weakness which a cryptoanalyst could use to
break the cipher a "1" is returned. In this case you should
reload the original boxes and let the user choose a different
password.

译文：

_WeakKey

功能：测试产生的 box 是否安全

参数：无

返回：AX=1 不安全；AX=0 安全

影响：AX, BX, CX, DX, SI, DI, 方向标志

描述：使用"_InitCrypt"函数产生用于加密的 Boxes 后，你应该用这个函数测试产生的 Boxes 是否安全。如果该 key 产生的 Boxes 不安全——可以被密码分析者通过分析 Boxes 得到 key，那么，你应该采用另外一个 key 产生一个安全的 Boxes 用来加密。

二、BlowFish's CrackMe1 分析

由于该 CrackMe 主要是测试你的密码学知识，所以没有在其他方面设关卡。为了减小文件体积，缩短大家下载的时间，用 upx 加了壳，直接用 Trw2000 的"PNewSec+Makepe"很方便地就能脱掉壳。

用常规的方法，很快找到下面关键比较处：

```
:004015D9 51                    push ecx
:004015DA 52                    push edx
:004015DB 6880894000            push 00408980
:004015E0 E8EBFAFFFF            call 004010D0      //BF_De(sn)
:004015E5 8B442464              mov eax, dword ptr [esp+64]
:004015E9 8B0DF0994000          mov ecx, dword ptr [004099F0]
:004015EF 83C41C                add esp, 0000001C
```

```
:004015F2 3BC1                    cmp eax, ecx        //比较
:004015F4 7529                    jne 0040161F
:004015F6 8B4C244C                mov ecx, dword ptr [esp+4C]
:004015FA A1EC994000              mov eax, dword ptr [004099EC]
:004015FF 3BC8                    cmp ecx, eax        //比较
:00401601 751C                    jne 0040161F
:00401603 6A30                    push 00000030
```

由于 BlowFish 算法加密，解密输出的信息都是 64Bit 的，所以要进行两次比较。

我们既然知道了他对我们的 sn 进行的变换是 BF_De，那么，很显然，我们要找到程序初始化 key_pbox 和 key_sbox 的地方。跟进 4015E0 的 Call，找到 key_pbox 在 408980 处，下 bpm，然后跟踪，分析，找到程序初始化 key_pbox 和 key_sbox 的地方，如下：

```
:004016C0 50                      push eax

* Possible StringData Ref from Data Obj ->"CrackingForFun"
                                  |
:004016C1 6844804000              push 00408044
:004016C6 6880894000              push 00408980
:004016CB E860FAFFFF              call 00401130   //初始化 Boxes
```

由此我们知道了 BF_De(sn)的 key 是"CrackingForFun"。

问题的一半已经解决了。下面我们来看用来比较的另外的 64Bit 的数是从何而来。

bpm 4099EC w

跟踪分析后，发现这个用来比较的数是由 BF_En(ComputerID,key="ChinaCrackingGroup")生成。

至此，我们可以写出注册机的算法：

sn=BF_En((BF_En(ComputerID,key="ChinaCrackingGroup"),key="CrackingForFun")

只要你编程够强，密码学也还过得去，写出这个东西的注册机就不是困难的事情了。

附：

ComputerID 的产生

如果你对这个 CrackMe 很有兴趣，还想研究一下他的 ComputerID 是如何产生的，也可以继续跟踪，分析，在这里，我给处我分析的结果：

ComputerID=BF_En(0776f6c62h, 068736966h,key=PW_1)

其中，PW_1 就是你的 Windows 版本号，可以在“系统属性”里头看到，也就是注册表中的 H_L_M\Software\Microsoft\Windows\CurrentVersion 中的 ProductId 项。在我的机器上是：

"25001-OEM-0080247-46673"

注册机源码里头有一些语句没有派上用场，用“;”屏蔽了，如果你有兴趣，可以把前面的;号去掉然后把.data 段里头的 PW_1 换成你机器的 ComputerID,再按照程序中的说明自己修改一下源程序，用 Masm32V6 重新编译，直接按 Generate，也能得到正确的序列号。

三、注册机源码

```
;BlowFish's Crackme's KeyGen Writen By  夜月[CCG]
;Any Questions,Please E-Mail To luoyi.ly@yeah.net
;Thancks To Garfield,BlowFish,Toye
;软件流程：
```

```
;1.GetVersion 得到机器 Windows 版本号。PW_1
;2.固定字符串"ChinaCrackingGroup"。PW_2
;3.固定字符串"CrackingForFun"。PW_3
;4.你输入的字符串。sn
;BF_En(0776f6c62h, 068736966h,key=PW_1)得到 Computer ID
;BF_En(ComputerID,key=PW_2)得到 MagicNum
;IF(BF_De(sn,key=PW_3)==MagicNum) Then Registed OK!


        .386
        .model flat,stdcall
        option casemap:none
include         windows.inc
include         user32.inc
include         kernel32.inc
include         comctl32.inc
include         comdlg32.inc
include         masm32.inc

includelib      masm32.lib
includelib      user32.lib
includelib      kernel32.lib
includelib      comctl32.lib
includelib      comdlg32.lib

DLG_MAIN        equ     100
IDGEN           equ     10
Edit1           equ     11
Edit2           equ     12

len_PW_1        equ offset data1_p - offset PW_1

_ProcDlgMain    PROTO   :DWORD,:DWORD,:DWORD,:DWORD
_Math           PROTO   :DWORD,:DWORD,:DWORD
BlowFish_En     PROTO   :DWORD,:DWORD
BlowFish_Fun    PROTO   :DWORD
BlowFish_Init   PROTO   :DWORD,:DWORD


        .data?
hInstance       dd      ?

        .data
;如果你直接用 ComputerID 产生序列号，你应该把 PW_1 换成你自己机器的 Windows 版本号
```

```
;PW_1     db "25001-OEM-0080247-46673"
PW_2     db "ChinaCrackingGroup"
PW_3     db "CrackingForFun"
szID          db 20 dup(0)
szText        db 9 dup(0)
data1_p       dd 0776f6c62h, 068736966h
key     dd 1058 dup (0)
BFLOW         dd 0

BFHIGH     dd 0
MYBFLOW      DD 0
MYBFHIGH      DD 0

pbox     dd 0243f6a88h, 085a308d3h, 013198a2eh, 003707344h, 0a4093822h, 0299f31d0h
    dd 0082efa98h, 0ec4e6c89h, 0452821e6h, 038d01377h, 0be5466cfh, 034e90c6ch
    dd 0c0ac29b7h, 0c97c50ddh, 03f84d5b5h, 0b5470917h, 09216d5d9h, 08979fb1bh


sbox1     dd 0d1310ba6h, 098dfb5ach, 02ffd72dbh, 0d01adfb7h, 0b8e1afedh, 06a267e96h
    dd 0ba7c9045h, 0f12c7f99h, 024a19947h, 0b3916cf7h, 00801f2e2h, 0858efc16h
    dd 0636920d8h, 071574e69h, 0a458fea3h, 0f4933d7eh, 00d95748fh, 0728eb658h
    dd 0718bcd58h, 082154aeeh, 07b54a41dh, 0c25a59b5h, 09c30d539h, 02af26013h
    dd 0c5d1b023h, 0286085f0h, 0ca417918h, 0b8db38efh, 08e79dcb0h, 0603a180eh
    dd 06c9e0e8bh, 0b01e8a3eh, 0d71577c1h, 0bd314b27h, 078af2fdah, 055605c60h
    dd 0e65525f3h, 0aa55ab94h, 057489862h, 063e81440h, 055ca396ah, 02aab10b6h
    dd 0b4cc5c34h, 01141e8ceh, 0a15486afh, 07c72e993h, 0b3ee1411h, 0636fbc2ah
    dd 02ba9c55dh, 0741831f6h, 0ce5c3e16h, 09b87931eh, 0afd6ba33h, 06c24cf5ch
    dd 07a325381h, 028958677h, 03b8f4898h, 06b4bb9afh, 0c4bfe81bh, 066282193h
    dd 061d809cch, 0fb21a991h, 0487cac60h, 05dec8032h, 0ef845d5dh, 0e98575b1h
    dd 0dc262302h, 0eb651b88h, 023893e81h, 0d396acc5h, 00f6d6ff3h, 083f44239h
    dd 02e0b4482h, 0a4842004h, 069c8f04ah, 09e1f9b5eh, 021c66842h, 0f6e96c9ah
    dd 0670c9c61h, 0abd388f0h, 06a51a0d2h, 0d8542f68h, 0960fa728h, 0ab5133a3h
    dd 06eef0b6ch, 0137a3be4h, 0ba3bf050h, 07efb2a98h, 0a1f1651dh, 039af0176h
    dd 066ca593eh, 082430e88h, 08cee8619h, 0456f9fb4h, 07d84a5c3h, 03b8b5ebeh
    dd 0e06f75d8h, 085c12073h, 0401a449fh, 056c16aa6h, 04ed3aa62h, 0363f7706h
    dd 01bfedf72h, 0429b023dh, 037d0d724h, 0d00a1248h, 0db0fead3h, 049f1c09bh
    dd 0075372c9h, 080991b7bh, 025d479d8h, 0f6e8def7h, 0e3fe501ah, 0b6794c3bh
    dd 0976ce0bdh, 004c006bah, 0c1a94fb6h, 0409f60c4h, 05e5c9ec2h, 0196a2463h
    dd 068fb6fafh, 03e6c53b5h, 01339b2ebh, 03b52ec6fh, 06dfc511fh, 09b30952ch
    dd 0cc814544h, 0af5ebd09h, 0bee3d004h, 0de334afdh, 0660f2807h, 0192e4bb3h
    dd 0c0cba857h, 045c8740fh, 0d20b5f39h, 0b9d3fbdbh, 05579c0bdh, 01a60320ah
    dd 0d6a100c6h, 0402c7279h, 0679f25feh, 0fb1fa3cch, 08ea5e9f8h, 0db3222f8h
    dd 03c7516dfh, 0fd616b15h, 02f501ec8h, 0ad0552abh, 0323db5fah, 0fd238760h
    dd 053317b48h, 03e00df82h, 09e5c57bbh, 0ca6f8ca0h, 01a87562eh, 0df1769dbh
```

dd 0d542a8f6h, 0287effc3h, 0ac6732c6h, 08c4f5573h, 0695b27b0h, 0bbca58c8h
dd 0e1ffa35dh, 0b8f011a0h, 010fa3d98h, 0fd2183b8h, 04afcb56ch, 02dd1d35bh
dd 09a53e479h, 0b6f84565h, 0d28e49bch, 04bfb9790h, 0e1ddf2dah, 0a4cb7e33h
dd 062fb1341h, 0cee4c6e8h, 0ef20cadah, 036774c01h, 0d07e9efeh, 02bf11fb4h
dd 095dbda4dh, 0ae909198h, 0eaad8e71h, 06b93d5a0h, 0d08ed1d0h, 0afc725e0h
dd 08e3c5b2fh, 08e7594b7h, 08ff6e2fbh, 0f2122b64h, 08888b812h, 0900df01ch
dd 04fad5ea0h, 0688fc31ch, 0d1cff191h, 0b3a8c1adh, 02f2f2218h, 0be0e1777h
dd 0ea752dfeh, 08b021fa1h, 0e5a0cc0fh, 0b56f74e8h, 018acf3d6h, 0ce89e299h
dd 0b4a84fe0h, 0fd13e0b7h, 07cc43b81h, 0d2ada8d9h, 0165fa266h, 080957705h
dd 093cc7314h, 0211a1477h, 0e6ad2065h, 077b5fa86h, 0c75442f5h, 0fb9d35cfh
dd 0ebcdaf0ch, 07b3e89a0h, 0d6411bd3h, 0ae1e7e49h, 000250e2dh, 02071b35eh
dd 0226800bbh, 057b8e0afh, 02464369bh, 0f009b91eh, 05563911dh, 059dfa6aah
dd 078c14389h, 0d95a537fh, 0207d5ba2h, 002e5b9c5h, 083260376h, 06295cfa9h
dd 011c81968h, 04e734a41h, 0b3472dcah, 07b14a94ah, 01b510052h, 09a532915h
dd 0d60f573fh, 0bc9bc6e4h, 02b60a476h, 081e67400h, 008ba6fb5h, 0571be91fh
dd 0f296ec6bh, 02a0dd915h, 0b6636521h, 0e7b9f9b6h, 0ff34052eh, 0c5855664h
dd 053b02d5dh, 0a99f8fa1h, 008ba4799h, 06e85076ah

sbox2 dd 04b7a70e9h, 0b5b32944h
dd 0db75092eh, 0c4192623h, 0ad6ea6b0h, 049a7df7dh, 09cee60b8h, 08fedb266h
dd 0ecaa8c71h, 0699a17ffh, 05664526ch, 0c2b19ee1h, 0193602a5h, 075094c29h
dd 0a0591340h, 0e4183a3eh, 03f54989ah, 05b429d65h, 06b8fe4d6h, 099f73fd6h
dd 0a1d29c07h, 0efe830f5h, 04d2d38e6h, 0f0255dc1h, 04cdd2086h, 08470eb26h
dd 06382e9c6h, 0021ecc5eh, 009686b3fh, 03ebaefc9h, 03c971814h, 06b6a70a1h
dd 0687f3584h, 052a0e286h, 0b79c5305h, 0aa500737h, 03e07841ch, 07fdeae5ch
dd 08e7d44ech, 05716f2b8h, 0b03ada37h, 0f0500c0dh, 0f01c1f04h, 00200b3ffh
dd 0ae0cf51ah, 03cb574b2h, 025837a58h, 0dc0921bdh, 0d19113f9h, 07ca92ff6h
dd 094324773h, 022f54701h, 03ae5e581h, 037c2dadch, 0c8b57634h, 09af3dda7h
dd 0a9446146h, 00fd0030eh, 0ecc8c73eh, 0a4751e41h, 0e238cd99h, 03bea0e2fh
dd 03280bba1h, 0183eb331h, 04e548b38h, 04f6db908h, 06f420d03h, 0f60a04bfh
dd 02cb81290h, 024977c79h, 05679b072h, 0bcaf89afh, 0de9a771fh, 0d9930810h
dd 0b38bae12h, 0dccf3f2eh, 05512721fh, 02e6b7124h, 0501adde6h, 09f84cd87h
dd 07a584718h, 07408da17h, 0bc9f9abch, 0e94b7d8ch, 0ec7aec3ah, 0db851dfah
dd 063094366h, 0c464c3d2h, 0ef1c1847h, 03215d908h, 0dd433b37h, 024c2ba16h
dd 012a14d43h, 02a65c451h, 050940002h, 0133ae4ddh, 071dff89eh, 010314e55h
dd 081ac77d6h, 05f11199bh, 0043556f1h, 0d7a3c76bh, 03c11183bh, 05924a509h
dd 0f28fe6edh, 097f1fbfah, 09ebabf2ch, 01e153c6eh, 086e34570h, 0eae96fb1h
dd 0860e5e0ah, 05a3e2ab3h, 0771fe71ch, 04e3d06fah, 02965dcb9h, 099e71d0fh
dd 0803e89d6h, 05266c825h, 02e4cc978h, 09c10b36ah, 0c6150ebah, 094e2ea78h
dd 0a5fc3c53h, 01e0a2df4h, 0f2f74ea7h, 0361d2b3dh, 01939260fh, 019c27960h
dd 05223a708h, 0f71312b6h, 0ebadfe6eh, 0eac31f66h, 0e3bc4595h, 0a67bc883h
dd 0b17f37d1h, 0018cff28h, 0c332ddefh, 0be6c5aa5h, 065582185h, 068ab9802h
dd 0eecea50fh, 0db2f953bh, 02aef7dadh, 05b6e2f84h, 01521b628h, 029076170h
dd 0ecdd4775h, 0619f1510h, 013cca830h, 0eb61bd96h, 00334fe1eh, 0aa0363cfh

dd 0b5735c90h, 04c70a239h, 0d59e9e0bh, 0cbaade14h, 0eecc86bch, 060622ca7h
dd 09cab5cabh, 0b2f3846eh, 0648b1eafh, 019bdf0cah, 0a02369b9h, 0655abb50h
dd 040685a32h, 03c2ab4b3h, 0319ee9d5h, 0c021b8f7h, 09b540b19h, 0875fa099h
dd 095f7997eh, 0623d7da8h, 0f837889ah, 097e32d77h, 011ed935fh, 016681281h
dd 00e358829h, 0c7e61fd6h, 096dedfa1h, 07858ba99h, 057f584a5h, 01b227263h
dd 09b83c3ffh, 01ac24696h, 0cdb30aebh, 0532e3054h, 08fd948e4h, 06dbc3128h
dd 058ebf2efh, 034c6ffeah, 0fe28ed61h, 0ee7c3c73h, 05d4a14d9h, 0e864b7e3h
dd 042105d14h, 0203e13e0h, 045eee2b6h, 0a3aaabeah, 0db6c4f15h, 0facb4fd0h
dd 0c742f442h, 0ef6abbb5h, 0654f3b1dh, 041cd2105h, 0d81e799eh, 086854dc7h
dd 0e44b476ah, 03d816250h, 0cf62a1f2h, 05b8d2646h, 0fc8883a0h, 0c1c7b6a3h
dd 07f1524c3h, 069cb7492h, 047848a0bh, 05692b285h, 0095bbf00h, 0ad19489dh
dd 01462b174h, 023820e00h, 058428d2ah, 00c55f5eah, 01dadf43eh, 0233f7061h
dd 03372f092h, 08d937e41h, 0d65fecf1h, 06c223bdbh, 07cde3759h, 0cbee7460h
dd 04085f2a7h, 0ce77326eh, 0a6078084h, 019f8509eh, 0e8efd855h, 061d99735h
dd 0a969a7aah, 0c50c06c2h, 05a04abfch, 0800bcadch, 09e447a2eh, 0c3453484h
dd 0fdd56705h, 00e1e9ec9h, 0db73dbd3h, 0105588cdh, 0675fda79h, 0e3674340h
dd 0c5c43465h, 0713e38d8h, 03d28f89eh, 0f16dff20h, 0153e21e7h, 08fb03d4ah
dd 0e6e39f2bh, 0db83adf7h

sbox3 dd 0e93d5a68h, 0948140f7h, 0f64c261ch, 094692934h
dd 0411520f7h, 07602d4f7h, 0bcf46b2eh, 0d4a20068h, 0d4082471h, 03320f46ah
dd 043b7d4b7h, 0500061afh, 01e39f62eh, 097244546h, 014214f74h, 0bf8b8840h
dd 04d95fc1dh, 096b591afh, 070f4ddd3h, 066a02f45h, 0bfbc09ech, 003bd9785h
dd 07fac6dd0h, 031cb8504h, 096eb27b3h, 055fd3941h, 0da2547e6h, 0abca0a9ah
dd 028507825h, 0530429f4h, 00a2c86dah, 0e9b66dfbh, 068dc1462h, 0d7486900h
dd 0680ec0a4h, 027a18deeh, 04f3ffea2h, 0e887ad8ch, 0b58ce006h, 07af4d6b6h
dd 0aace1e7ch, 0d3375fech, 0ce78a399h, 0406b2a42h, 020fe9e35h, 0d9f385b9h
dd 0ee39d7abh, 03b124e8bh, 01dc9faf7h, 04b6d1856h, 026a36631h, 0eae397b2h
dd 03a6efa74h, 0dd5b4332h, 06841e7f7h, 0ca7820fbh, 0fb0af54eh, 0d8feb397h
dd 0454056ach, 0ba489527h, 055533a3ah, 020838d87h, 0fe6ba9b7h, 0d096954bh
dd 055a867bch, 0a1159a58h, 0cca92963h, 099e1db33h, 0a62a4a56h, 03f3125f9h
dd 05ef47e1ch, 09029317ch, 0fdf8e802h, 004272f70h, 080bb155ch, 005282ce3h
dd 095c11548h, 0e4c66d22h, 048c1133fh, 0c70f86dch, 007f9c9eeh, 041041f0fh
dd 0404779a4h, 05d886e17h, 0325f51ebh, 0d59bc0d1h, 0f2bcc18fh, 041113564h
dd 0257b7834h, 0602a9c60h, 0dff8e8a3h, 01f636c1bh, 00e12b4c2h, 002e1329eh
dd 0af664fd1h, 0cad18115h, 06b2395e0h, 0333e92e1h, 03b240b62h, 0eebeb922h
dd 085b2a20eh, 0e6ba0d99h, 0de720c8ch, 02da2f728h, 0d0127845h, 095b794fdh
dd 0647d0862h, 0e7ccf5f0h, 05449a36fh, 0877d48fah, 0c39dfd27h, 0f33e8d1eh
dd 00a476341h, 0992eff74h, 03a6f6eabh, 0f4f8fd37h, 0a812dc60h, 0a1ebddf8h
dd 0991be14ch, 0db6e6b0dh, 0c67b5510h, 06d672c37h, 02765d43bh, 0dcd0e804h
dd 0f1290dc7h, 0cc00ffa3h, 0b5390f92h, 0690fed0bh, 0667b9ffbh, 0cedb7d9ch
dd 0a091cf0bh, 0d9155ea3h, 0bb132f88h, 0515bad24h, 07b9479bfh, 0763bd6ebh
dd 037392eb3h, 0cc115979h, 08026e297h, 0f42e312dh, 06842ada7h, 0c66a2b3bh
dd 012754ccch, 0782ef11ch, 06a124237h, 0b79251e7h, 006a1bbe6h, 04bfb6350h

dd 01a6b1018h, 011caedfah, 03d25bdd8h, 0e2e1c3c9h, 044421659h, 00a121386h
dd 0d90cec6eh, 0d5abea2ah, 064af674eh, 0da86a85fh, 0bebfe988h, 064e4c3feh
dd 09dbc8057h, 0f0f7c086h, 060787bf8h, 06003604dh, 0d1fd8346h, 0f6381fb0h
dd 07745ae04h, 0d736fccch, 083426b33h, 0f01eab71h, 0b0804187h, 03c005e5fh
dd 077a057beh, 0bde8ae24h, 055464299h, 0bf582e61h, 04e58f48fh, 0f2ddfda2h
dd 0f474ef38h, 08789bdc2h, 05366f9c3h, 0c8b38e74h, 0b475f255h, 046fcd9b9h
dd 07aeb2661h, 08b1ddf84h, 0846a0e79h, 0915f95e2h, 0466e598eh, 020b45770h
dd 08cd55591h, 0c902de4ch, 0b90bace1h, 0bb8205d0h, 011a86248h, 07574a99eh
dd 0b77f19b6h, 0e0a9dc09h, 0662d09a1h, 0c4324633h, 0e85a1f02h, 009f0be8ch
dd 04a99a025h, 01d6efe10h, 01ab93d1dh, 00ba5a4dfh, 0a186f20fh, 02868f169h
dd 0dcb7da83h, 0573906feh, 0a1e2ce9bh, 04fcd7f52h, 050115e01h, 0a70683fah
dd 0a002b5c4h, 00de6d027h, 09af88c27h, 0773f8641h, 0c3604c06h, 061a806b5h
dd 0f0177a28h, 0c0f586e0h, 0006058aah, 030dc7d62h, 011e69ed7h, 02338ea63h
dd 053c2dd94h, 0c2c21634h, 0bbcbee56h, 090bcb6deh, 0ebfc7da1h, 0ce591d76h
dd 06f05e409h, 04b7c0188h, 039720a3dh, 07c927c24h, 086e3725fh, 0724d9db9h
dd 01ac15bb4h, 0d39eb8fch, 0ed545578h, 008fca5b5h, 0d83d7cd3h, 04dad0fc4h
dd 01e50ef5eh, 0b161e6f8h, 0a28514d9h, 06c51133ch, 06fd5c7e7h, 056e14ec4h
dd 0362abfceh, 0ddc6c837h, 0d79a3234h, 092638212h, 0670efa8eh, 0406000e0h

sbox4 dd 03a39ce37h, 0d3faf5cfh, 0abc27737h, 05ac52d1bh, 05cb0679eh, 04fa33742h
dd 0d3822740h, 099bc9bbeh, 0d5118e9dh, 0bf0f7315h, 0d62d1c7eh, 0c700c47bh
dd 0b78c1b6bh, 021a19045h, 0b26eb1beh, 06a366eb4h, 05748ab2fh, 0bc946e79h
dd 0c6a376d2h, 06549c2c8h, 0530ff8eeh, 0468dde7dh, 0d5730a1dh, 04cd04dc6h
dd 02939bbdbh, 0a9ba4650h, 0ac9526e8h, 0be5ee304h, 0a1fad5f0h, 06a2d519ah
dd 063ef8ce2h, 09a86ee22h, 0c089c2b8h, 043242ef6h, 0a51e03aah, 09cf2d0a4h
dd 083c061bah, 09be96a4dh, 08fe51550h, 0ba645bd6h, 02826a2f9h, 0a73a3ae1h
dd 04ba99586h, 0ef5562e9h, 0c72fefd3h, 0f752f7dah, 03f046f69h, 077fa0a59h
dd 080e4a915h, 087b08601h, 09b09e6adh, 03b3ee593h, 0e990fd5ah, 09e34d797h
dd 02cf0b7d9h, 0022b8b51h, 096d5ac3ah, 0017da67dh, 0d1cf3ed6h, 07c7d2d28h
dd 01f9f25cfh, 0adf2b89bh, 05ad6b472h, 05a88f54ch, 0e029ac71h, 0e019a5e6h
dd 047b0acfdh, 0ed93fa9bh, 0e8d3c48dh, 0283b57cch, 0f8d56629h, 079132e28h
dd 0785f0191h, 0ed756055h, 0f7960e44h, 0e3d35e8ch, 015056dd4h, 088f46dbah
dd 003a16125h, 00564f0bdh, 0c3eb9e15h, 03c9057a2h, 097271aech, 0a93a072ah
dd 01b3f6d9bh, 01e6321f5h, 0f59c66fbh, 026dcf319h, 07533d928h, 0b155fdf5h
dd 003563482h, 08aba3cbbh, 028517711h, 0c20ad9f8h, 0abcc5167h, 0ccad925fh
dd 04de81751h, 03830dc8eh, 0379d5862h, 09320f991h, 0ea7a90c2h, 0fb3e7bceh
dd 05121ce64h, 0774fbe32h, 0a8b6e37eh, 0c3293d46h, 048de5369h, 06413e680h
dd 0a2ae0810h, 0dd6db224h, 069852dfdh, 009072166h, 0b39a460ah, 06445c0ddh
dd 0586cdecfh, 01c20c8aeh, 05bbef7ddh, 01b588d40h, 0ccd2017fh, 06bb4e3bbh
dd 0dda26a7eh, 03a59ff45h, 03e350a44h, 0bcb4cdd5h, 072eacea8h, 0fa6484bbh
dd 08d6612aeh, 0bf3c6f47h, 0d29be463h, 0542f5d9eh, 0aec2771bh, 0f64e6370h
dd 0740e0d8dh, 0e75b1357h, 0f8721671h, 0af537d5dh, 04040cb08h, 04eb4e2cch
dd 034d2466ah, 00115af84h, 0e1b00428h, 095983a1dh, 006b89fb4h, 0ce6ea048h
dd 06f3f3b82h, 03520ab82h, 0011a1d4bh, 0277227f8h, 0611560b1h, 0e7933fdch

```
dd 0bb3a792bh, 0344525bdh, 0a08839e1h, 051ce794bh, 02f32c9b7h, 0a01fbac9h
dd 0e01cc87eh, 0bcc7d1f6h, 0cf0111c3h, 0a1e8aac7h, 01a908749h, 0d44fbd9ah
dd 0d0dadecbh, 0d50ada38h, 00339c32ah, 0c6913667h, 08df9317ch, 0e0b12b4fh
dd 0f79e59b7h, 043f5bb3ah, 0f2d519ffh, 027d9459ch, 0bf97222ch, 015e6fc2ah
dd 00f91fc71h, 09b941525h, 0fae59361h, 0ceb69cebh, 0c2a86459h, 012baa8d1h
dd 0b6c1075eh, 0e3056a0ch, 010d25065h, 0cb03a442h, 0e0ec6e0eh, 01698db3bh
dd 04c98a0beh, 03278e964h, 09f1f9532h, 0e0d392dfh, 0d3a0342bh, 08971f21eh
dd 01b0a7441h, 04ba3348ch, 0c5be7120h, 0c37632d8h, 0df359f8dh, 09b992f2eh
dd 0e60b6f47h, 00fe3f11dh, 0e54cda54h, 01edad891h, 0ce6279cfh, 0cd3e7e6fh
dd 01618b166h, 0fd2c1d05h, 0848fd2c5h, 0f6fb2299h, 0f523f357h, 0a6327623h
dd 093a83531h, 056cccd02h, 0acf08162h, 05a75ebb5h, 06e163697h, 088d273cch
dd 0de966292h, 081b949d0h, 04c50901bh, 071c65614h, 0e6c6c7bdh, 0327a140ah
dd 045e1d006h, 0c3f27b9ah, 0c9aa53fdh, 062a80f00h, 0bb25bfe2h, 035bdd2f6h
dd 071126905h, 0b2040222h, 0b6cbcf7ch, 0cd769c2bh, 053113ec0h, 01640e3d3h
dd 038abbd60h, 02547adf0h, 0ba38209ch, 0f746ce76h, 077afa1c5h, 020756060h
dd 085cbfe4eh, 08ae88dd8h, 07aaaf9b0h, 04cf9aa7eh, 01948c25ch, 002fb8a8ch
dd 001c36ae4h, 0d6ebe1f9h, 090d4f869h, 0a65cdea0h, 03f09252dh, 0c208e69fh
dd 0b74e6132h, 0ce77e25bh, 0578fdfe3h, 03ac372e6h


    .code
    ;s 盒变换函数
    BlowFish_Fun proc uses ebx edi esi edx ecx,BfNum:DWORD
        MOV     ECX,BfNum
MOV     AL,CL
AND     EAX,0FFh
SHR     ECX,08
MOV     EDX,EAX
MOV     AL,CL
MOV     EDI,offset key
AND     EAX,0FFh
SHR     ECX,08
MOV     ESI,EAX
MOV     EAX,ECX
SHR     EAX,08
AND     EAX,0FFh
AND     ECX,0FFh
AND     ESI,0FFFFh
AND     EDX,0FFFFh
MOV     EAX,[EDI+EAX*4+48h]
MOV     EBX,[EDI+ECX*4+0448h]
MOV     ECX,[EDI+ESI*4+0848h]
ADD     EAX,EBX
XOR     EAX,ECX
```

```
    MOV     ECX,[EDI+EDX*4+0C48h]
    ADD     EAX,ECX
    RET
        BlowFish_Fun endp

        ;BlowFish 加密算法函数
        BlowFish_En   proc uses ebx edi esi edx ecx,highbf:DWORD,lowbf:DWORD
    LOCAL   num :DWORD
    MOV     EAX,highbf
    MOV     ECX,lowbf
    MOV     EAX,[EAX]
    MOV     ESI,[ECX]
    MOV     EDI,offset key
    MOV     num,10h
    MOV     EBX,EDI
loc_40108E:
    XOR     EAX,[EBX]
    MOV     EDX,EAX
    invoke  BlowFish_Fun,EAX
    MOV     ECX,num
    XOR     EAX,ESI
    ADD     EBX,4
    DEC     ECX
    MOV     ESI,EDX
    MOV     num,ECX
    JNZ     loc_40108E

    MOV     ECX,[EDI+40h]
    MOV     EDX,[EDI+44h]
    XOR     ECX,EAX
    XOR     EDX,ESI

    MOV     [BFHIGH],EDX
    MOV     [BFLOW],ECX
    RET
        BlowFish_En   endp

        ;BlowFish 初始化函数
        BlowFish_Init proc uses ebx edi esi edx ecx,PWD:DWORD,len_PWD:DWORD
    LOCAL   pbox_num18:DWORD
    LOCAL   pbox_num4 :DWORD
    LOCAL   snum      :DWORD
;初始化 s 盒
    MOV     ESI,offset key
```

```
    MOV     EAX,offset sbox1
    LEA     ECX,[ESI+48h]
loc_401141:
    MOV     EDX,0100h
loc_401146:
    MOV     EDI,[EAX]
    ADD     EAX,4
    MOV     [ECX],EDI
    ADD     ECX,4
    DEC     EDX
    JNZ     loc_401146
    CMP     EAX,offset sbox1+1000h
    JL      loc_401141

;初始化 p 盒
;第一步：原 p 盒与 PWD 逐项异或

    MOV     EDX,PWD
    MOV     EDI,offset pbox
    XOR     EAX,EAX
    SUB     EDI,ESI
    MOV     pbox_num18,12h
loc_401173:
    XOR     ECX,ECX
    MOV     pbox_num4,04
loc_40117D:
    XOR     EBX,EBX
    MOV     BL,[EAX+EDX]
    SHL     ECX,08
    OR      ECX,EBX
    INC     EAX
    CMP     EAX,len_PWD
    JL      loc_40118E
    XOR     EAX,EAX
loc_40118E:
    MOV     EBX,pbox_num4
    DEC     EBX
    MOV     pbox_num4,EBX
    JNZ     loc_40117D
    MOV     EBX,[EDI+ESI]
    ADD     ESI,4
    XOR     EBX,ECX
    MOV     ECX,pbox_num18
    MOV     [ESI-04],EBX
```

```
    DEC         ECX
    MOV         pbox_num18,ECX
    JNZ         loc_401173

;用连续的 blowfish 算法填充 p 盒
    MOV         EBX,offset key
    XOR         EAX,EAX
    MOV         BFLOW,EAX
    MOV         BFHIGH,EAX
    MOV         ESI,EBX
    MOV         EDI,09
loc_4011C4:
    LEA         EAX,BFLOW
    LEA         ECX,BFHIGH
    invoke   BlowFish_En,ECX,EAX
    MOV         EAX,BFHIGH
    MOV         ECX,BFLOW
    MOV         [ESI],EAX
    MOV         [ESI+04],ECX
    ADD         ESI,8
    DEC         EDI
    JNZ         loc_4011C4

;用连续的 blowfish 算法填充 s 盒
    LEA         ESI,[EBX+4Ch]
    MOV         snum,04         ;4 个 s 盒。
loc_4011F2:
    MOV         EDI,80H         ;每个盒填充 80h=128 次（每次填充两个数）。
loc_4011F7:
    LEA         ECX,BFLOW
    LEA         EDX,BFHIGH
    invoke   BlowFish_En,EDX,ECX
    MOV         ECX,BFHIGH
    MOV         EDX,BFLOW
    MOV         [ESI-04],ECX
    MOV         [ESI],EDX
    ADD         ESI,8
    DEC         EDI
    JNZ         loc_4011F7
    DEC         snum
    JNZ         loc_4011F2
    RET
      BlowFish_Init endp
```

```
;消息处理函数
_ProcDlgMain proc uses ebx edi esi edx
ecx,hWnd:DWORD,wMsg:DWORD,wParam:DWORD,lParam:DWORD
mov     eax,wMsg
.if     eax==WM_CLOSE
        invoke   EndDialog,hWnd,NULL
.elseif eax==WM_COMMAND
        mov   eax,wParam
        and   eax,0ffffh
        .if     eax==IDGEN
;如果你直接用 ComputerID 产生序列号，从这里到 mov MYBFLOW,ebx 一段应该屏蔽
invoke   GetDlgItemText,hWnd,Edit1,offset szID,17
xor     ebx,ebx
xor     eax,eax
mov     esi,offset szID
mov     ecx,8
@@33:

or      ebx,eax
xor     eax,eax
lodsb
cmp     eax,39h
jle     @@3
sub     eax,7
@@3:
sub     eax,30h

shl     ebx,4
loop    @@33
or      ebx,eax
mov     MYBFHIGH,ebx

mov     esi,offset szID+8
mov     ecx,8
xor     eax,eax
xor     ebx,ebx
@@44:
or      ebx,eax
lodsb
cmp     eax,39h
jle     @@4
sub     eax,7
@@4:
sub     eax,30h
```

```
        shl     ebx,4
        loop    @@44
        or      ebx,eax
        mov     MYBFLOW,ebx
;……
;如果你直接用 ComputerID 产生序列号，这里后面的所有语句你都应将其激活
;       invoke  BlowFish_Init,offset PW_1,23
;       invoke  BlowFish_En,offset data1_p,offset data1_p+4
;       MOV     EAX,BFHIGH
;       MOV     MYBFHIGH,EAX
;       MOV     EAX,BFLOW
;       MOV     MYBFLOW,EAX
        invoke  BlowFish_Init,offset PW_2,18
        invoke  BlowFish_En,offset MYBFHIGH,offset MYBFLOW
        MOV     EAX,BFHIGH
        MOV     MYBFHIGH,EAX
        MOV     EAX,BFLOW
        MOV     MYBFLOW,EAX
        invoke  BlowFish_Init,offset PW_3,14
        invoke  BlowFish_En,offset MYBFHIGH,offset MYBFLOW

        mov     ebx,BFHIGH
        mov     eax,ebx
        mov     edi,offset szText
        mov     ecx,8
    @@12:
        mov     eax,ebx
        shl     ebx,4
        shr     eax,28
        cmp     eax,9
        jle     @@11
        add     eax,7
    @@11:       add     eax,30h
        and     eax,0ffh
        stosb
        loop    @@12

        mov     ebx,BFLOW
        mov     eax,ebx
        mov     edi,offset szText+8
        mov     ecx,8
    @@22:
        mov     eax,ebx
        shl     ebx,4
```

```
        shr     eax,28
        cmp     eax,9
        jle     @@21
        add     eax,7
    @@21:       add     eax,30h
        and     eax,0ffh
        stosb
        loop    @@22

        xor     eax,eax
        mov     [edi],eax
        invoke  SetDlgItemText,hWnd,Edit2,offset szText
        mov     eax,FALSE
                        ret
                .elseif eax==IDCLOSE
                        invoke  EndDialog,hWnd,NULL
                .endif
        .else
                mov     eax,FALSE
                ret
        .endif
        mov     eax,TRUE
        ret

_ProcDlgMain endp

        ;主程序
start:
        invoke  InitCommonControls
        invoke  GetModuleHandle,NULL
        mov     hInstance,eax
        invoke  DialogBoxParam,hInstance,DLG_MAIN,NULL,offset _ProcDlgMain,0
        invoke  ExitProcess,NULL
end     start

end

;资源文件：rsrc.rc
;#include        <Resource.h>
;#define         IDGEN       10
;#define         DLG_MAIN    100
;#define         EDIT1       11
;#define         EDIT2       12
;
```

```
;DLG_MAIN     DIALOGEX     100,150,250,60
;STYLE         DS_MODALFRAME|WS_POPUP|WS_VISIBLE|WS_CAPTION|WS_SYSMENU|WS_THICKFR
AME
;CAPTION         "BlowFish's CrackMe KenGen By  夜月[CCG]   "
;FONT         9,"宋体"
;
;BEGIN
;CONTROL           " ID:",-1,"Static",SS_LEFT,10,13,40,17
;CONTROL           "SN:"   ,-2,"Static",SS_CENTER,10,40,20,17
;CONTROL           ""      ,11,"Edit",ES_LEFT,30,13,150,10
;CONTROL           ""      ,12,"Edit",ES_LEFT,30,40,150,10
;CONTROL           "GENERATE",IDGEN,"BUTTON",BS_PUSHBUTTON,200,11,40,15
;CONTROL           "EXIT",IDCLOSE,"BUTTON",BS_PUSHBUTTON,200,36,41,14
;END
```

标 题:BlowFish's CrackMe1 算法分析，以前夜月写过 (18 千字)
发信人:DiKeN
时 间:2002-4-11 13:53:00
详细信息:

```
============================================================
=
=            BlowFish's CrackMe1 验证算法分析
=                              DiKeN/OCG
============================================================

* Possible Reference to Dialog: DialogID_0065, CONTROL_ID:03EB, ""
                                  |
:004015A4 68EB030000                push 000003EB
:004015A9 56                        push esi

* Reference To: USER32.GetDlgItemTextA, Ord:0000h
                                  |
:004015AA FF151C614000              Call dword ptr [0040611C]
:004015B0 85C0                      test eax, eax
:004015B2 0F8432010000              je 004016EA
:004015B8 8D4C244C                  lea ecx, dword ptr [esp+4C]
:004015BC 8D542448                  lea edx, dword ptr [esp+48]
:004015C0 51                        push ecx
:004015C1 52                        push edx
:004015C2 8D44240C                  lea eax, dword ptr [esp+0C]

* Possible StringData Ref from Data Obj ->"%08lX%08lX"
```

```
                                |
:004015C6 686C804000            push 0040806C
:004015CB 50                    push eax
:004015CC E81F020000            call 004017F0
:004015D1 8D4C245C              lea ecx, dword ptr [esp+5C]
:004015D5 8D542458              lea edx, dword ptr [esp+58]
:004015D9 51                    push ecx=========>[ecx]=0x90ABCDEF=xr
:004015DA 52                    push edx=========>[edx]=0x12345678=xl
:004015DB 6880894000            push 00408980====>P-Box(密钥盒)

:004015E0 E8EBFAFFFF            call 004010D0====>计算 Blowfish_Dec(long *xl,long *xr)
=====================================BF_Dec 过程分析===========================
    :004010D0 8B442408          mov eax, dword ptr [esp+08]
    :004010D4 8B4C240C          mov ecx, dword ptr [esp+0C]
    :004010D8 53                push ebx
    :004010D9 55                push ebp
    :004010DA 8B00              mov eax, dword ptr [eax]====>xl
    :004010DC 56                push esi
    :004010DD 8B31              mov esi, dword ptr [ecx]====>xr
    :004010DF 57                push edi
    :004010E0 8B7C2414          mov edi, dword ptr [esp+14]
    :004010E4 C744241410000000  mov [esp+14], 00000010
    :004010EC 8D5F44            lea ebx, dword ptr [edi+44]==>P-Box(FORM 18 to 1<==因此使用的
Dec)

    * Referenced by a (U)nconditional or (C)onditional Jump at Address:
    |:0040110D(C)
    |
    :004010EF 3303              xor eax, dword ptr [ebx]
    :004010F1 50                push eax
    :004010F2 57                push edi
    :004010F3 8BE8              mov ebp, eax
    :004010F5 E806FFFFFF        call 00401000
================================================================================
================================函数 F(xl)
================================================================================
        :00401000 8B4C2408      mov ecx, dword ptr [esp+08]
        :00401004 53            push ebx
        :00401005 8AC1          mov al, cl
        :00401007 56            push esi
        :00401008 25FF000000    and eax, 000000FF
        :0040100D 57            push edi
        :0040100E C1E908        shr ecx, 08
        :00401011 8BD0          mov edx, eax
```

```
:00401013 8AC1                  mov al, cl
:00401015 8B7C2410              mov edi, dword ptr [esp+10]
:00401019 25FF000000            and eax, 000000FF
:0040101E C1E908                shr ecx, 08
:00401021 8BF0                  mov esi, eax
:00401023 8BC1                  mov eax, ecx
:00401025 C1E808                shr eax, 08
:00401028 25FF000000            and eax, 000000FF
:0040102D 81E1FF000000          and ecx, 000000FF
:00401033 81E6FFFF0000          and esi, 0000FFFF
:00401039 81E2FFFF0000          and edx, 0000FFFF
:0040103F 8B448748              mov eax, dword ptr [edi+4*eax+48]
:00401043 8B9C8F48040000        mov ebx, dword ptr [edi+4*ecx+00000448]
:0040104A 8B8CB748080000        mov ecx, dword ptr [edi+4*esi+00000848]
:00401051 03C3                  add eax, ebx
:00401053 33C1                  xor eax, ecx
:00401055 8B8C97480C0000        mov ecx, dword ptr [edi+4*edx+00000C48]
:0040105C 5F                    pop edi
:0040105D 5E                    pop esi
:0040105E 03C1                  add eax, ecx
:00401060 5B                    pop ebx
:00401061 C3                    ret
```

==========

==========end 函数 F(xl)

==========

```
:004010FA 8B4C241C              mov ecx, dword ptr [esp+1C]
:004010FE 83C408                add esp, 00000008
:00401101 33C6                  xor eax, esi
:00401103 83EB04                sub ebx, 00000004
:00401106 49                    dec ecx
:00401107 8BF5                  mov esi, ebp
:00401109 894C2414              mov dword ptr [esp+14], ecx
:0040110D 75E0                  jne 004010EF
:0040110F 8B4F04                mov ecx, dword ptr [edi+04]
:00401112 8B17                  mov edx, dword ptr [edi]
:00401114 33C8                  xor ecx, eax
:00401116 8B442418              mov eax, dword ptr [esp+18]
:0040111A 33D6                  xor edx, esi
:0040111C 5F                    pop edi
:0040111D 8910                  mov dword ptr [eax], edx
:0040111F 8B542418              mov edx, dword ptr [esp+18]
:00401123 5E                    pop esi
:00401124 5D                    pop ebp
:00401125 890A                  mov dword ptr [edx], ecx
```

```
:00401127 5B                    pop ebx
:00401128 C3                    ret
```

==========BF_Dec 过程分析完毕==========

```
:004015E5 8B442464              mov eax, dword ptr [esp+64]
:004015E9 8B0DF0994000          mov ecx, dword ptr [004099F0]
:004015EF 83C41C                add esp, 0000001C
:004015F2 3BC1                  cmp eax, ecx==============>
:004015F4 7529                  jne 0040161F
:004015F6 8B4C244C              mov ecx, dword ptr [esp+4C]
:004015FA A1EC994000            mov eax, dword ptr [004099EC]=======>我们的找到这个数据的来源，
```

==========>我们定义为 Yl,Yr

==========>我们定义输入的注册码为 Ml,Mr

==========>即有 Blowfish_Dec(Ml,Mr)=Yl,Yr

==========>所以 Blowfish_Enc(Yl,Yr)=Ml,Mr

==========>我们还需要 key

```
:004015FF 3BC8                  cmp ecx, eax==============>两次比较
:00401601 751C                  jne 0040161F
:00401603 6A30                  push 00000030
```

.....

刚分析了 BF_Dec 过程，再来分析一个 Enc 过程：

==========

其实 BF_Enc 过程与 BF_Dec 完全一样，只是使用 P-Box 顺序到过来了

==========

```
:00401070 8B442408              mov eax, dword ptr [esp+08]
:00401074 8B4C240C              mov ecx, dword ptr [esp+0C]
:00401078 53                    push ebx
:00401079 55                    push ebp
:0040107A 8B00                  mov eax, dword ptr [eax]
:0040107C 56                    push esi
:0040107D 8B31                  mov esi, dword ptr [ecx]
:0040107F 57                    push edi
:00401080 8B7C2414              mov edi, dword ptr [esp+14]
:00401084 C744241410000000      mov [esp+14], 00000010
:0040108C 8BDF                  mov ebx, edi

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004010AC(C)
|
:0040108E 3303                  xor eax, dword ptr [ebx]
:00401090 50                    push eax
:00401091 57                    push edi
:00401092 8BE8                  mov ebp, eax
```

```
:00401094 E867FFFFFF        call 00401000<=========函数 F(xl)，参见上面的分析
:00401099 8B4C241C          mov ecx, dword ptr [esp+1C]
:0040109D 83C408            add esp, 00000008
:004010A0 33C6              xor eax, esi
:004010A2 83C304            add ebx, 00000004
:004010A5 49                dec ecx
:004010A6 8BF5              mov esi, ebp
:004010A8 894C2414          mov dword ptr [esp+14], ecx
:004010AC 75E0              jne 0040108E
:004010AE 8B4F40            mov ecx, dword ptr [edi+40]
:004010B1 8B5744            mov edx, dword ptr [edi+44]
:004010B4 33C8              xor ecx, eax
:004010B6 8B442418          mov eax, dword ptr [esp+18]
:004010BA 33D6              xor edx, esi
:004010BC 5F                pop edi
:004010BD 8910              mov dword ptr [eax], edx
:004010BF 8B542418          mov edx, dword ptr [esp+18]
:004010C3 5E                pop esi
:004010C4 5D                pop ebp
:004010C5 890A              mov dword ptr [edx], ecx
:004010C7 5B                pop ebx
:004010C8 C3                ret
```

==========BF_Enc 分析完毕==========

最后再来一个 Init_Key 的过程分析:

==========

```
:00401130 51                push ecx
:00401131 53                push ebx
:00401132 55                push ebp
:00401133 56                push esi
:00401134 8B742414          mov esi, dword ptr [esp+14]
:00401138 57                push edi
:00401139 B898614000        mov eax, 00406198
:0040113E 8D4E48            lea ecx, dword ptr [esi+48]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00401158(C)
|
:00401141 BA00010000        mov edx, 00000100

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00401151(C)
|
:00401146 8B38              mov edi, dword ptr [eax]==========>S-Box
```

```
:00401148 83C004                add eax, 00000004
:0040114B 8939                  mov dword ptr [ecx], edi
:0040114D 83C104                add ecx, 00000004
:00401150 4A                    dec edx
:00401151 75F3                  jne 00401146
:00401153 3D98714000            cmp eax, 00407198
:00401158 7CE7                  jl 00401141
:0040115A 8B6C2420              mov ebp, dword ptr [esp+20]
:0040115E 8B54241C              mov edx, dword ptr [esp+1C]
:00401162 BF50614000            mov edi, 00406150
:00401167 33C0                  xor eax, eax
:00401169 2BFE                  sub edi, esi
:0040116B C744241012000000      mov [esp+10], 00000012

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004011AD(C)
|
:00401173 33C9                  xor ecx, ecx
:00401175 C744242004000000      mov [esp+20], 00000004

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00401197(C)
|
:0040117D 33DB                  xor ebx, ebx
:0040117F 8A1C10                mov bl, byte ptr [eax+edx]
:00401182 C1E108                shl ecx, 08
:00401185 0BCB                  or ecx, ebx
:00401187 40                    inc eax
:00401188 3BC5                  cmp eax, ebp
:0040118A 7C02                  jl 0040118E
:0040118C 33C0                  xor eax, eax

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040118A(C)
|
:0040118E 8B5C2420              mov ebx, dword ptr [esp+20]
:00401192 4B                    dec ebx
:00401193 895C2420              mov dword ptr [esp+20], ebx
:00401197 75E4                  jne 0040117D
:00401199 8B1C37                mov ebx, dword ptr [edi+esi]
:0040119C 83C604                add esi, 00000004
:0040119F 33D9                  xor ebx, ecx
:004011A1 8B4C2410              mov ecx, dword ptr [esp+10]
:004011A5 895EFC                mov dword ptr [esi-04], ebx
```

```
:004011A8 49                    dec ecx
:004011A9 894C2410              mov dword ptr [esp+10], ecx
:004011AD 75C4                  jne 00401173
:004011AF 8B5C2418              mov ebx, dword ptr [esp+18]
:004011B3 33C0                  xor eax, eax
:004011B5 89442420              mov dword ptr [esp+20], eax
:004011B9 8944241C              mov dword ptr [esp+1C], eax
:004011BD 8BF3                  mov esi, ebx
:004011BF BF09000000            mov edi, 00000009

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004011E8(C)
|
:004011C4 8D44241C              lea eax, dword ptr [esp+1C]
:004011C8 8D4C2420              lea ecx, dword ptr [esp+20]
:004011CC 50                    push eax
:004011CD 51                    push ecx
:004011CE 53                    push ebx
:004011CF E89CFEFFFF            call 00401070================>BF_Enc(0,0,key)
:004011D4 8B54242C              mov edx, dword ptr [esp+2C]
:004011D8 8B442428              mov eax, dword ptr [esp+28]
:004011DC 8916                  mov dword ptr [esi], edx
:004011DE 894604                mov dword ptr [esi+04], eax
:004011E1 83C40C                add esp, 0000000C
:004011E4 83C608                add esi, 00000008
:004011E7 4F                    dec edi
:004011E8 75DA                  jne 004011C4
:004011EA 8D734C                lea esi, dword ptr [ebx+4C]
:004011ED BD04000000            mov ebp, 00000004

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040121E(C)
|
:004011F2 BF80000000            mov edi, 00000080

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0040121B(C)
|
:004011F7 8D4C241C              lea ecx, dword ptr [esp+1C]
:004011FB 8D542420              lea edx, dword ptr [esp+20]
:004011FF 51                    push ecx
:00401200 52                    push edx
:00401201 53                    push ebx
:00401202 E869FEFFFF            call 00401070================>BF_Enc(xl,xr,key)
```

```
:00401207 8B44242C          mov eax, dword ptr [esp+2C]
:0040120B 8B4C2428          mov ecx, dword ptr [esp+28]
:0040120F 8946FC            mov dword ptr [esi-04], eax
:00401212 890E              mov dword ptr [esi], ecx
:00401214 83C40C            add esp, 0000000C
:00401217 83C608            add esi, 00000008
:0040121A 4F                dec edi
:0040121B 75DA              jne 004011F7
:0040121D 4D                dec ebp
:0040121E 75D2              jne 004011F2
:00401220 5F                pop edi
:00401221 5E                pop esi
:00401222 5D                pop ebp
:00401223 5B                pop ebx
:00401224 59                pop ecx
:00401225 C3                ret

=====================Init_Key 过程分析完毕============================
=========================================================================
=========================================================================
=========================================================================
============================分析详细总结============================
=========================================================================
=========================================================================
=====>BF_Enc(ComputerID,key="ChinaCrackingGroup");
* Possible StringData Ref from Data Obj ->"ChinaCrackingGroup"
:00401434 6830804000        push 00408030
:00401439 6880894000        push 00408980
:0040143E E8EDFCFFFF        call 00401130===>Init_Key
...........
:00401667 68EC994000        push 004099EC
:0040166C 68F0994000        push 004099F0
:00401671 6880894000        push 00408980
:00401676 E8F5F9FFFF        call 00401070===>BF_Enc
=====>BF_Enc(ComputerID,key="ChinaCrackingGroup");
=========================================================================

=========================================================================
=====>BF_Dec(Code,key="CrackingForFun")
* Possible StringData Ref from Data Obj ->"CrackingForFun"
                                  |
:004016C1 6844804000        push 00408044
:004016C6 6880894000        push 00408980
:004016CB E860FAFFFF        call 00401130===>Init_Key
```

```
.....
:004015D9 51                    push ecx
:004015DA 52                    push edx
:004015DB 6880894000            push 00408980
:004015E0 E8EBFAFFFF            call 004010D0===>BF_Dec
=====>BF_Dec(Code,key="CrackingForFun")
=================================================================
=================================================================
=====>BF_Enc("blowfish",key=ProductID)
:0040131F 6880894000            push 00408980
:00401324 E807FEFFFF            call 00401130===>Init_Key
:00401329 68EC994000            push 004099EC
:0040132E 68F0994000            push 004099F0
:00401333 6880894000            push 00408980
:00401338 C705F0994000626C6F77  mov dword ptr [004099F0], 776F6C62
:00401342 C705EC99400066697368  mov dword ptr [004099EC], 68736966
:0040134C E81FFDFFFF            call 00401070===>BF_Enc
=====>BF_Enc("blowfish",key=ProductID)
=================================================================
=================================================================
```

=====>最后分析结果

ComputerID=BF_Enc("blowfish",key=ProductID)

x=BF_Dec(Code,key="CrackingForFun")

y=BF_Enc(ComputerID,key="ChinaCrackingGroup")

x=y 则注册成功；

我们要得到正确的注册码，那么

```
Code=BF_Enc(x,key="CrackingForFun");
  =BF_Enc(y,key="CrackingForFun");
  =BF_Enc(BF_Enc(ComputerID,key="ChinaCrackingGroup"),key="CrackingForFun");
```

如果更进一步，那么

```
  =BF_Enc(BF_Enc(BF_Enc("blowfish",
                        key=ProductID),
                 key="ChinaCrackingGroup"),
          key="CrackingForFun");
```

这样我们便可以编写它的 keygen 了