白洪欢
iceman@zju.edu.cn
http://10.71.45.100/bhh

本学期要学的《密码学》主要内容：
数学基础、古典密码、MD5、SHA-1、RC4、DES、AES、RSA、ECC
编程工具：VC6（http://10.71.45.100/bhh/VC6_Aegisys.exe）
大数库：Openssl（http://10.71.45.100/bhh/openssl.rar）
官网为www.openssl.org

第 1 章 数学基础

1.1 整除

整除的定义：设 a、b 均为整数，且 a≠0，若存在整数 k 使得 b=a*k，则称 a 整除 b，记作 a|b。
整除相关的 3 个命题：
①对于任意整数 a，都有 1|a；若 a≠0，则有 a|0 且 a|a。
②若 a|b 且 b|c，则 a|c。
③若 a|b 且 a|c，则 a|(s*b+t*c)，其中 s、t 为任意整数。

1.2 素数与互素

素数的定义：若整数 p 只有因子±1 及±p，则称 p 为素数。
互素(relatively prime)的定义：对于整数 a、b，若

gcd(a,b)=1，则称 a、b 互素。

a=3, b=5 gcd(a,b)=1

a=3, b=4 gcd(a,b)=1

a=4, b=9 gcd(a,b)=1

素数相关的定理：任一整数 a(a>0)都能唯一分解成以下形式：

$$a = p_1 * p_2 * p_3 * \ldots * p_t$$

其中 p_1、p_2、p_3、...、p_t 是素数。

1.3 最大公约数(greatest common divisor)

gcd 相关的定理：设 a、b 为整数，且 a、b 中至少有一个不等于 0，令 d=gcd(a,b)，则一定存在整数 x、y 使得下式成立：

$$a*x + b*y = d$$

特别地，当 a、b 互素时，则一定存在整数 x、y 使得 a*x+b*y=1 成立。(证明见 P.93)

1.4 模(mod)运算和同余(congruent)

同余的定义：设 a、b、n 均为整数，且 n≠0，当 a-b 是 n 的倍数时即 a=b+n*k(k 为整数)，我们称 a、b 对于模 n 同余(a is congruent to b mod n)，记作：

$$a \equiv b \pmod{n}$$

可以理解为：a%n == b %n

例如：1 ≡ 4 mod 3 可以理解为：4%3=1

例如：5 ≡ 8 mod 3

同余相关的命题：设 a,b,c,d,n 均为整数，且 n≠0，则有

①当且仅当 n|a 时，有 a≡0 (mod n)
②a≡a (mod n)
③当且仅当 b≡a (mod n)时，有 a≡b (mod n)
④若 a≡b 且 b≡c (mod n)，则一定有 a≡c (mod n)
a = k*n+x
b = k'*n+x
c = k"*n+x

⑤若 a≡b (mod n)且 c≡d (mod n)，则有
a+c≡b+d, a-c≡b-d, a*c≡b*d (mod n)

1.5 逆元(inverse)

1.5.1 加法模逆元

定义：若 a+b≡0 (mod n)，则称 a 是 b 的加法模 n 逆元，b 是 a 的加法模 n 逆元。

例如：恺撒加密法在解密时会用到加法逆元

```
  0 1 2 3 4 5 6   23 24 25
x:a b c d e f g ..x  y   z
y:d e f g h i j ..a  b   c
```

加密过程：y = (x+3) % 26
解密过程：x = (y+23) % 26
因为 23 是 3 的加法逆元，即 23 相当于-3
-3 = -26+23 = 23 mod 26

加密算法:y=(x-'a'+3)%26 + 'a';
解密算法:x=(y-'a'+23)%26 + 'a';这里的+23 相当于-3
3+23 ≡ 0 (mod 26) 表示 3 的加法逆元是 23
3-3 ≡ 0 (mod 26) 表示 3 的加法逆元是-3
所以-3 ≡ 23 (mod 26)

或者这样推理：

-3 % 26 == (-26+23) % 26 == 23 % 26

1.5.2 乘法模逆元

定义：若 a*b≡1 (mod n)，则称 a 是 b 的乘法模 n 逆元，b 是 a 的乘法模 n 逆元。a 的乘法逆元记作 a^{-1}。(乘法模 n 逆元计算过程参考 P.95)

例如：求 13 模 35 的乘法逆元

设 13 模 35 的乘法逆元为 x，则

13*x ≡ 1 (mod 35)

上述等式成立的充要条件为 gcd(13,35)=1(证明见 P.94)

于是一定存在 x、y 使得 13*x + 35*y = 1 成立，利用扩展欧几里德法(extended Euclidean algorithm)可以解出上述方程中的 x 及 y，其中的 x 就是 13 模 35 的逆元。

35=2*13+9

13=1*9+4

9=2*4+1

1 = 9-2*4 = 35-2*13 - (13-1*9)*2

= 35-2*13 - (13-(35-2*13))*2

= (35-2*13)*3 - 13*2

= 35*3 - 8*13

x=-8, y=3

其中-8 ≡ (-35+27) ≡ 27 (mod 35)

13 * (-8) = 1 mod 35

13 * (-35*27) = 13*(-35) + 13*27

= 13*27 = 1 mod 35

所以 13 模 35 的乘法逆元为 27。

2 的 mod 5 乘法逆元=3

1 的 mod 5 乘法逆元=1

3 的 mod 5 乘法逆元=2

4 的 mod 5 乘法逆元=4

---

2 的 mod 6 乘法逆元=不存在，因为 gcd(2,6)=2≠1

第 2 章 古典密码

encryption(加密)：

密钥(key)

明文(plaintext) ----------→ 密文(ciphertext)

加密

The process of encoding data to prevent unauthorized access, especially during transmission. Encryption is usually based on one or more keys, or codes, that are essential for decoding 解密, or returning the data to readable form.

decryption(解密)：

密钥(key)

密文(ciphertext)----------→ 明文(plaintext)

The process of restoring encrypted data to its original form.

2.1 单表密码

单表密码只使用一张密码字母表，且明文字母与密文字母有固定的对应关系。频率分析法可以对付单表密码。

Edgar Allan Poe "The Gold-Bug"

Arthur Conan Doyle "The Dancing Men"

2.1.1 加法密码

[demo] Julius Caesar: 恺撒加密法

序号	加密前		加密后
0	A	→	D
1	B	→	E
2	C	→	F
...	...		...
23	X	→	A
24	Y	→	B
25	Z	→	C

加密算法: y = (x-'A'+3) % 26 + 'A';
解密算法: x = (y-'A'-3) % 26 + 'A'; 或
x = (y-'A'+23) % 26 + 'A'

2.1.2 乘法密码

加密算法: $y = x*k \ \% \ n;$
解密算法: $x = y*k^{-1} \ \% \ n;$ 其中 k^{-1} 是 k 的乘法逆元

2.1.3 仿射密码

加密算法: $y = (x*k1 + k2) \ \% \ n;$
解密算法: $x = (y-k2)*k1^{-1} \ \% \ n;$

2.2 多表密码

多表密码是对每个明文字母采用不同的单表代换，即同一明文字母对应多个密文字母。

《连城诀》

2.2.1 Playfair (P.10)

2.2.2 Vigenere (P.11)

加密算法: $y = (x+k_i) \ \% \ n;$
解密算法: $x = (y-k_i) \ \% \ n;$
例如:
明文=this crypto system is not secure
(19,7,8,18,2,17,24,15,19,14,...)
密钥=cipher cipher cipher ...
(2,8,15,7,4,17)
密文=VPXZGIAXIV...

(21,15,23,25,6,8,0,23,8,21,...)
't'-'a'=19 明文
'c'-'a'=2 密钥 +)

21
(19+2) % 26 = 21
21+'A' = 'V' 密文

2.2.3 Beaufort (P.12)
2.2.4 Vernam (P.14)
2.2.5 Hill (P.14)
2.2.6 Enigma

$A_5^3 * 26^3 * (C_{26}^{20} * A_{20}^{10} / 2^{10} + C_{26}^{18} * A_{18}^{9} / 2^{9} + \ldots C_{26}^{2} * A_2^1 / 2^1 + 1)$

10 对字母在接线板上的组合=$C_{26}^{20} * (A_{20}^{10} / 2^{10})$
ABCD 中抽取 2 对字母的组合如下：
AB CD; AC BD; AD BC

如何把 MessageKey 传递给对方?
发送方随机想出 3 个齿轮的外部状态(MessageKey)为：ABC
以明文的形式把 ABC 发送给对方；
再想出今天要用到密钥即真正用来加密的齿轮初始状态为：ZJU
在当前齿轮初始状态为 ABC 的情况下，连续按下 ZJU 得到 ZJU 的密文设为 Z'J'U'发送给对方。
对方在齿轮初始状态为 ABC 的情况下，输入 Z'J'U'一定可以解密出 ZJU。
即下去双方都把齿轮的初始状态设为 ZJU，然后就可以开始正式通讯。

注意，当按下某个键时，对该键进行加密的密钥并非当前齿轮的状态，而是齿轮转了一下以后的状态。例如：设 3 个齿轮从左到右分别为 III，II，I，并且齿轮的状态从左到右为 AAZ，则输入字母 A 时，齿轮转到 AAA 的位置，此时 AAA 就是密钥。加密后得到的密文为 N。
RingSetting 是齿轮内部的初始状态，它们在齿轮转动时不会发生变化。而齿轮外部的状态 MessageKey 是会随每一次按键而发生变化的。
设 I 号齿轮的 RingSetting=B，MessageKey=D，则

Δ=MessageKey-RingSetting='D'-'B'=2
现在假定输入字母 A，则 A 进入 I 号齿轮时，需要先加上 Δ 即变成 A+2='C'，查表:'C'→'M'，而从 I 号齿轮出去时要减去 Δ，即'M'-2='K'。

RingSetting: III=A, II=A, I=B
MessageKey: III=A, II=A, I=C
敲键 A 后 MessageKey 变成: III=A, II=A, I=D
从 II 号齿轮返回到 I 号齿轮时，进入 I 号齿轮的字母为 G
此时 G+Δ=G+2=I，I 反查表得 V，V-Δ=V-2=T，最终 T 就是密文。

接线板只在从右到左进入最右侧那个齿轮以及
从左到右从最右侧齿轮出来亮灯的时候才起作用。

Enigma 模拟软件(http://10.71.45.100/bhh/enigma.rar)
加密过程要经过 5 个元件:
假定明文为 A
1. plugboard
假定接线板设置为:A-B, C-D

```
char plug[27]="BADCEFGHIJKLMNOPQRSTUVWXYZ";
              /*ABCDEFGHIJKLMNOPQRSTUVWXYZ*/
char c='A', e;
e = plug[c-'A']; // e='B';
```

A→B

2. rotor I

```
char rotor[27]="EKMFLGDQVZNTOWYHXUSPAIBRCJ";
              // ABCDEFGHIJKLMNOPQRSTUVWXYZ
char c='B', e;
e = rotor[c-'A']; // e='K';
```

B→K

3. rotor II

```
AJDKSIRUXBLHWTMCQGZNPYFVOE
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

K→L

4. rotor III

```
BDFHJLCPRTXVZNYEIWGAKMUSQO
```

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
L→V

5. reflector
char reflector[27]=
  "YRUHQSLDPXNGOKMIEBFZCWVJAT";
// ABCDEFGHIJKLMNOPQRSTUVWXYZ
V→W
```

经过上述 5 步，A 转成 W，但 W 还不是密文，接下去还要走一条逆向路径:4-3-2-1，把 W 进一步转化成密文，步骤如下：

```
4. rotor III
char rotor[27]="BDFHJLCPRTXVZNYEIWGAKMUSQO";
                // ABCDEFGHIJKLMNOPQRSTUVWXYZ
W→R
```

注意按逆向路径经过 3 个齿轮时要反查表，即在数组 rotor 中寻找元素'W'，得到该元素的下标设为 i，再把 i+'A',就得到'R'。

```
3. rotor II
AJDKSIRUXBLHWTMCQGZNPYFVOE
ABCDEFGHIJKLMNOPQRSTUVWXYZ
R→G

2. rotor I
EKMFLGDQVZNTOWYHXUSPAIBRCJ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
G→F

1. plugboard
BADCEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
F→F
```

注意接线板查表既可以正向查,也可以反向查,结果是一样的。

经过上述 4 步，W 转成 F，其中 F 就是密文。

各位可以尝试把 F 当作明文重新把前面的 9 步走一遍,最后出来的一定是 A，这就实现了解密。

要注意在 Enigma 模拟器中，除了把 3 个齿轮按从左到右顺序设成 III、II、I 外，还需要把 MessageKey 设成 01 01 26 即 AAZ，因为 Enigma 是先转动齿轮再查表的，当你敲 A 键时，

齿轮 I 会从 26 转到 1。

另外还有两个齿轮在上面的例子中没有用到，现列出它们的转化表。
齿轮 IV 的转换表如下所示：
ESOVPZJAYQUIRHXLNFTGKDCMWB
ABCDEFGHIJKLMNOPQRSTUVWXYZ

齿轮 V 的转换表如下所示：
VZBRGITYUPSDNHLXAWMJQOFECK
ABCDEFGHIJKLMNOPQRSTUVWXYZ

5 个齿轮使下一个齿轮发生跳转的字母：
QEVJZ；齿轮的当前位置，从左到右对应齿轮 I II III IV V
RFWKA；齿轮的下一步位置
；Royal Flags Wave Kings Above
假定 3 个齿轮为 III、II、I，齿轮 I 的当前位置=Q 且 II 的当前位置=A，现在敲任何一个键，都会使 I 转到 R 这个位置，此时 I 会带动 II 旋转，于是 II 会从 A 转到 B。
另外，还要注意一个 double stepping 现象：
假定 III=1=A，II=4=D，I=17=Q
现在 I 旋转，从 Q 变成 R，一定会带动 II 旋转：
III=1=A，II=5=E，I=18=R
此时再旋转 I 的话，I 本来是不应该带动 II 转的(因为当前 I 不在 Q 这个位置)，但是 II 还会再转(double stepping)，同时 II 带动 III 旋转：
III=2=B，II=6=F，I=19=S
归纳起来讲，II 有两种情况会转动：
(1) I 从 Q 转到 R
(2) II 当前在 E 位置，I 不管在什么位置
double stepping 是由 Enigma 的机械结构决定的，该现象只会出现在中间那个齿轮上。

每个齿轮有一个内部设置叫 ring setting：
假定 I 齿轮的 RingSetting=B(内部)，
MessageKey=A(外部)
现在按键盘 A 的时候，A 进入 I 齿轮后，要做以下运算：

```
char c = 'A';
int delta = MessageKey-RingSetting;
```

c = ((c-'A')+delta+26)%26 + 'A';
此时 c='Z';
接下去拿 c 去查齿轮 I 的表得 J
现在 J 要出去的时候，还得减去 delta，即：
c = c-delta; 即 c=K
所以从齿轮 I 出来的字母就是 K。另外两个齿轮的处理跟 I 一样。
从 III 到 II 到 I 逆向走回来的时候，同样，
从左到右进入要加 delta，右边出来要减 delta。
假定 I 的 RingSetting=C, MessageKey=A
现在按 A，此时 MessageKey=B
A-1➔Z 查表得 J J+1➔K

第 3 章 hash 函数

3.1 MD5

plaintext 明文 ←→ ciphertext 密文
message → digest 报文摘要，相当于有损压缩
md5 是一种单向函数
从报文到摘要，中间不需要密钥
王小云只能做到 a、b 由她自己给出时，md(a) == md(b);
但做不到对于任意 a，找出 b 使得 md(a) == md(b)。
预计算法其实也是穷举。
1.exe=0.exe+m1 2.exe=0.exe+m2
当 md5(m1)==md5(m2)时，则有
md5(1.exe) == md5(2.exe)
rsa(md5(1.exe))就是 1.exe 的数字签名
== rsa(md5(2.exe))是 2.exe 的数字签名
b1c0d8dc8d73f5ba9fb8d5bc89ffe0c1 test2.exe
b1c0d8dc8d73f5ba9fb8d5bc89ffe0c1 test1.exe

彩虹表(rainbow table)

ef863dbdf95a3cadf971f3dbf06e1c5c xM01.dat
ef863dbdf95a3cadf971f3dbf06e1c5c M01.dat

md5 算法的 message 长度为[0,n]字节，其中 n 可以无限大；而 digest 的长度固定为 128 位，即 16 字节。
假定可以找到报文 m1 及 m2(m1 ≠ m2) 使得 md5(m1)==md5(m2)，则我们称发生了 md5 碰撞(collision)。

设文件大小为 n 字节，则最后那个块的长度为 n%64。

3.1.1 md5 计算过程中的分块及填充

分块计算，每块大小为 64 字节。
当最后一块大小刚好为 64 字节时，该块只能算倒数第 2 块，即它的后面那块(大小为 0 字节)才算最后一块。故最后块的大小为[0,63]字节，该块需要按以下步骤补充数据凑成 64 字节的一个块或者 128 字节的两个块：

(1)假定该块大小 n<56 字节，则在末尾补上以下数据：
0x80 0x00 0x00 0x00 ... 0x00；共 56-n 个
例如 n==55 时，只要补 0x80 一个字节；
当 n==54 时，要补上 0x80 及 0x00 两个字节。

(2)假定该块大小 n 在[56,63]范围内时，则应在末尾补上 64-n+56 个字节。例如当 n==56 时，应该补上 64 即 8+56 个字节；当 n==57 时，应该补上 63 即 7+56 个字节。

(3)再在后面补上 8 个字节，这 8 个字节相当于一个 64 位的整数，它的值=message 总共的位数(不含填充内容)。

比如有一个文件的内容为'A'，仅一个字节，则：

```
A,0x80,0x00,…,0x00,  0x08,0x00,0x00,…,0x00
  ~~~~~~~~~~~~~~~    ====================
   共补上 55 个字节     共 8 字节，其类型为 64 位整数
                      =0x00 00 00 00 00 00 00 08
```

3.1.2 调用 Openssl 库函数 MD5()计算 md5 值

主页下载 Openssl： http://10.71.45.100/bhh/openssl.rar
\include\openssl*.h 复制到 VC 的 include 目录内；
\lib\libeay32.lib 及\lib\ssleay32.lib 复制到 VC 的 lib 目录内。

代码演示 1：

```
#include <openssl/md5.h>
#pragma comment(lib, "libeay32.lib")
```

```
#pragma comment(lib, "ssleay32.lib")
main()
{
   int i;
   unsigned char s[100]="Hello", t[100];
   MD5(s, strlen(s), t);
   for(i=0; i<16; i++)
     printf("%02X ", t[i]);
}
```

上述代码的输出结果为：

```
8B 1A 99 53 C4 61 12 96 A8 27 AB F8 C4 78 04 D7
```

代码演示 2：

```
#include <openssl/md5.h>
#pragma comment(lib, "libeay32.lib")
#pragma comment(lib, "ssleay32.lib")
main()
{
   int i;
   unsigned char s[100]="Hello", t[100];
   MD5_CTX m; //ctx:context
   MD5_Init(&m);
   MD5_Update(&m, s, 64);
   MD5_Update(&m, s+64, 64);
   MD5_Update(&m, s+64+64, 64);
   MD5_Update(&m, s+64+64+64, 1);

   MD5_Final(t, &m);
   for(i=0; i<16; i++)
     printf("%02X ", t[i]);
}
```

3.1.3 md5 源代码分析

http://10.71.45.100/bhh/mymd5.c

3.1.4 md5 碰撞

md5 经常与其它算法如 RSA 结合在一起用于数字签名。

```
m = md5(letter);
m' = rsa(m, A 的私钥); // m'就是数字签名
```

假定 A 把 letter 及 m'发送给 B，则

B 收到信后验证签名：

```
rsa(m', A 的公钥) == md5(letter)
```

假定有碰撞 md5(m1)==md5(m2)，其中计算 md5(m1)及 md5(m2)时所用的 4 个 state 种子值均等于 md5(letter)完成 Final_MD5()前的 4 个 state 值，则一定有：

```
md5(letter+m1) == md5(letter+m2)
```

因此 letter+m1 的签名可以用于 letter+m2。

http://10.71.45.100/bhh/CollideTest.rar

下载上述链接中的 CollideTest.rar，解包后可以看到有 3 个文件：test.c, test1.exe, test2.exe

其中 test.c 是源程序，test1.exe 及 test2.exe 的末尾各添加了 128 字节的能发生碰撞的 m1 及 m2，用 md5 工具可以检测 test1.exe 与 test2.exe 是相同的，但这两个程序的运行结果不同。

计算 md5(m1) == md5(m2)碰撞时，md5 的种子应该等于 md5(test.exe)，而且计算 md5(test.exe)时只能计算到 Final_MD5()之前，不能把末尾追加的数据及报文位长度也计算进去。

3.1.5 md5 破解方法：rainbow table

假定要建立一张包含 4 个大写英文字母的彩虹表，则可以按以下步骤进行：

```
for(i=1; i<=SomeBigNumber; i++)
{
  ①产生一个随机数 n0，其中 n0∈[0,26⁴-1]
  找出与 n0 对应的字母组合 p0，计算 m0=md5(p0)
  for(j=1; j<=100; j++)
  {
    ②计算 nj = mj-1 mod 26⁴ /* 消减函数 */
    ③以 nj 为序号,找出与它对应的 4 个英文字母组合 pj。
```

例如 n=0 时 p="AAAA"，n=1 时 p="AAAB"，n=25 时 p="AAAZ"，n=26 时 p="AABA"。

④计算 m_j = md5(p_j)

}

⑤在数据库中记录步骤①所得的 n_0 及 j==100 时步骤④所得的 m_{100}。

}

上述过程完成后，可以得到 SomeBigNumber 条链，每条链包含 101 个 md5 值。

假定 M 是由某 4 个大写字母组合生成的 md5 值，现要破解 M 究竟是由哪 4 个英文字母生成的，则可以按以下步骤进行：

①在数据库中查找 M，若找到，则表示 M 是 j==100 时算出来的，于是执行 for(j=1; j<=100; j++){②③④}，其中 n 的初始值 n_0 是从数据库中取出来的，不能用随机数，当 j==100 时，步骤③根据 n_{100} 得到的 4 个字母组合 p_{100} 即为所求。

②若数据库中找不到 M，则设 m_j=M，再执行②③④算出 m_{j+1}，现再在数据库中查找此 m_{j+1}，若找到，则表示 M 是 j==99 时算出来的，于是执行 for(j=1; j<=99; j++){②③④}，其中 j==99 时，步骤③根据 n_{99} 得到的 4 个字母组合 p_{99} 极有可能满足 md5(p_{99})==M。

3.2 SHA

sha-1 散列算法计算出来的 hash 值达 160 位，即 20 字节，比 md5 多了 32 位。

sha-1 也是分块计算，每块也是 64 字节，当最后一块不足 64 字节也按照 md5 的方式进行填充。

数据块的最后一定要补上表示报文总共位数的 8 个字节。

SHA-1 源代码分析：
http://10.71.45.100/bhh/MySha1.c

第 4 章 分组密码工作模式与流密码

4.1 分组密码工作模式

1. 电子密码簿 ECB

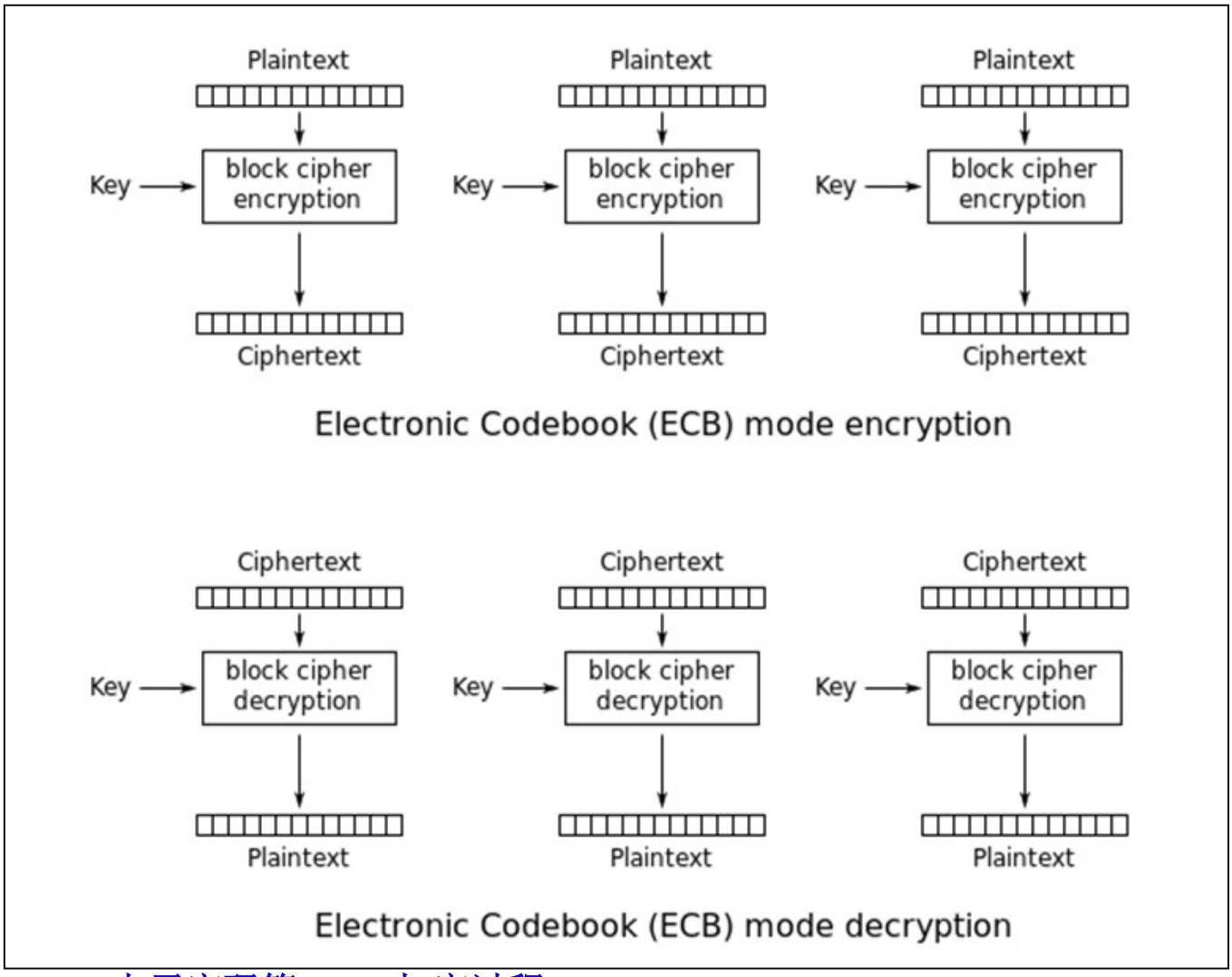
The natural manner for using a block cipher is to break a long piece of plaintext into appropriate sized blocks of plaintext and process each block separately with the encryption function $E_K()$. This is known as the electronic codebook (ECB) mode of operation. The plaintext P is broken into smaller chunks

$$P = [P_1, P_2, \ldots, P_L]$$

and the ciphertext is

$$C = [C_1, C_2, \ldots, C_L]$$

where $C_j = E_k(P_j)$ is the encryption of P_j using the key K.



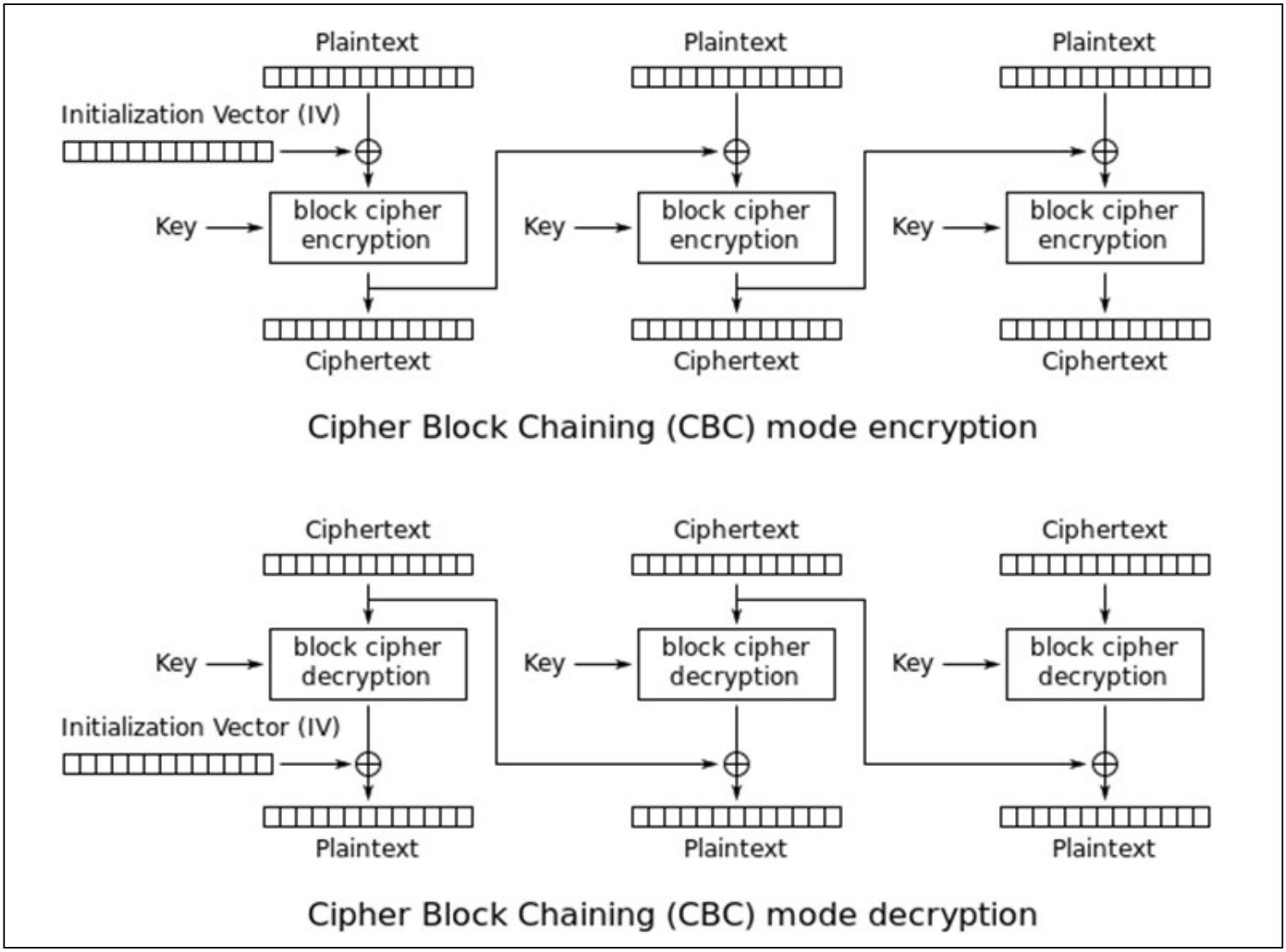
电子密码簿 **ECB** 加密过程：$C_j = E_k(P_j)$

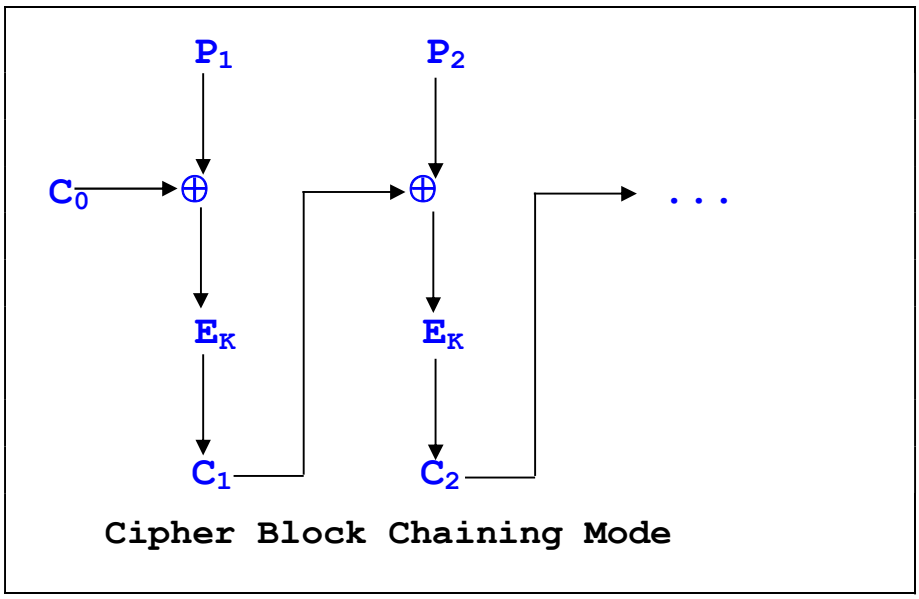
电子密码簿 **ECB** 解密过程：$P_j = D_k(C_j)$

电子密码簿模式的弱点是：对于相同内容的明文段，加密后得到的密文块是相同的。

电子密码簿模式的优点是：加密和解密过程均可以并行处理。

2．密文块链接模式 CBC





Cipher Block Chaining Mode

加密过程： $C_j = E_k(P_j \oplus C_{j-1})$

解密过程： $P_j = D_k(C_j) \oplus C_{j-1}$

CBC 模式的特点是：当前块的密文与前一块的密文有关；加密过程只能串行处理；解密过程可以并行处理；

3．密文反馈模式 CFB

Cipher feedback mode is a stream mode of operation where one 8-bit piece of message is encrypted without having to wait for an entire block of data to be available.

The plaintext is broken into 8-bit piece:

$P = [P_1, P_2, \ldots]$

where each P_j has 8 bits, rather than 64 bits used in ECB and CBC. Encryption proceeds as follows. An initial 64-bit X_1 is chosen. Then for j=1,2,3,… the following is performed:

$C_j = P_j \oplus L_8(E_k(X_j))$　　X_j 实际是一个 64 位种子数

$X_{j+1} = R_{56}(X_j) \parallel C_j$

where $L_8(X)$ denotes the 8 leftmost bits of X, $R_{56}(X)$ denotes the rightmost 56 bits of X, and $X \parallel Y$ denotes

the string obtained by writing X followed by Y. Decryption is done with the following steps:

$$P_j = C_j \oplus L_8(E_k(X_j))$$

$$X_{j+1} = R_{56}(X_j) \| C_j$$

Let's step through one round of the CFB algorithm. First, we have a 64-bit register that is initialized with X_1. These 64 bits are encrypted using E_k and the leftmost 8 bits of $E_K(X_1)$ are extracted and XORed with the 8-bit P_1 to form C_1. Then C1 is sent to the recipient. Before working P_2, the 64-bit register X_1 is updated by extracting the rightmost 56 bits. The 8 bits of C_1 are appended on the right to form $X_2 = R_{56}(X_1) \| C_1$. After P_2 is encrypted to C_2, the 64-bit register is updated to form

$$X_3 = R_{56}(X_2) \| C_2 = R_{48}(X_1) \| C_1 \| C_2$$

密文反馈模式的优点是可以从密文传输的错误中恢复。比如要传输密文 $C_1, C_2, C_3, \ldots, C_K$，现假定 C_1 传输错误，把它记作 C_1'，则解密还原得到的 P_1 有错。若 X_1 记作

$$(*, *, *, *, *, *, *, *)$$

则

$$X_2 = (*, *, *, *, *, *, *, C_1')$$

$$X_3 = (*, *, *, *, *, *, C_1', C_2)$$

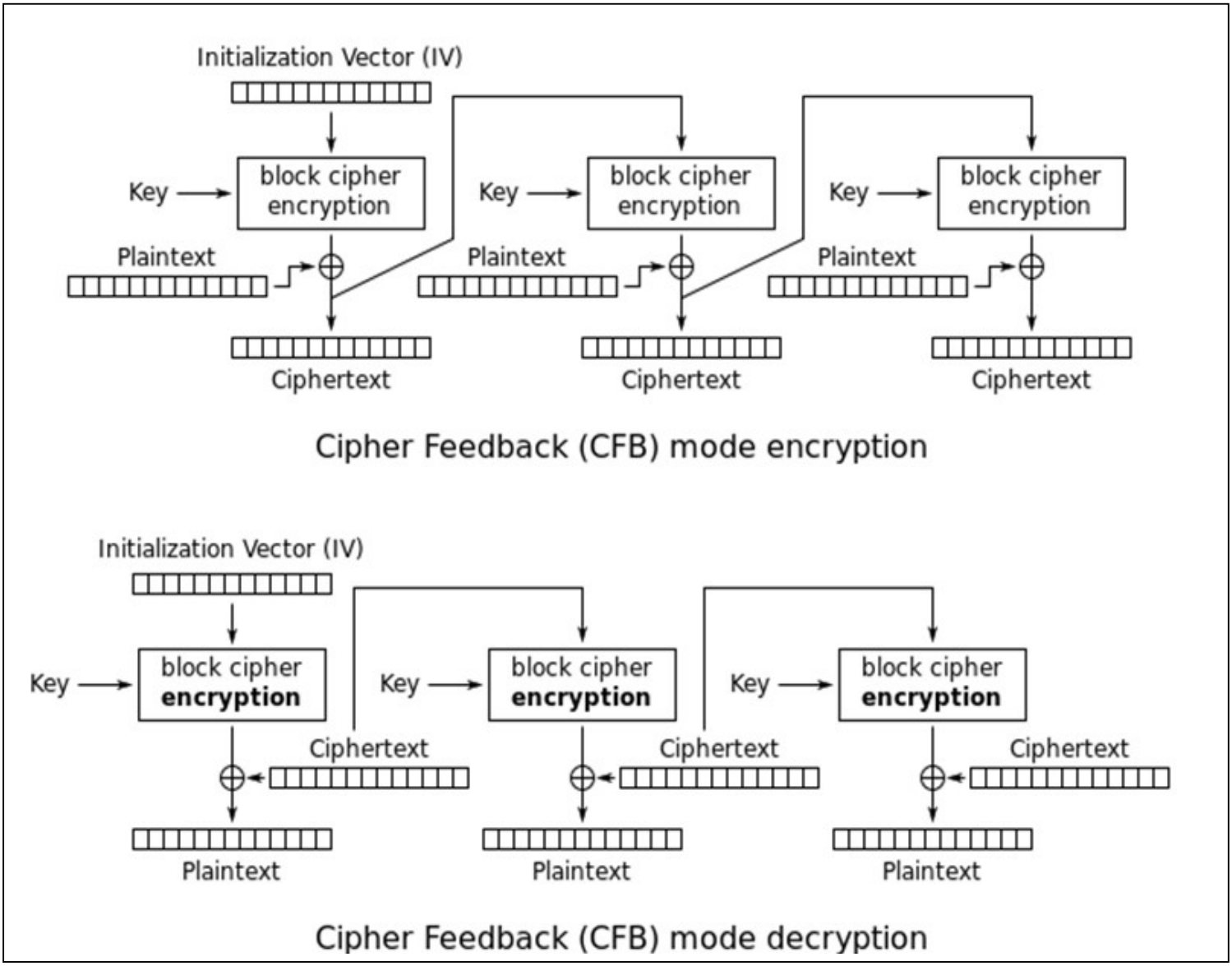
$$X_4 = (*, *, *, *, *, C_1', C_2, C_3)$$

$$\ldots$$

$$X_9 = (C_1', C_2, C_3, C_4, C_5, C_6, C_7, C_8)$$

$$X_{10} = (C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9)$$

使用密钥 $X_1, X_2, X_3, \ldots, X_9$ 解密 $C_1'C_2C_3C_4C_5C_6C_7C_8C_9$ 还原得到的 $P_1, P_2, P_3, \ldots, P_9$ 全部有错，但是从 P_{10} 开始的解密还原是正确的。



4.2 流密码算法 RC4

RC4 源代码分析：

http://10.71.45.100/bhh/MyRc4.c

第 5 章 DES 算法和 AES 算法

5.1 DES 算法

1973 年，NBS(National Bureau of Standards)公开征集一个希望成为国家标准的加密算法。1974 年，IBM 公司提交了一个叫做 LUCIFER 的算法。NBS 把该算法交给 National Security Agency 评估，后者对此算法做了一些修改。1975 年，NBS 公布了经 NSA 修改过的 LUCIFER 算法，称为 DES 算法。

DES: Data Encryption Standard

明文是 64 位=8 字节；

密钥是 64 位=8 字节；

加密与解密的密钥相同；

1．密钥太短

2．差分分析 可以攻击 des 算法

3． sbox 没有公开，怀疑有后门

5.1.1 DES 算法流程图

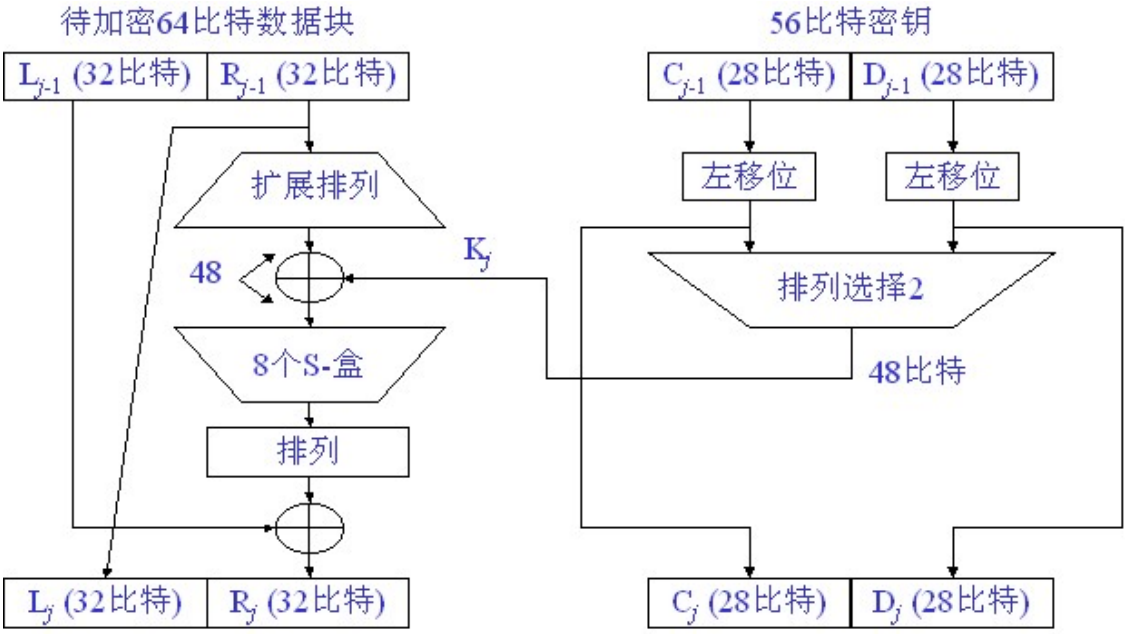


图2.6 DES算法每轮处理过程

(1) 64 位明文在进入加密前有一个打乱的过程，此时要用到一张打乱表(static char ip[64])，把 64 位的顺序打乱，例如：

ip[0]=58 表示源数据中的第 58 位(实际是第 58-1=57 位)要转化成目标数据中的第 0+1=1 位，其中下标代表的是以 0 为基数的目标位，而数组的元素值代表的是以 1 为基数的源位；

(2) 64 位明文经过加密变成密文后还要有一个打乱的过程，此时要用到的打乱表为 static char fp[64];

(3) 8 个 sbox 转化出来 32 位结果也需要打乱，这张打乱表为

static char sbox_perm_table[32];

(4) 64 位密钥要砍掉 8 位变成 56 位，此时要用到一张表：

static char key_perm_table[56];

(5) 56 位密钥在循环左移后，要提取其中的 48 位，此时也要用到一张表：

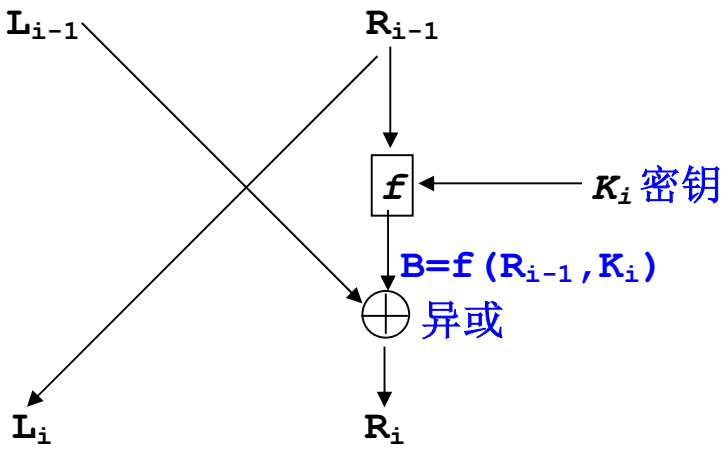
static char key_56bit_to_48bit_table[48];

64 位数据分成 4 位*16 组，输入的 4 位假定为第 1 位、2 位、3 位、4 位，输出的 4 位变成乱序比如为第 8 位、16 位、24 位、32 位。

a[16] [16] [8]
组号 每组值 输出的 4 位
0 的变化 分散在 64 位中
有 16 种 00000001 00000001 00000001 0000000
1110 00000000 ...

输出:b[0] b[1] b[2] b[3] b[4] b[5] b[6] b[7]
0000 0001
假定源 58 位=1 转化成目标第 7 位=1
组号=7/4=1
组内位号=7%4=3

$B = f(R_{i-1}, K_i)$
L_{i-1} xor $B = R_i$ 加密过程
B xor $R_i = L_{i-1}$ 解密过程



$R_i = L_{i-1}$ xor $f(R_{i-1}, K_i)$

S-盒：是一个替换的过程

5.1.2 调用 Openssl 库实现 DES 加密及解密

[demo] 调用 Openssl 库实现 DES 加密

```
#include <stdio.h>
#include <string.h>
#include <openssl/des.h>
#pragma comment(lib, "libeay32.lib")
#pragma comment(lib, "ssleay32.lib")
main()
{
    unsigned char bufin[100], bufout[100], *pin, *pout;
    char hex[100];
    int i, n, blocks;
    unsigned char k[]="ABCD1234";
    des_key_schedule ks;
    des_set_key((DES_cblock *)k, ks);
    gets(bufin);
    n = strlen(bufin);
    blocks = (n+7) / 8;
    pin = bufin;
    pout = bufout;
    for(i=0; i<n/8; i++)
    {
        des_ecb_encrypt((DES_cblock*)pin, (DES_cblock*)pout, ks,
                        DES_ENCRYPT);  // 加密
        pin += 8;
        pout += 8;
    }
    if(n%8 != 0)
    {
        memset(pin+n%8, 0, 8-n%8);
        des_ecb_encrypt((DES_cblock*)pin, (DES_cblock*)pout, ks,
                        DES_ENCRYPT);  // 加密
    }
    pin += n%8;
    pout += 8;
```

```
    pout[0] = '\0';
    for(i=0; i<blocks*8; i++)  // 转化成 16 进制字符串
    {
        sprintf(hex+i*2, "%02X", bufout[i]);
    }
    puts(hex);
}
```

[demo] 调用 Openssl 库实现 DES 解密

```
#include <stdio.h>
#include <string.h>
#include <openssl/des.h>
#pragma comment(lib, "libeay32.lib")
#pragma comment(lib, "ssleay32.lib")
main()
{
    unsigned char bufin[100], bufout[100], *pin, *pout;
    char hex[100];
    int i, n, blocks;
    unsigned char k[]="ABCD1234";
    des_key_schedule ks;
    des_set_key((DES_cblock *)k, ks);
    gets(hex);
    n = strlen(hex)/2;
    for(i=0; i<n; i++)
    {
      sscanf(&hex[i*2], "%02X", &bufin[i]);
    }
```

```
    blocks = n/8;
    pin = bufin;
    pout = bufout;
    for(i=0; i<blocks; i++)
    {
        des_ecb_encrypt((DES_cblock*)pin, (DES_cblock*)pout, ks,
                        DES_DECRYPT);  // 解密
        pin += 8;
        pout += 8;
    }
    pout[0] = '\0';
    puts(bufout);
}
```

5.1.3 DES 源代码分析

http://10.71.45.100/bhh/MyDes.c

void des_set_key(char *key)

(1) 根据 key_perm_table 从 8 字节的 key 中选择 56 位存放到 pc1m，每一位保存为一个字节。

(2) 根据 key_rol_steps 对 pc1m 左边 28 个元素及右边 28 个元素分别进行循环左移，移位以后的结果保存到 pcr 中。

(3) 根据 key_56bit_to_48bit_table 从 pcr 中选出 48 个元素，每 6 个元素靠右对齐合并成 1 个字节，保存到 kn 中。

void sbox_output_perm_table_init(void)

(1) 根据 sbox_perm_table 生成一张反查表 sbox_perm_table_inverse

(2) 根据 sbox 生成 sbox_output_perm_table：进入 sbox 的 6 位数据出来后变成 4 位，再打散并保存到 sbox_output_perm_table 中的一个 32 位元素内。

perm_init(char perm[16][16][8], char p[64])

(1) p 定义的是一张 64 位打乱表，下标为目标位，元素值为源位。
(2) 假定 X 是一个任意的 64 位数，现把它的 64 位按从左到右的顺序划分成 16 组，每组 4 位。
(3) 设 j 是第 i 组中的 4 位，显然 j 的值一共有 16 种变化，现通过查表 p 得到 j 中每个位分别落在 64 位中的哪一位，于是就把 j 中的 4 位打散并保存到 perm[i][j]的 8 个字节内。

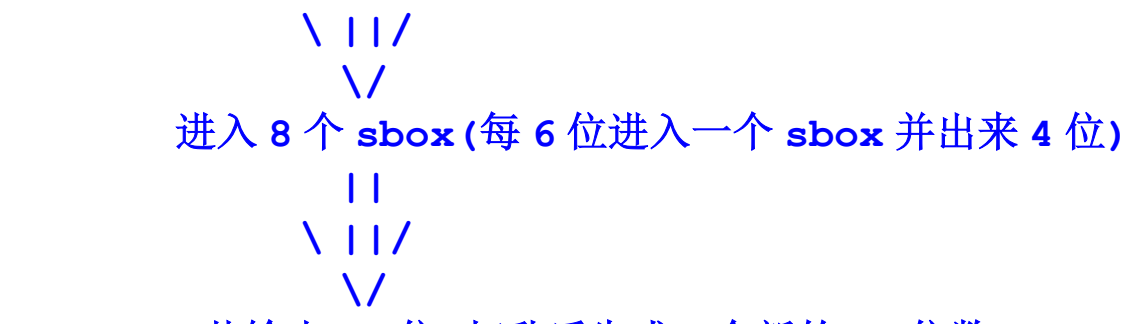
```
permute(char *inblock, char perm[16][16][8], char *outblock)
```

(1) 把 inblock 中的 8 字节划分成 16 组，每组 4 位。
(2) 设 j 是第 i 组中的 4 位，查 perm[i][j]得 8 字节即 64 位。
(3) 把每组查 perm 所得的 64 位求或,结果保存到 outblock 中。

```
long f(unsigned long r, unsigned char subkey[8])
```

(1) 根据 plaintext_32bit_expanded_to_48bit_table 把 r 扩展成 48 位，并把这 48 位划分成 8 组，每组 6 位。
(2) 把这 8 组数按顺序分别与 subkey 中包含的 8 组数做异或运算
(3) 异或后仍旧得到 8 组数，每组 6 位。
在 sbox_output_perm_table 中按顺序查这 8 组数，每组数 6 位进去，出来一个包含有打散 4 位的 32 位数，把 8 个 32 位数求或即得 f()的返回值。

64 位明文分成 L32、R32　　64 位 key 选 56 位,砍 8 位
56 位 key 分成 L28、R28
L28 及 R28 循环左移 n 次(1<=n<=2)
56 位 key 中选取 48 位
R32 展开成 48 位 xor 48 位 key
||



5.1.4 DES 分组加密模式:ecb, cbc, cfb

1. ecb

参见 5.1.2

2. cbc

需要一个 8 字节的初始向量(initialization vector)或初始变量(starting value)。令 iv=这 8 个字节。现要加密 block[0], block[1]:

```
c[0] = des_encrypt(block[0] ^ iv);
c[1] = des_encrypt(block[1] ^ c[0]);
```

c[0]和 c[1]就是密文。

3. cfb

同样需要一个 8 字节的初始向量。令 iv=这 8 个字节。现在要加密 b(共 8 字节):

```
v = des_encrypt(iv); /* v 包含 8 字节 */
c[0] = b[0] ^ v[0]; /* c[0]是 1 字节密文 */
iv = iv << 8 | c[0]; /*iv 左移 1 字节,末尾补 c[0]*/
v = des_encrypt(iv);
c[1] = b[1] ^ v[0]; /* c[1]是 1 字节密文 */
```

4. des_cbc_encrypt()及 des_cfb_encrypt()源代码

http://10.71.45.100/bhh/MyDesCbc.c

5．调用 Openssl 库函数 DES_ncbc_encrypt(), DES_cfb_encrypt()

(1) DES_ncbc_encrypt()

加密：

```
unsigned char bufin[100]={0}, bufout[100]={0};
char hex[100]={0};
int i, n, crypt_len;
DES_cblock iv = {0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF};
unsigned char k[9]="ABCD1234";
des_key_schedule ks;
des_set_key((DES_cblock *)k, ks);
gets(bufin);
n = strlen(bufin);
DES_ncbc_encrypt(bufin, bufout, n,
     &ks, &iv, DES_ENCRYPT);
crypt_len = (n+7)/8*8;
bufout[crypt_len] = '\0';
for(i=0; i<crypt_len; i++)  // 转化成 16 进制字符串
{
   sprintf(hex+i*2, "%02X", bufout[i]);
}
puts(hex);
```

解密：

```
unsigned char bufin[100], bufout[100];
char hex[100];
int i, n;
DES_cblock iv = {0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF};
unsigned char k[9]="ABCD1234";
des_key_schedule ks;
des_set_key((DES_cblock *)k, ks);
gets(hex);
n = strlen(hex)/2;
for(i=0; i<n; i++)
{
   sscanf(&hex[i*2], "%02X", &bufin[i]);
}
DES_ncbc_encrypt(bufin, bufout, n,
     &ks, &iv, DES_DECRYPT);
bufout[n] = '\0';
puts(bufout);
```

(2) DES_cfb_encrypt()

加密：

```
unsigned char bufin[100]={0}, bufout[100]={0};
char hex[100]={0};
int i, n, crypt_len, bits=8, bytes;
DES_cblock iv = {0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF};
unsigned char k[9]="ABCD1234";
des_key_schedule ks;
des_set_key((DES_cblock *)k, ks);
gets(bufin);
n = strlen(bufin);
DES_cfb_encrypt(bufin, bufout, bits, n,
     &ks, &iv, DES_ENCRYPT);
bytes = (bits+7)/8;
crypt_len = n/bytes*bytes;
bufout[crypt_len] = '\0';
for(i=0; i<crypt_len; i++)  // 转化成 16 进制字符串
{
   sprintf(hex+i*2, "%02X", bufout[i]);
}
puts(hex);
```

解密：

```
unsigned char bufin[100], bufout[100];
char hex[100];
int i, n, crypt_len, bits=8, bytes;
DES_cblock iv = {0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF};
unsigned char k[9]="ABCD1234";
des_key_schedule ks;
des_set_key((DES_cblock *)k, ks);
gets(hex);
n = strlen(hex)/2;
for(i=0; i<n; i++)
{
   sscanf(&hex[i*2], "%02X", &bufin[i]);
}
DES_cfb_encrypt(bufin, bufout, bits, n,
     &ks, &iv, DES_DECRYPT);
bytes = (bits+7)/8;
crypt_len = n/bytes*bytes;
bufout[crypt_len] = '\0';
```

```
puts(bufout);
```

5.1.5 ecb 及 cbc 模式中的 ciphertext stealing

参考链接： https://en.wikipedia.org/wiki/Ciphertext_stealing

1. ecb

12345678 12345

~~12345678~~

12345~~678~~ ~~12345~~

12345~~678~~ ~~12345~~

2. cbc

IV

12345678 12345,0,0,0

~~12345678~~

12345~~678~~ ~~12345,~~0,0,0 ;

12345~~678~~ ~~12345~~

meet in the middle attack

5.1.6 三重 DES

$c = E(D(E(p,k_1), k_2), k_3);$

$p = D(E(D(c,k_3), k_2), k_1);$

5.2 AES 算法

AES: Advanced Encryption Standard

1997 年 NIST(National Institute of Standards and Technology)公开征集新的加密标准以代替 DES 算法。

1998 年，NIST 从收到的 21 个算法中挑选出 15 算法;1999 年又从 15 个算法中挑选了 5 个算法：MARS，RC6，Rijndael，Serpent，Twofish；最终 Rijndael 算法获胜，成为 AES。

Rijndael 算法的作者是 Joan Daemen 和 Vincent Rijmen。

Rijndael 算法与 AES 算法有一定区别，前者允许明文块

长度可变，后者不可。

AES 算法的明文长度=128 位(16 字节)，密文长度=16 字节；

密钥长度分成三种：

128 位(16 字节)，192 位(24 字节)，256 位(32 字节)

5.2.1 AES 算法流程

以 128 位(16 字节)密钥为例，设 p 为明文，k 为密钥，则 AES 的加密过程如下：

```
unsigned char a[4] = {0x03, 0x01, 0x01, 0x02};
AddRoundKey(p, k);
for(i=1; i<=10; i++)
{
   ByteSub(p, 16); /* p[i] = sbox[p[i]]; */
   ShiftRow(p);
   if(i != 10)
      MixColumn(p, a, 1); /* do mul */
   else
      MixColumn(p, a, 0); /* don't mul */
   AddRoundKey(p, k+i*(4*4));
}
```

ShiftRow：对明文 16 字节构成的 4*4 矩阵逐行做循环左移

0 1 2 3；不移动　　0 1 2 3

4 5 6 7；循环左移 1 次　5 6 7 4

8 9 A B；循环左移 2 次　A B 8 9

C D E F；循环左移 3 次　F C D E

sbox 的生成步骤：

$sbox[a] = ((a^{-1} * 0x1F) \bmod (X^8 + 1)) \wedge 0x63;$

求 a^{-1} 可以按照教材 p.93 的算法(扩展欧几里德算法)来做；

$a^{-1} * 0x1F \bmod 0x101$ 可以用农夫算法；

查教材 p.67 sbox 表，得 sbox[4]=F2，此值的计算步骤如下：

4^{-1} = CB （AesDemo 程序中输入 x=4，点 x^{-1} mod 0x11B）

CB * 1F = 91 mod 0x101 （AesDemo 中，x=CB，y=1F，点 x*y mod 0x101）

91 ^ 63 = 1001 0001
0110 0011 ^）
= 1111 0010 = F2

5.2.2 MixColumn 涉及的数学基础

1．有限域

域 F 有时记作{F,+,*}，是有两个运算的集合，这两个运算分别称为加法和乘法；

本质上说，域(field)就是一个集合，我们可以在其上做加减乘除运算而不脱离该集合；

有限域中元素的个数称为有限域的阶(order)；

有限域的阶必为素数 p 的幂，即 p^n，其中 n 为正整数；

对任意素数 p 和正整数 n，存在 p^n 阶的有限域，记为 GF(p^n)。当 n=1 时，有限域 GF(p)也称素域；

2．有限域 GF(p) Galois 伽罗瓦 Field

给定一个素数 p，元素个数为 p 的有限域 GF(p)定义为整数{0,1,2,...,p-1}的集合 Z_p，其运算为 mod p 的算术运算。

最简单的有限域是 GF(2)，该有限域的元素的个数为 2，它们分别是 0 和 1，在 GF(2)上的加法运算等价于异或运算，乘法运算等价于二进制与运算。

3．有限域 GF(2^8)

(1)对于 F[x]中的每个不可约多项式 p(x)，可以构造一

个域 $F[x]_{p(x)}$；

(2)设 $p(x)$ 是 $F[x]$ 中 n 次不可约多项式，令 $F[x]_{p(x)}$ 为 $F[x]$ 中所有次数小于 n 的多项式集合，即：

$$F[x]_{p(x)} = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \dots + a_1x + a_0$$

其中 $a_i \in F$，即在集合{0,1,2,…,p-1}上取值，则我们可以定义 $F[x]_{p(x)}$ 上的加法运算及乘法运算如下：

$$a(x)+b(x) = (a(x)+b(x)) \bmod p(x)$$

$$a(x)*b(x) = (a(x)*b(x)) \bmod p(x)$$

(3)在 $GF(2^n)$ 中，$F[x]_{p(x)}$ 中所有次数小于 n 的多项式可以表示为：

$$f(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \dots + a_1x + a_0$$

其中系数 a_i 是二进制数，该多项式可以由它的 n 个二进制系数唯一表示。因此 $GF(2^n)$ 中的每个多项式都可以表示成一个 n 位的二进制数；

(4)AES 算法选择用多项式表示有限域 $GF(2^8)$ 中的元素，其中不可约多项式 $p(x)$ 为 $x^8+x^4+x^3+x+1$，该多项式对应的二进制数为 1 0001 1011，对应的十六进制数为 0x11B。AES 算法的 MixColumn 步骤中对两个 8 位数做乘法运算时必须 mod 0x11B。

$$x^6 \% (x^4+1) = ((x^4+1)x^2 - x^2) \% (x^4+1) = -x^2 = x^2 \bmod (x^4+1)$$

(5)AES 算法的 MixColumn 步骤中还涉及 3 次多项式的乘法，多项式乘法运算必须 mod (X^4+1)，目的是使得乘积仍旧为一个 3 次多项式。AES 加密及解密分别使用以下两个多项式作为被乘数：

① $3x^3 + x^2 + x + 2$

② $0x0B*x^3 + 0x0D*x^2 + 0x09*x + 0x0E$

5.2.3 MixColumn 中的 8 位数乘法运算

1. 用农夫算法计算 8 位数乘法 mod 0x11B

农夫算法链接：https://en.wikipedia.org/wiki/Rijndael_Galois_field

设 x=1000 1000, y=0000 0101, p=0,
现要计算 x*y mod 0x11B:

```
for(i=0; i<8; i++)
{
  x = x << 1;
  y = y >> 1;
```

若发现 y 右移后丢失一个 1： y=0000 0101 ➔ y = 0000 0010
则做 p = p ^ x; 相当于 p=p+x, 其中 x 是左移前的值
若发现 x 左移后产生进位 1： x=1000 1000 ➔ x = 1 0001 0000
则做 x = x ^ 0x11B; 相当于 x=x-0x11B

```
  即 x =  1 0001 0000     ; 被减数是 x 左移后的值
          1 0001 1011 ^)  ; 异或相当于做减法
-------------------------------------------
          0 0000 1011

}
```

上述过程重复 8 次，即 x 左移 8 次，y 右移 8 次后，
p 的最终值就是 x*y = 0x9E mod 0x11B

2．8 位数乘法实质上是多项式乘法 mod $(x^8+x^4+x^3+x+1)$
例如：求 1000 1000 * 0000 0101 mod 0x11B
可以把上述两数乘法转化成两个多项式相乘：

$(x^7 + x^3) * (x^2 + 1) =$
$x^9 + x^7 + x^5 + x^3 =$
$x^7 + x^4 + x^3 + x^2 + x \mod (x^8 + x^4 + x^3 + x + 1)$
再用手工除法求模：

x

$x^8+x^4+x^3+x+1$ $x^9 + x^7 + x^5 + x^3$

$x^9 + x^5 + x^4 + x^2 + x$

$x^7 + x^4 + x^3 + x^2 + x$

把余式转化成二进制就是：

1001 1110

结论：

0x88 * 0x05 = 0x9E mod 0x11B

3．用农夫算法分步计算 0x05 * 0x43 mod 0x11B

```
a=0000 0101, b=0100 0011, p=0
a=0000 0101, b=0100 0010, p=0000 0101
a=0000 1010, b=0010 0001, p=0000 0101; ①
a=0000 1010, b=0010 0000, p=0000 0101 ^
                            0000 1010
                           =0000 1111
a=0001 0100, b=0001 0000, p=0000 1111; ②
a=0010 1000, b=0000 1000, p=0000 1111; ③
a=0101 0000, b=0000 0100, p=0000 1111; ④
a=1010 0000, b=0000 0010, p=0000 1111; ⑤
=10100 0000, b=0000 0001, p=0000 1111; ⑥
^10001 1011
 =0101 1011, b=0000 0001, p=0000 1111
a=0101 1011, b=0000 0001, p=0000 1111
a=0101 1011, b=0000 0000, p=0000 1111 ^
                            0101 1011
                           =0101 0100
a=1011 0110, b=0000 0000, p=0101 0100; ⑦
=10110 1100, b=0000 0000, p=0101 0100; ⑧
^10001 1011
 =0111 0111
a=0111 0111, b=0000 0000, p=0101 0100
```

结论：0x05 * 0x43 = 0x54 mod ($x^8+x^4+x^3+x+1$)

关于 x 左移进位后为什么要做 x = x ^ 0x11B 的进一步说明：

x*y+p mod 0x11B

设 x=1000 1000，在 x=x<<1 后

x=1 0001 0000

现把 x 分解成 1 0001 1011+0000 1011 之和(+其实是异或)

(1 0001 1011 + 0000 1011)*y+p mod 0x11B

=0x11B*y + 0x0B*y + p mod 0x11B

=0x0B*y + p mod 0x11B

这里消去划线这一项，其实就是做了 x*y mod 0x11B 的除法求余运算。

5.2.4 MixColumn 中的 3 次多项式乘法运算

char plain[16]={4,3,2,1,11,22,33,44,…}

$(3x^3 + x^2 + x + 2) * (x^3 + 2x^2 + 3x + 4) \bmod (x^4 + 1)$

= 低次系数

$$\begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} * \begin{pmatrix} 4 \\ 3 \\ 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 14 \\ 5 \\ 0 \\ 15 \end{pmatrix}$$

高次系数

$x^3(2*1 + 1*2 + 1*3 + 3*4)$

$x^2(2*2 + 1*3 + 1*4 + 3*1)$

$x^1(2*3 + 1*4 + 1*1 + 3*2)$

$x^0(2*4 + 1*1 + 1*2 + 3*3)$

注意 3 次多项式系数的乘法是指 8 位数乘法 mod 0x11B，而加法是指异或，以 x^0 系数为例：

2*4 + 1*1 + 1*2 + 3*3 = 0x08 ^ 0x01 ^ 0x02 ^ 0x05

= 0x09 ^ 0x02 ^ 0x05 = 0x0B ^ 0x05 = 0x0E

mod x^4+1 的作用是把大于等于 4 次的项转成小于等于 3 次：

例如： x^6 mod x^4+1

$x^6 = (x^4+1)*x^2-x^2 = -x^2 = x^2 \bmod x^4+1$

加密时，MixColumn 多项式乘法中的被乘数是{3,1,1,2}，而解密时的被乘数是{0x0B, 0x0D, 0x09, 0x0E}，因为：

$0xB*x^3 + 0xD*x^2 + 9x + 0x0E$ 是 $3x^3 + x^2 + x + 2$ 的逆。

5.2.5 MixColumn 的具体步骤

```
void MixColumn(unsigned char *p, unsigned char a[4], int do_mul);
```

(1) 对 p 指向的 4*4 矩阵 m 中的 4 列做乘法运算；

(2) 这里的乘法是指有限域 GF(2^8)多项式模(X^4+1)乘法，具体步骤请参考 5.2.4 或教材 p.61 及 p.62；

(3) aes 加密时采用的被乘数 a 为多项式 3*X^3 + X^2 + X + 2，用数组表示为 unsigned char a[4]={0x03, 0x01, 0x01, 0x02}；

(4) aes 解密时采用的被乘数 a 为加密所用多项式的逆，即{0x0B, 0x0D, 0x09, 0x0E}；

(5) 矩阵 m 中的 4 列按以下顺序分别与 a 做乘法运算：

第 0 列：由 m[0][0], m[1][0], m[2][0], m[3][0]

第 3 列：由 m[1][3], m[2][3], m[3][3], m[0][3]

第 2 列：由 m[2][2], m[3][2], m[0][2], m[1][2]

第 1 列：由 m[3][1], m[0][1], m[1][1], m[2][1]

注：明文若按纵向排列，则与 a 做乘法的各列顺序为第 0、1、2、3 列，且每列不需要向上做循环移位。明文纵向排列与横向排列的对比见以下链接插图：

http://10.71.45.100/bhh/MixColumn.bmp

(6) 乘法所得 4 列转成 4 行，保存到 p 中，替换掉 p 中原有的矩阵；

(7) do_mul 用来控制是否要做乘法运算，加密最后一轮及解密第一轮 do_mul=0；

5.2.3 AES 密钥的生成过程

以 128 位种子密钥为例，假定当前机器的存储模式为 Big-Endian(高字节在前，低字节在后)，设 long k[4]是种子密钥，则后面还需要生成 k[4]至 k[43]共 40 个 long，步骤如下：

k[4] = k[3];

把 k[4]包含的 4 个字节循环左移 1 字节；

ByteSub(&k[4], 4)； 把 k[4]包含的 4 个字节全部替换成 sbox 中的值

$r = 2^{(i-4)/4} \bmod 0x11B$； 计算轮常数 r，其中 i 是 k 的下标即 4

k[4]首字节 ^= r; k[4]首字节与 r 异或
k[4] ^= k[0]; k[4]与 k[0]异或
k[5] = k[4] ^ k[1];
k[6] = k[5] ^ k[2];
k[7] = k[6] ^ k[3];
以上过程生成了 4 个 long，是一组 16 字节的 key。接下去生成 k[8]至 k[11]的过程与 k[4]至 k[7]类似，其中 k[8]像 k[4]那样需要作特殊处理：
k[8] = k[7];
k[8]包含的 4 字节循环左移 1 字节；
ByteSub(&k[8], 4);
轮常数 r = 2^1 = 2 mod 0x11B;
k[8]首字节 ^= r;
k[8] ^= k[4];
k[9] = k[8] ^ k[5];
k[10] = k[9] ^ k[6];
k[11] = k[10] ^ k[7];
按上述步骤可以生成 10 组密钥，每组为 4 个 long。请特别注意最后两组的轮常数因为 mod 0x11B 的缘故与 2^i 的值并不相等。对应于 i∈[0,9]，生成 10 组密钥的轮常数 r=2^i mod 0x11B 的值分别为：
0x01 0x02 0x04 0x08 0x10 0x20 0x40 0x80 0x1B 0x36

192 位及 256 位种子密钥生成密钥的过程比 128 位复杂一些，具体请参考 MyAes.c 中的函数 aes_set_key()。

5.2.4 调用 Openssl 库中的 AES 加密解密函数

```
#include <openssl/aes.h>
#include <stdio.h>
#pragma comment(lib, "libeay32.lib")
#pragma comment(lib, "ssleay32.lib")
main()
{
  AES_KEY k;
```

```
    int i;
    unsigned char bufin[100]="A Quick BrownFox";
    unsigned char bufout[100]={0};
    unsigned char hex[100];
    AES_set_encrypt_key("0123456789ABCDEF", 128, &k);
    AES_encrypt(bufin, bufout, &k);
    for(i=0; i<16; i++)
    {
       sprintf(hex+i*2, "%02X", bufout[i]);
    }
    puts(hex);
    memset(bufin, 0, sizeof(bufin));
    memset(bufout, 0, sizeof(bufout));
    for(i=0; i<16; i++)
    {
       sscanf(hex+i*2, "%02X", &bufin[i]);
    }
    AES_decrypt(bufin, bufout, &k);
    puts(bufout);
}
```

第 6 章 RSA 算法

6.1 RSA 算法简介

DES 及 AES 属于对称密码体制(symmetric cryptosystem)，加密及解密使用同一密钥。

RSA算法是1977年由Ron Rivest, Adi Shamir, Leonard Adleman提出的公钥密码体制(public-key cryptosystem)。

公钥密码体制也称非对称密码体制(asymmetric cryptosystem)。

在公钥密码体制中，加密密钥与解密密钥是不同的，加密密钥简称公钥(public key)，解密密钥简称私钥(private key)。

首先，选取两个大素数：p 和 q，计算乘积：n=p*q
其中 n 公开，p、q 保密。
然后随机选取加密密钥 e，使 e 和(p-1)*(q-1)互素。
接着要找出 d，使得：

$$e*d = 1 \bmod ((p-1)*(q-1))$$

加密与解密时都没有用到：p、q、(p-1)*(q-1)
只要 n 足够大，比如达到 1024 位，即 2^{1024}，则在短时间内无法把 n 分解成 p、q 的乘积。

按下面的公式进行加密：

$$c=m^e \pmod n$$

按下面的公式进行解密：

$$m=c^d \pmod n$$

具体流程：

.	随机选择两个不等素数 p 和 q。
.	计算出 n=p*q。
.	选择一个数 e 使它和(p-1)(q-1)互素。
.	计算 e 在模(p-1)(q-1)的逆元 d。
.	公开 (e,n)作为 RSA 公钥。
.	保留 (d,n)作为 RSA 私钥。

举例说明其工作过程：取两个素数
p=11，q=13，n = p*q = 11*13 = 143，
z=(p-1)*(q-1)=(11-1)*(13-1)= 120，
再选取与 z=120 互素的整数 e，如 e=7，现可计算出满足
7*d = 1 mod 120
的整数 d=103(私钥)，即：7*103=1 mod 120

整理如下：

p=11，q=13，n=143，

e=7，d=103，

(n,e)=(143,7)，

(n,d)=(143,103)

以数据加密为例：

A 向 B 发送机密数据信息 m=85，并已知 B 的公钥(n,e)=(143,7)，于是可计算出密文：

$$c = m^e \bmod n = 85^7 \bmod 143 = 123$$

A 将 c 发送至 B，B 利用私钥(n,d)=(143,103)对 c 进行解密：

$$m = c^d \bmod n = 123^{103} \bmod 143 = 85$$

6.2 RSA 算法的数学基础

1. Euler 函数(p.86)

$\phi(n)$：小于n且与n互素的整数个数。

例如 $\phi(5) = 4$，因为与5互素的整数有：1，2，3，4

例如 $\phi(10) = 4$，因为与10互素的整数有：1，3，7，9

2. Euler定理(p.88)

若gcd(x,n)=1，则 $x^{\phi(n)} = 1 \pmod n$。

例如 $3^{\phi(5)} = 3^4 = 81 = 1 \pmod 5$

3. Fermat小定理(p.89)

设p为素数，且gcd(x,p)=1，则 $x^{p-1} = 1 \pmod p$。

因为当p为素数时，显然有 $\phi(p) = p-1$。

4. 中国剩余定理(Chinese Remainder Theorem)(p.83)

设 $m_1, m_2, m_3 \ldots, m_r$ 两两互素，则以下同余方程组

$$x \equiv a_i \pmod{m_i},\ i=1,2,3,\ldots r$$

模 $M=m_1m_2m_3\ldots m_r$ 的唯一解为

$$x = \sum_{i=1}^{r} a_i * M_i * (M_i^{-1} \bmod m_i) \bmod M$$

$i=1$

其中$M_i = M/m_i$

例如韩信点兵问题：

$x = 1 \bmod 5 \quad (1)$

$x = 5 \bmod 6 \quad (2)$

$x = 4 \bmod 7 \quad (3)$

$x = 10 \bmod 11 \quad (4)$

则$M = 5*6*7*11 = 2310,$

$M_1 = 6*7*11 = 462$

$M_2 = 5*7*11 = 385$

$M_3 = 5*6*11 = 330$

$M_4 = 5*6*7 = 210$

$M_1^{-1} \bmod m_1 = 3$

$M_2^{-1} \bmod m_2 = 1$

$M_3^{-1} \bmod m_3 = 1$

$M_4^{-1} \bmod m_4 = 1$

$$\sum a_i * M_i * (M_i^{-1} \bmod m_i) \bmod M$$

$=1*462*3 + 5*385*1 + 4*330*1 + 10*210*1 \bmod 2310$

$=2111$

5. Euler函数的乘法性质(p.86)

若n_1, n_2互素，则$\phi(n_1*n_2) = \phi(n_1)*\phi(n_2)$

例如：$\phi(3*5) = \phi(3)*\phi(5) = 2*4 = 8$

与15互素的数包括：1,2,4,7,8,11,13,14

6. Euler函数的乘积公式(p.87)

$$\phi(n) = n * \prod_{p|n} (1- 1/p)$$

例如：$\phi(10) = 10 * (1-1/2) * (1-1/5) = 4$

6.3 RSA 算法证明 (p.90)

方法 1:

设 m 是明文，c 是密文，$c=m^e \bmod n$。现证明 $m=c^d \bmod n$。

因为 $\phi(n)=\phi(p*q)=\phi(p)*\phi(q)=(p-1)(q-1)$，

又因为 $ed = 1 \bmod (p-1)(q-1)$，

所以一定可以找到一个 k 使得 $ed = 1 + k(p-1)(q-1)$ 成立，

于是 $c^d = m^{ed} = m^{1+k(p-1)(q-1)} = m * m^{k(p-1)(q-1)}$

$= m * (m^{\phi(n)})^k = m * (1)^k = m \bmod n$

为什么 $(m^{\phi(n)})^k = (1)^k \bmod n$?

$a = a' \bmod n$

$b = b' \bmod n$

则一定有 $a*b = a'* b' \bmod n$，这是因为:

$a = kn+a'$

$b = jn+b'$

$a*b=(kn+a')(jn+b') = kjnn + a'jn + b'kn + a'b'$

上述证明的前提是 gcd(m,n)=1。

当 $gcd(m,n) \neq 1$ 时，则一定有 gcd(m,n)=p 或 gcd(m,n)=q。现假设 gcd(m,n)=p，即 m 是 p 的倍数，则 m 与 q 一定互素。于是有:

$m^{\phi(q)} = 1 \bmod q \rightarrow m^{(q-1)} = 1 \bmod q \rightarrow$

$m^{(q-1)*k(p-1)} = 1 \bmod q \rightarrow m^{\phi(n)*k} = 1 \bmod q \rightarrow$

$m^{\phi(n)*k} = q*s + 1 \rightarrow m * m^{\phi(n)*k} = m*q*s + m \rightarrow$

$m^{\phi(n)*k+1} = cp * q * s + m \rightarrow m^{\phi(n)*k+1} = m \bmod n$

$\rightarrow m^{ed} = m \bmod n$

方法 2:

$ed = 1 \bmod \phi(p*q) \rightarrow$

$ed - 1 = k(p-1)(q-1) \rightarrow$

①设 $m = 0 \bmod p$，则 $m^{ed} = 0 = m \pmod p$

②设 m ≠ 0 mod p，则

$m^{ed} = m^{(ed-1)}*m = m^{k(p-1)(q-1)}*m$

$= (m^{(p-1)})^{k(q-1)} *m = (1)^{k(q-1)} *m = m \pmod p$

综合①②两种情况可得：

$m^{ed} = m \bmod p$ (a)

同理可证：

$m^{ed} = m \bmod q$ (b)

根据(a)(b)可得：

$m^{ed} - m = 0 \bmod p$

$m^{ed} - m = 0 \bmod q$

即 $m^{ed} - m$ 既可以被 p 整除，又可以被 q 整除，

而 p、q 是互素的，所以 $m^{ed} - m$ 一定同时包含因子 p 及 q，

于是有：

$m^{ed} - m = 0 \bmod p*q \rightarrow m^{ed} = m \bmod p*q$

上述结论其实可以由中国剩余定理得出：

$m^{ed} - m = 0*q*(q^{-1} \bmod p)+0*p*(p^{-1} \bmod q) \pmod{p*q}$

$= 0 \bmod p*q$

上述划波浪线的证明也可以换成以下方法：

$m^{ed} = m \bmod P$ (a)

$m^{ed} = m \bmod Q$ (b)

由(a)(b)可得：

$m^{ed} = k_1*P + m$

$m^{ed} = k_2*Q + m$

$\rightarrow k_1*P + m = k_2*Q + m \rightarrow$

$k_1 * P = k_2 * Q$

$\rightarrow$

$k_1 * P = 0 \bmod Q$

$k_2 * Q = 0 \bmod P$

由于 P、Q 互素，所以

$k_1 = a*Q, \quad K_2 = b*P$

➔

$m^{ed} = k_1*P + m$

$m^{ed} = k_2*Q + m$

$m^{ed} = a*Q*P + m = m \bmod (P*Q)$

或者

$m^{ed} = b*P*Q + m = m \bmod (P*Q)$

保密 p,q,(p-1)*(q-1)的前提下:

已知 e、n 的情况下，无法算出 d;

同理在 d、n 已知的情况下，也无法算出 e;

aes+rsa:

对文件加密用的是 128 位密钥的 aes 算法。

用 rsa 算法加密文件的话，速度太慢，故没有采纳。

abc.txt:

rsa(CG,aes_key)|aes(aes_key,"ABC123")

这里的 rsa 算法使用 2048 位，是在受害电脑上随机产生的，

设公钥为 CG，私钥为 CS，那么加密 aes 密钥用的是 CG。

病毒程序里面有一个固定的 rsa 公钥设为 HG,它对应的私钥

设为 HS 在程序里面是不存在的。

key=rsa(HG, CS)写在受害电脑的某个文件中。

同时 key 有可能传给黑客的服务器。

$c = m^e \bmod N$ 加密过程

$m = c^d \bmod N$ 解密过程

在加密时，要求 m < N

"ABCD" ➔ 0x41424344

6.4 调用 openssl 库函数

0. RSA_generate_key() 产生密钥

如 RSA_generate_key(256, 0x10001, NULL, NULL);

N 位数　公钥

1. RSA_public_encrypt()　公钥加密
2. RSA_private_decrypt() 私钥解密
3. RSA_private_encrypt() 私钥加密
4. RSA_public_decrypt()　公钥解密
5. RSA_new() 分配一个 RSA 结构指针
6. RSA_free() 释放一个 RSA 结构指针
7. BN_new() 分配一个大数指；BN:Big Number

BN 指 128 位、256 位、512 位、1024 位整数。

密钥 N 为 128 位即 16 字节的情况下，明文长度必须是 16 字节，并且明文的值一定要小于 N，密文长度也只能是 16 字节。

公钥加密：$m' = m^e \bmod n$

私钥解密：$m = (m')^d \bmod n$

共 12 个 0

设明文 char m[]={'A','B','C','D',0,0,…}

则实际上该明文是被当成如下这个大数来处理：

0x41424344000000000000000000000000

(1)软件打开时显示一个机器码

其中机器码 m'=rsa(mac, 公钥)

(2)软件作者：mac=rsa(m', 私钥)

注册码 sn=(mac, 私钥)

(3)软件验证注册码：

rsa(sn, 公钥)==mac

为什么加密时明文 m 的位数应该与 N 相同？

假定 m 很小，如 $m^e < N$，则解密时不需要用到 d，

只要对 m 开 e 次方即可。

side-channel attack 旁路攻击
blinding 模式是为了对抗 timing attack(计时攻击).
P.95 大数幂乘的快速算法
私钥加密时本来是:$c = m^d \bmod n$ ➔ 转换成以下两步
① $m' = m^d * r^e \bmod n$;
② $c = m' * (r^e)^{-1} \bmod n$;
其中 r 是一个随机数

6.5 数字签名
假定以下是 A 的公钥及私钥:
n=F03E1D9F9BFB6827B4A49AED686F7790868CA58FA7B
A7110C29D3241A8E2EF53
p=F9298817691C971281C3CD6D203C28BF
q=F6D5EB95C1915957FB00604580B9EA6D
d=08F1C718922E220A9867287D7E4DE81D9EED52D623E
1BB48758146F22C515D41
e=010001

假定 A 要发一封信给 B:
信的内容 L="Hello, I'm A."
如何对信进行加密?
L' = RSA(L, B 的公钥)
A 把 L'发给 B, B 收到后如何解密?
L = RSA(L', B 的私钥)

A 如何对信件进行签名?

首先对信的内容计算摘要(digest),这里采用 MD5 算法:
M = MD5(L) =
82228599d3e30286a22e7d170ebe2546
用 A 的私钥对 M 进行签名(实际上是用 A 的私钥对 M 加密):
M' = RSA(M, A 的私钥) =
6EFADDDBBEC3F995F4F4398F2A6049BF42A41645B36B1
A2E1AE6C52BFE11950B
此时,M'就是 A 对信件摘要 M 的签名。
假定 A 把 L 及 M'都发送给 B。
B 如何对 A 的签名 M'进行验证?
首先要用 A 的公钥对 M'进行解密:
m = RSA(M', A 的公钥) =
82228599d3e30286a22e7d170ebe2546
最后还要判断 m 是否正确:
若 MD5(L) = m,则证明此信确实是 A 所发。

6.6 RSA 算法的安全性

rsatool 可以快速分解 128 位的 N
筛法
1024 位的 N 是相对安全的
填充

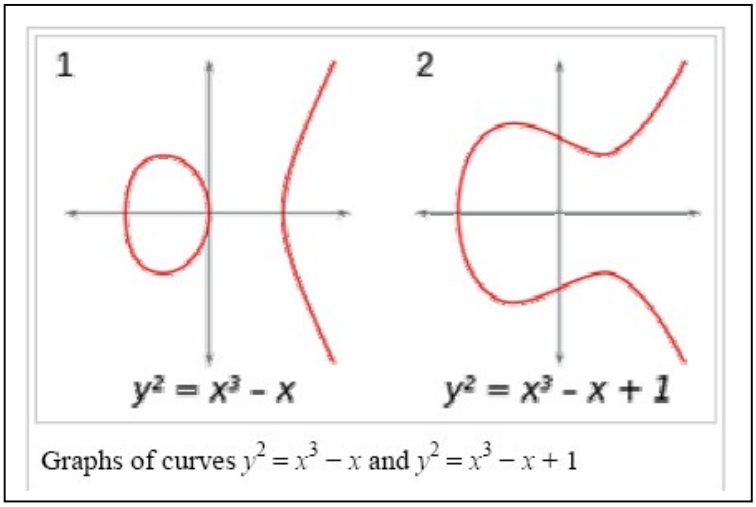
第 7 章 椭圆曲线(Elliptic Curve)算法

7.1 ECC 算法简介

椭圆曲线可以定义成所有满足方程
$E: y^2 = x^3 + ax + b$
的点(x,y)所构成的集合。
若 $x^3 + ax + b$ 没有重复的因式或 $4a^3+27b^2 \neq 0$(称为判别

式),则

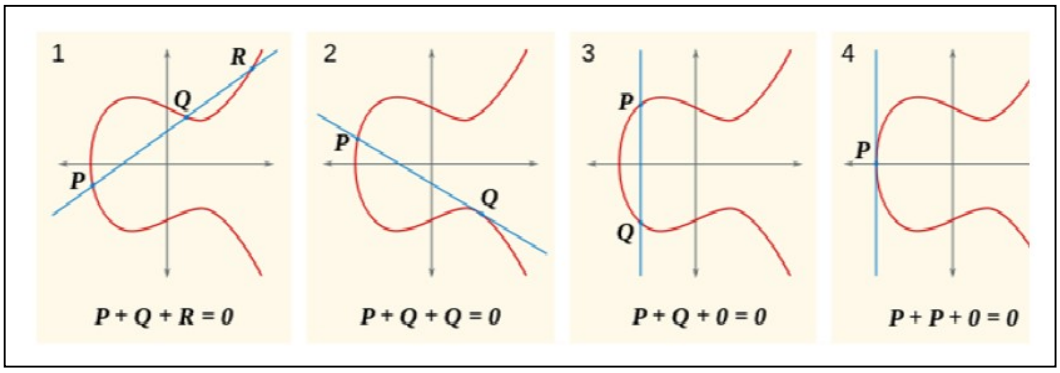
E: $y^2 = x^3 + ax + b$ 能定义成为一个群。



Graphs of curves $y^2 = x^3 - x$ and $y^2 = x^3 - x + 1$

7.2 ECC 算法的数学基础

1. 椭圆曲线在素域 Z_p 上的运算规则



(1) P+O=O+P=P

(2) 如果 **$P=(x_1,y_1)$, $Q=(x_2, y_2)$,** 且有 **$x_1=x_2$** 及 **$y_1=y_2=0$,** 或有 **$x_1=x_2$** 及 **$y_1=-y_2\neq 0$,** 则 **P+Q=O;**

(3) 如果 **$P=(x_1,y_1)$, $Q=(x_2, y_2)$,** 且排除 **(1)(2),** 则 **$P+Q=(x_3,y_3)$** 由下列规则决定:

$$x_3 = \lambda^2 - x_1 - x_2$$

$$y_3 = \lambda(x_1-x_3) - y_1$$

当 **$P\neq Q$** 时, $\lambda=(y_2-y_1)/(x_2-x_1)$;

当 P=Q 时， $\lambda=(3x_1^2+a)/(2y_1)$；

2. Euler 准则(p.98)

$y^2 = x \bmod p$

设 p>2 是一个素数， x 是一个整数， gcd(x,p)=1， 则

(1) x 是模 p 的平方剩余当且仅当

$x^{(p-1)/2} \equiv 1 \pmod p$

(2) x 是模 p 的平方非剩余当且仅当

$x^{(p-1)/2} \equiv -1 \pmod p$

3. 点加运算举例(p.127)

椭圆曲线 $y^2 = x^3 + x + 6 \pmod{11}$ 上的点。

a=1 b=6 p=11

基点 G G 的阶 余因子

以上 6 项决定一条椭圆曲线。

其中余因子=曲线的阶即曲线上点的个数/G 的阶,此值通常=1

可以把 α 称为生成元(generator)， 也称作基点(base point)，假定 nα=0，则 n 称为 α 的阶(order)。

曲线的阶是曲线上点的个数。

曲线上的点(x,y)一定满足条件 0<= x,y <p，并且 x,y 一定是整数。

Q = k*P 其中 k<n

已知 P 及 Q 的情况下，求 k 很困难。

设 α=(2,7)， 计算 2α=α+α：

$\lambda=(3x_1^2+a)/(2y_1)=(3*2^2+1)/(2*7)=$

$13/14=13*14^{-1}=2*3^{-1}=2*4=8 \bmod 11$

$x_3=\lambda^2 - x_1 - x_2=8^2-2-2=60=5 \bmod 11$

$y_3 = \lambda(x_1-x_3) - y_1=8*(2-5)-7=$
$8*8-7=64+4=2 \mod 11$
因此 2α=(5,2)

No.01=(02, 07)
No.02=(05, 02)
No.03=(08, 03)
No.04=(0A, 02)
No.05=(03, 06)
No.06=(07, 09)
No.07=(07, 02)
No.08=(03, 05)
No.09=(0A, 09)
No.10=(08, 08)
No.11=(05, 09)
No.12=(02, 04)
No.13=infinity

ssl\crypto\ec\ectest.c 包含的以下函数:
void prime_field_tests()
里面有各种 ECC 曲线的参数。

7.2 用 ECC 算法加密解密(p.129)
Menezes-Vanstone 公钥密码体制 p.129
1. 公钥及私钥
公钥点 R=d*G
私钥 d 是一个随机数，且 d<n，其中 n 是 G 的阶

2. 加密

r = (k*G).x; 其中 k 是一个随机数且 k<n
; r 不可以 mod n
s = m*(k*R).x mod n; 其中 m 是明文
密文包括两部分:r,s

3. 解密: 红色的 r 是一个点, r=k*G
m = s/(dr).x = m*(k*R).x/(d*(k*G)).x
= m*(k*d*G).x/(k*d*G).x

4. 举例
设曲线方程是 $y^2=x^3+x+6 \pmod{11}$
基点 G=(2,7), G 的阶 n=13
设私钥 d=7, 则公钥 R=dG=7*(2,7)=(7,2)
设随机数 k=6, 明文 m=9, 则
密文第 1 部分 r=kG=6*(2,7)=(7,9)
密文第 2 部分 s=m*(k*R).x=9*(6*(7,2)).x=9*(8,3).x
=9*8 mod 13 = 7
m=s/(dr).x= 7/(7*(7,9)).x =
=7/(8,3).x = 7/8 = $7*8^{-1}$ mod 13 = 7*5 mod 13
=9

7.3 用 ECC 算法签名验证
1. ecdsa(elliptic curve digital signature algorithm)
(1)签名
r = k*G ; k 是随机数
s = (m+r*d)/k ; m 是明文或 hash, d 是私钥

(2)验证

(m/s)*G+(r/s)*R == r

(m/s)*G+(r/s)*R = mG/s + rR/s =

(mG+rdG)/s = (m+rd)G/((m+rd)/k) = kG

如果伪造 m 或 d，都无法通过验证。

2. ecnr

(1)签名

r = k*G+m

s = k-r*d

(2)验证

r - (s*G+r*R) == m

r - (s*G+r*R) = r - ((k-rd)G + rdG)

= r - (kG-rdG+rdG) = r - kG = kG+m-kG = m

==========

1. 证明 gcd(n,u)=an+bu (p.93)

设 n/u 的商为 q,余数为 r，则有

r = n-q*u

若 gcd(n,u)=k，则 r 一定也包含因子 k，因此有

gcd(n,u) = gcd(u,r)

由此可得求 gcd(n,u)的 Euclid 算法如下:

```
y=n;
x=u;
while(x!=0)
{
   q = y/x;
   r = y%x;
   y=x;
   x=r;
}
```

当除数 x=0 时，被除数 y=gcd(n,u)。

现用数学归纳法证明上述算法中的被除数 y 及除数 x 可以表示成:

$$y_i = a1_i*n + b1_i*u \quad (a)$$

$$x_i = a2_i*n + b2_i*u \quad (b)$$

其中 y_i 及 x_i 表示 Euclid 算法第 i 次循环中计算 q、r 时的被除数及除数。

当 i=0 时，只要取 $a1_i=1, b1_i=0, a2_i=0, b2_i=1$，则(a)(b)成立。

设 i=j 时，(a)(b)成立，则当 i=j+1 时，

$$y_{j+1} = x_j = a2_j*n + b2_j*u$$

$$x_{j+1} = y_j \% x_j = a1_j*n + b1_j*u - q_j*(a2_j*n + b2_j*u)$$

$$=(a1_j-q_j*a2_j)*n + (b1_j-q_j*b2_j)*u$$

其中 q_j 表示 Euclid 算法第 j 次循环中计算出来的商。显然，当 i=j+1 时，取

$$a1_{j+1} = a2_j, \ b1_{j+1} = b2_j$$

$$a2_{j+1} = (a1_j-q_j*a2_j), \ b2_{j+1} = (b1_j-q_j*b2_j)$$

即可使(a)(b)成立。

因此，Euclid 算法中的 y 及 x 均可以表示成 an+bu 的形式，而当 x=0 时，y 就是 gcd(n,u)，于是有 gcd(n,u)=an+bu。

2．证明 Euler 准则(p.98)

若方程有解 $y \in Z_p$，则 x 是模 p 的平方剩余：$y^2 = x \bmod p$

设 p>2 是一个素数，x 是一个整数，gcd(x,p)=1，则 x 是模 p 的平方剩余的充要条件是：

$$x^{(p-1)/2} \equiv 1 \pmod p$$

证明：

(1)必要性

因为 $y^2 = x \bmod p$，并且 gcd(x,p)=1，所以一定有

gcd(y, p)=1;

根据 Fermat 小定理知，$y^{p-1} \equiv 1 \bmod p$，因此

$x^{(p-1)/2} = (y^2)^{p-1/2} = y^{p-1} = 1 \bmod p$

(2)充分性

因为 $x^{(p-1)/2} \equiv 1 \pmod p$，且 $x \bmod p \in Z_p$，不妨设 $x \in Z_p$。而 $Z_p=\{0,1,2,\ldots,p-1\}$是有限域，$Z_p^*=\{1,2,3,\ldots p-1\}$在模 p 乘法运算下是一个循环群，所以一定存在 Z_p^*的一个生成元 b，使得下式成立：

$x=b^i \bmod p,\ 1 \leqslant i \leqslant p-1$

例如：$1=4^2 \bmod 5$；$2=3^3 \bmod 5$；$3=2^3 \bmod 5$；

$4=3^2 \bmod 5$；

因此，

$1 = x^{(p-1)/2} = (b^i)^{(p-1)/2} = (b^{p-1})^{i/2} \bmod p$

因为 b 的阶为 p-1，即 $b^{p-1} \bmod p = 1$，所以 i 必定是偶数，于是 x 模 p 的平方根有整数解，并且其值为 $\pm b^{i/2} \bmod p$。

$x = a1 \bmod m1$

$x = a2 \bmod m2$

$x = a3 \bmod m3$

当 m1、m2、m3 互素时，上述方程组的解 x 是唯一的，

其值= Y mod M

Y=(

$a1*(m2*m3)*(m2*m3)^{-1} \bmod m1 +$

$a2*(m1*m3)*(m1*m3)^{-1} \bmod m2 +$

$a3*(m1*m2)*(m1*m2)^{-1} \bmod m3);$

M= m1*m2*m3;

先证明 $Y \bmod m_i$ 是方程组的一个解

3．证明中国剩余定理(p.83)

设 $m_1,m_2,m_3,\ldots,m_r$ 两两互素，则以下同余方程组

$$x \equiv a_i \pmod{m_i},\ i=1,2,3,\ldots,r \qquad (a)$$

模$M=m_1m_2m_3\ldots m_r$的唯一解为

$$x = \sum_{i=1}^{r} a_i * M_i * (M_i^{-1} \bmod m_i) \bmod M, \text{ 其中} M_i = M/m_i \qquad (b)$$

(1)先证明$\sum_{i=1}^{r} a_i * M_i * (M_i^{-1} \bmod m_i)$ (c)是同余方程组(a)的一个解

对于任意 $1\leqslant j\leqslant r$，都有$\sum_{i=1}^{r} a_i * M_i * (M_i^{-1} \bmod m_i) \bmod m_j$

$=a_j$，所以(c)是(a)的一个解。

(2)再证明(b)是同余方程组(a)的模 M 唯一解

假定 x_1 及 x_2 是(a)的两个不同解，即

$$x_1 = a_i \bmod m_i,\ i=1,2,3\ldots,r$$

$$x_2 = a_i \bmod m_i,\ i=1,2,3\ldots,r$$

则 $x_1-x_2 = 0 \bmod m_i,\ i=1,2,3,\ldots,r$

即 $m_i \mid (x_1-x_2),\ i=1,2,3,\ldots,r$

又因为 m_1,m_2,m_3,m_r 两两互素，所以

$$M \mid (x_1-x_2)$$

即

$$x_1 = x_2 \bmod M$$

因此(b)是(a)模 M 的唯一解。