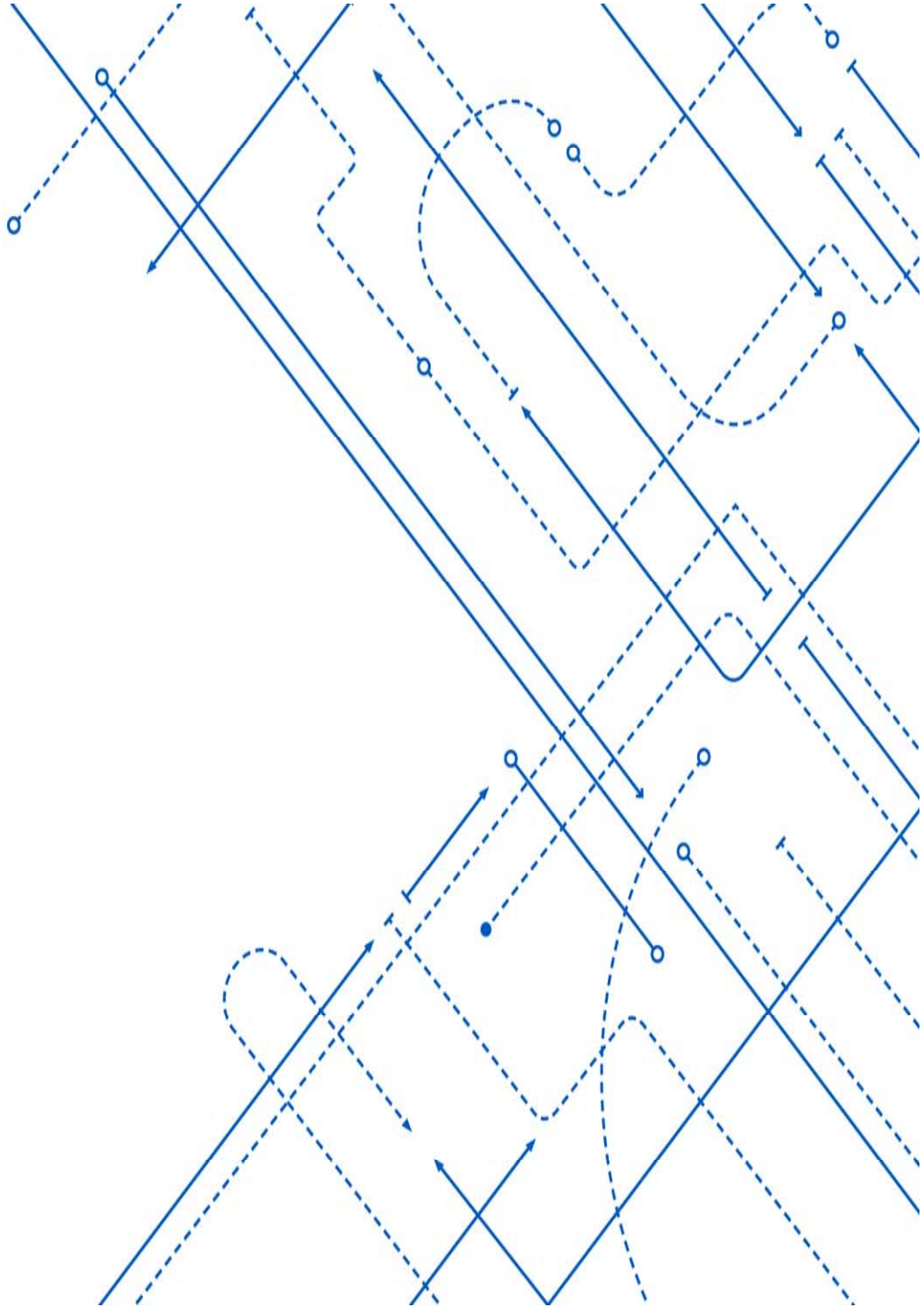
密码学

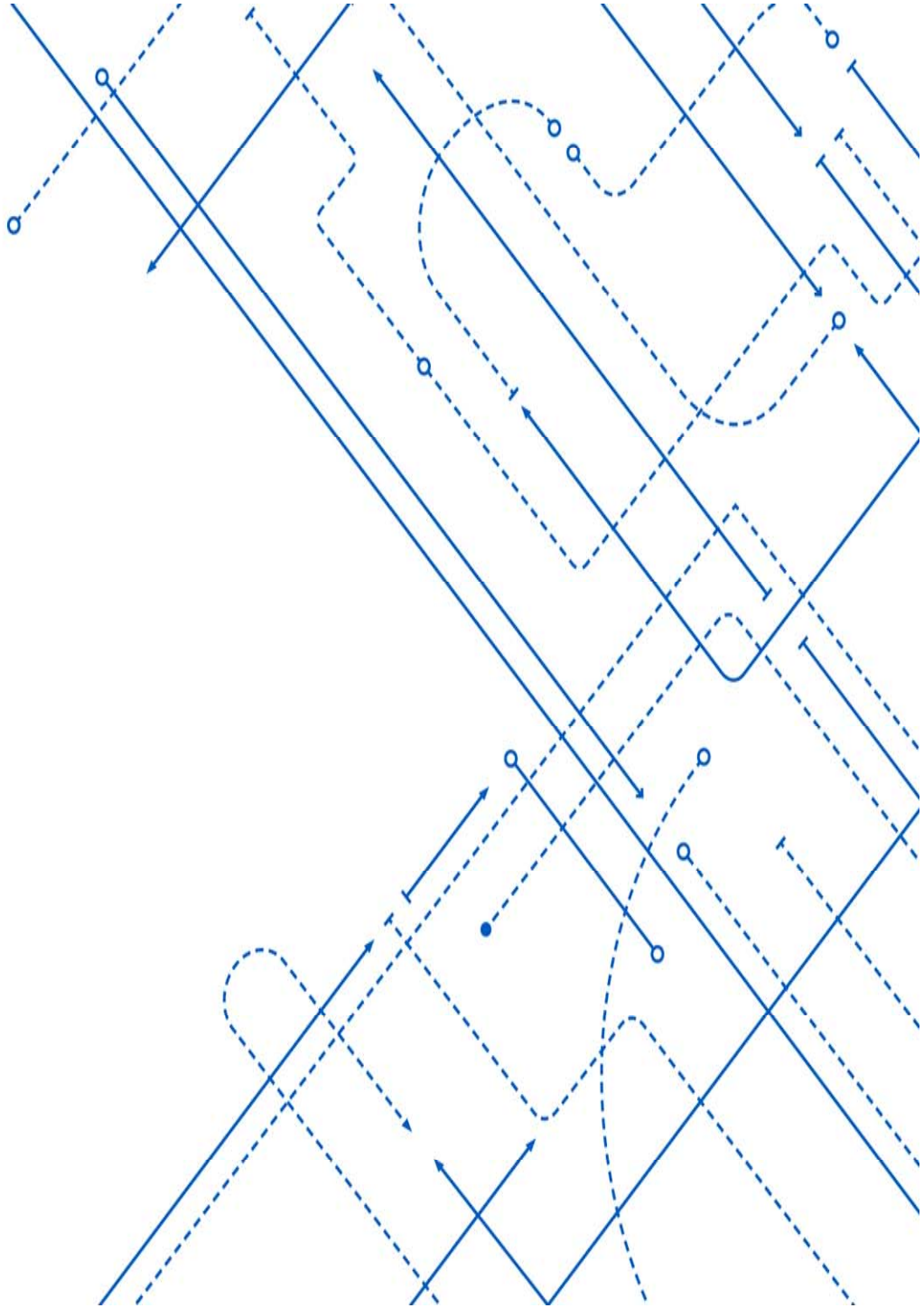
张帆

fanzhang@zju.edu.cn

浙江大学 ZHEJIANG UNIVERSITY



期末复习



浙江大学
ZHEJIANG UNIVERSITY

最大公约数(greatest common divisor)

- gcd相关的定理（裴蜀定理）：
  - 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。

欧几里德算法（Euclidean algorithm）

- 又称辗转相除法，能有效找出两数的最大公约数。
- 原理：两个整数的最大公约数等于其中较小的数和两数相除余数的最大公约数。
- 利用欧几里德算法计算gcd(a,b)：
  - EUCLID(a,b)
  - 1. A = a; B = b
  - 2. if B = 0 return A = gcd(a, b)
  - 3. R = A mod B
  - 4. A = B
  - 5. B = R
  - 6. goto 2

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)　　即 n是a的整数倍
- 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)

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}。

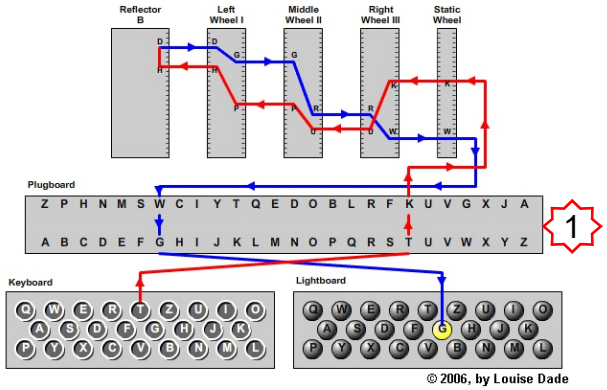
- 常用于加密算法中，如仿射算法。

- 举例： 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 即互素
  - 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)（辗转相除法）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。

扩展欧几里德法

- EXTENDED EUCLID(m, b)
- 1. (A1, A2, A3)=(1, 0, m);
- (B1, B2, B3)=(0, 1, b)
- 2. if B3 = 0
- return A3 = gcd(m, b); no inverse
- 3. if B3 = 1
- return B3 = gcd(m, b); $B2 = b^{-1} \bmod m$
- 4. Q = A3 div B3
- 5. (T1, T2, T3)=(A1 − Q B1, A2 − Q B2, A3 − Q B3)
- 6. (A1, A2, A3)=(B1, B2, B3)
- 7. (B1, B2, B3)=(T1, T2, T3)
- 8. goto 2

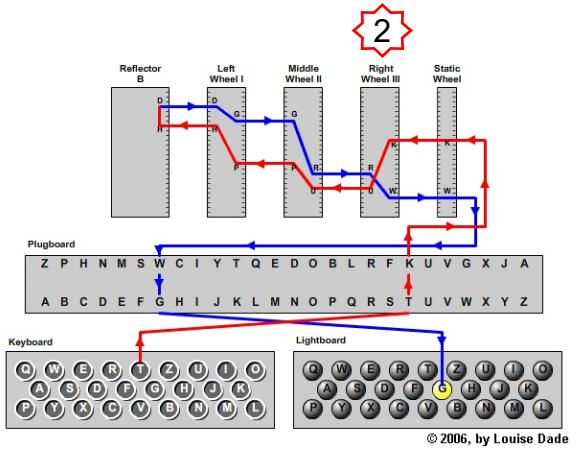
Enigma – 加密示例



- 假设明文为 A ，经历三个齿轮，delta = 0
- 第一步：接线板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

Enigma – 加密示例

- 第二步： 齿轮I rotor I
- char rotor[27]="EKMFLGDQVZNTOWYHXUSPAIBRCJ";
- // ABCDEFGHIJKLMNOPQRSTUVWXYZ
- char c='B', e;
- e = rotor[c-'A']; // e='K';
- B→K

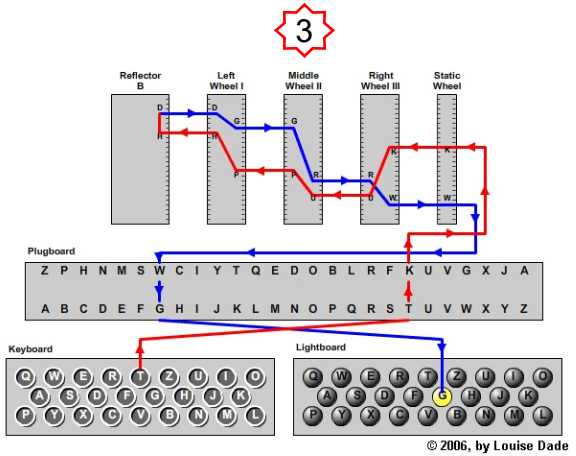


Enigma – 加密示例

- 第三步： 齿轮II rotor II

- AJDKSIRUXBLHWTMCQGZNPYFVOE
- ABCDEFGHIJKLMNOPQRSTUVWXYZ

- K→L

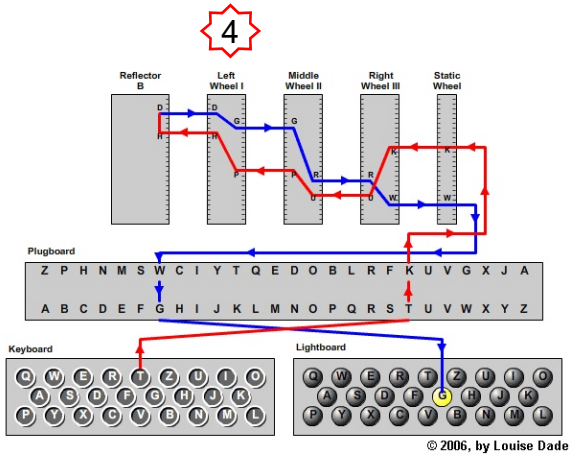


Enigma – 加密示例

- 第四步： 齿轮III rotor III

- BDFHJLCPRTXVZNYEIWGAKMUSQO
- ABCDEFGHIJKLMNOPQRSTUVWXYZ

- L→V

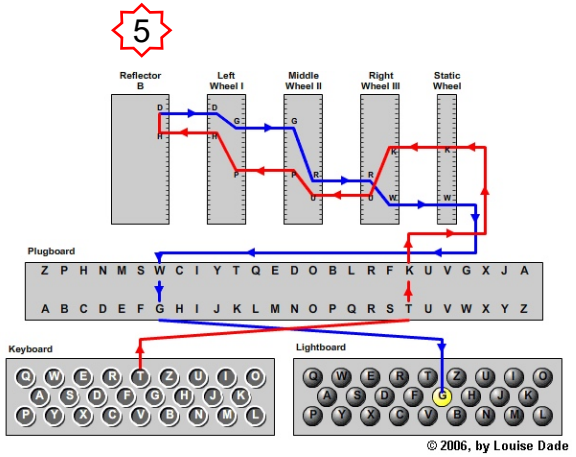


Enigma – 加密示例

- 第五步： 反射器 reflector

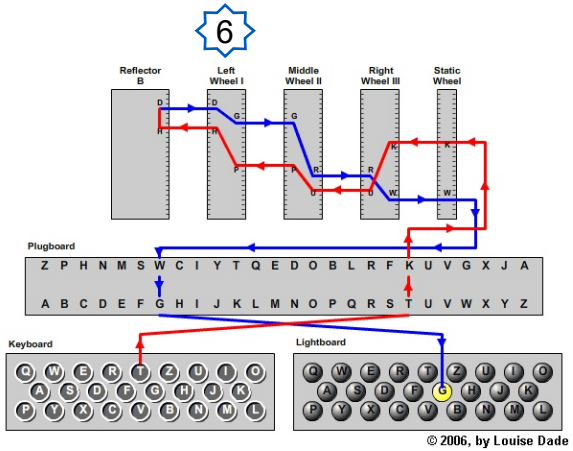
- char reflector[27]=" YRUHQSLDPXNGOKMIEBFZCWVJAT";
- // ABCDEFGHIJKLMNOPQRSTUVWXYZ

- V→W



Enigma – 加密示例

- 经过上述5步, A转成W, 但W还不是密文, 还要走一条逆向路径:4-3-2-1, 把W进一步转化成密文。
- 第六步： 齿轮III rotor III
- char rotor[27]="BDFHJLCPRTXVZNYEIWGAKMUSQO";
- // ABCDEFGHIJKLMNOPQRSTUVWXYZ



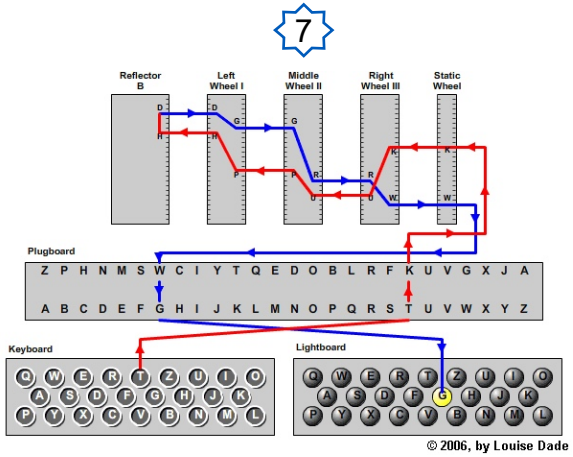
- W→R
- 注意按逆向路径经过3个齿轮时要反查表, 即在数组rotor中寻找元素'W', 得到该元素的下标设为i, 再把i+'A',就得到'R'。

Enigma – 加密示例

- 第七步： 齿轮II rotor II

- AJDKSIRUXBLHWTMCQGZNPYFVOE
- ABCDEFGHIJKLMNOPQRSTUVWXYZ

- R→G

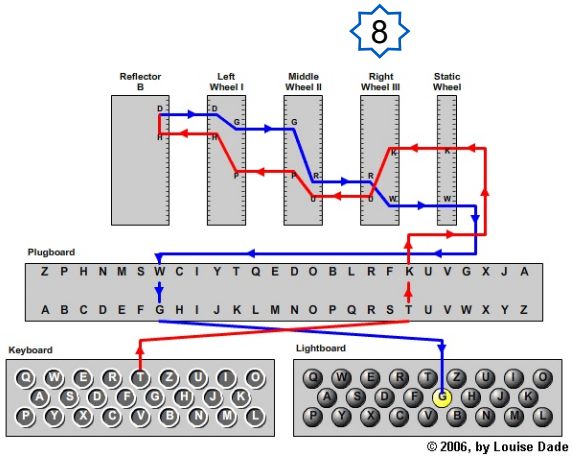


Enigma – 加密示例

- 第八步： 齿轮I rotor I

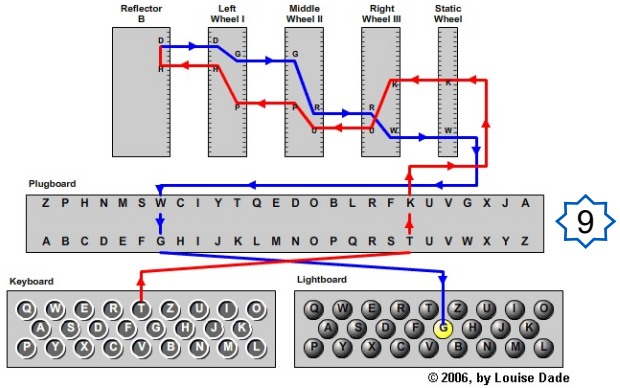
- EKMFLGDQVZNTOWYHXUSPAIBRCJ
- ABCDEFGHIJKLMNOPQRSTUVWXYZ

- G→F



Enigma – 加密示例

- 第九步：接线板plugboard
- char plug[27] ="BADCEFGHIJKLMNOPQRSTUVWXYZ";
  /*ABCDEFGHIJKLMNOPQRSTUVWXYZ*/



- F→F
- W转成F, 其中F就是密文。
- 解密时，尝试把F当作明文重新把前面的9步走一遍,最后出来的是A。

MD5计算-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。

MD5计算-分块及填充

- (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。

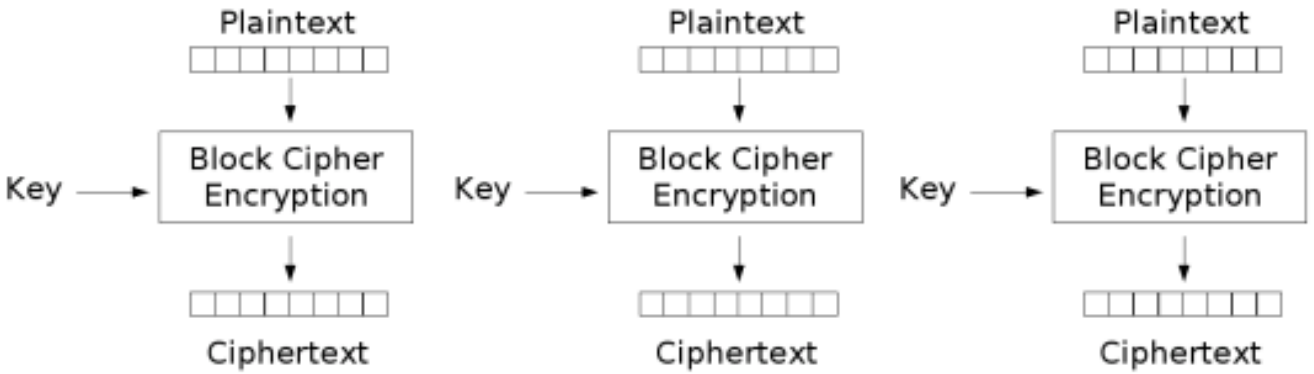
- (3)再在后面补上8个字节，这8个字节相当于一个64位的整数，它的值=message总共的位数(不含填充内容)。

SHA-1

- 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（大端存储位数）。

电子密码簿模式（ECB）- 加密

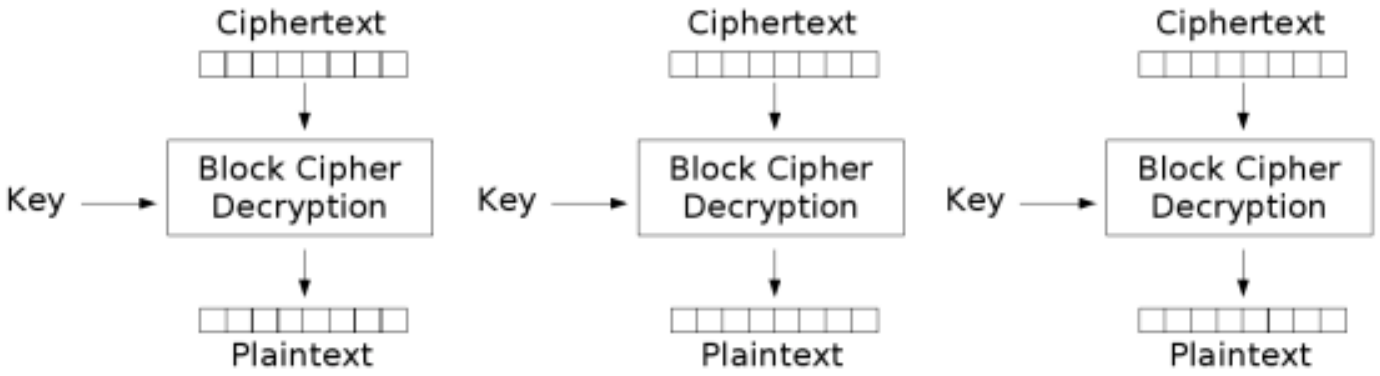
- 加密过程：$C_j = E_k(P_j)$



Electronic Codebook (ECB) mode encryption

电子密码簿模式（ECB）- 解密

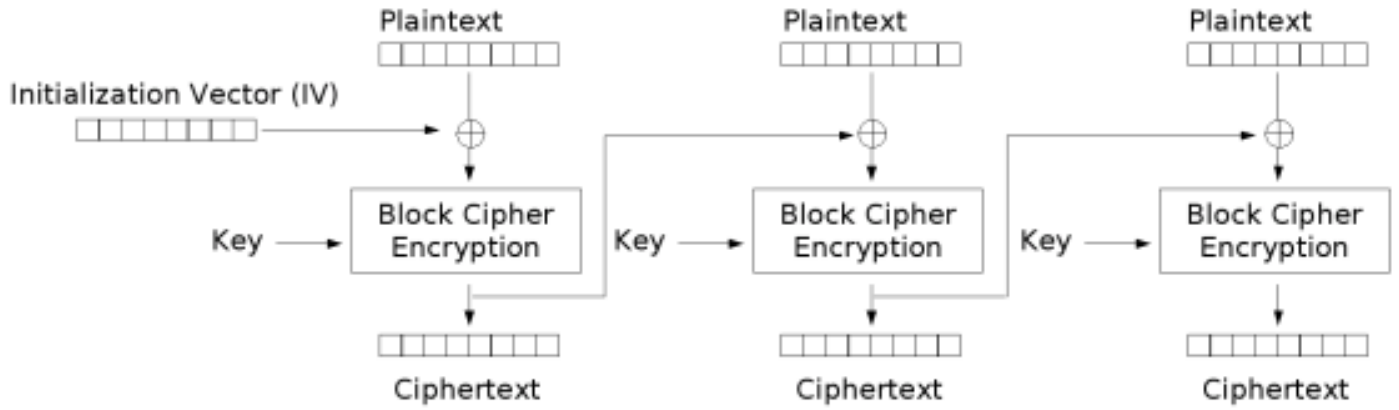
- 解密过程：$P_j = D_k(C_j)$



Electronic Codebook (ECB) mode decryption

密文块链接模式（CBC）- 加密

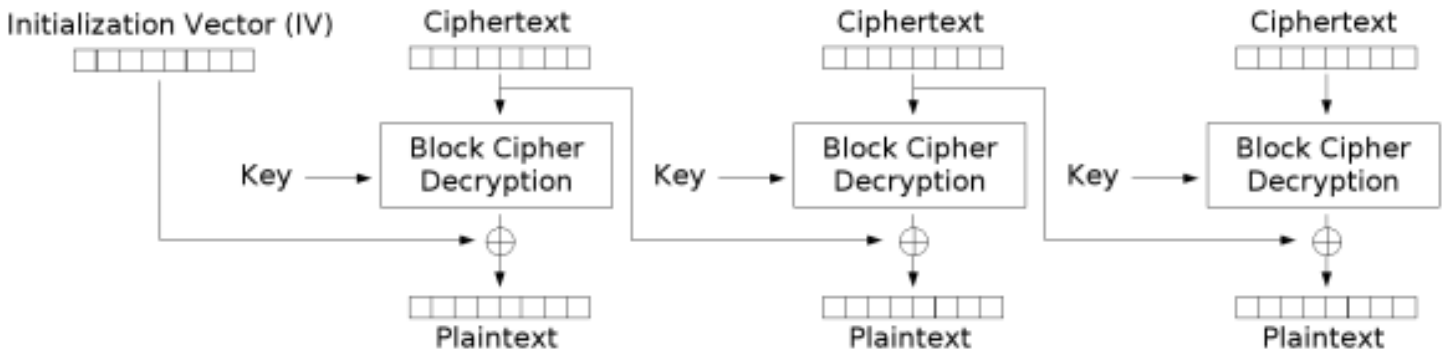
➢ 加密过程： $C_j = E_k(P_j \oplus C_{j-1})$



Cipher Block Chaining (CBC) mode encryption

密文块链接模式（CBC）- 解密

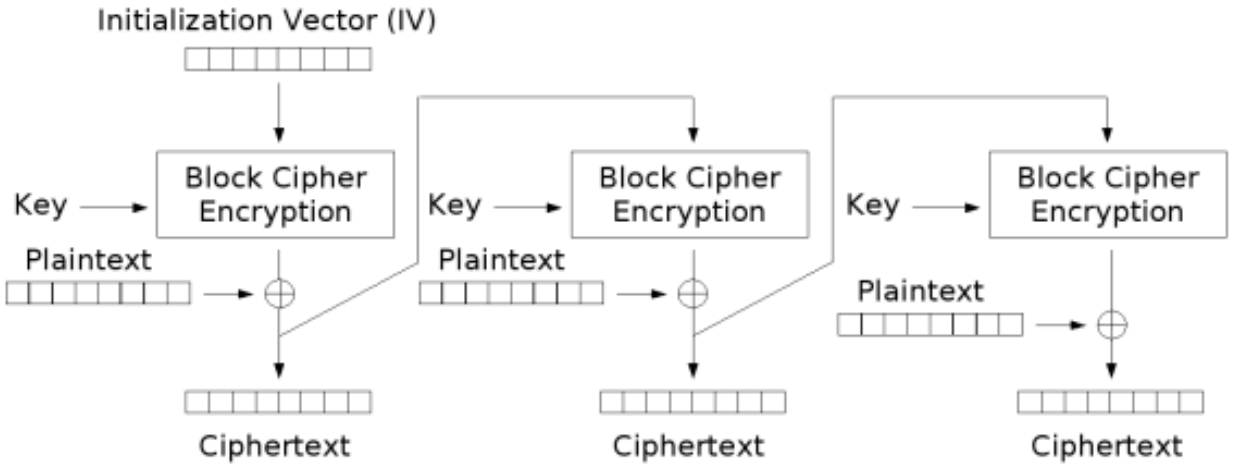
➢ 解密过程： $P_j = D_k(C_j) \oplus C_{j-1}$



Cipher Block Chaining (CBC) mode decryption

密文反馈模式（CFB）- 加密

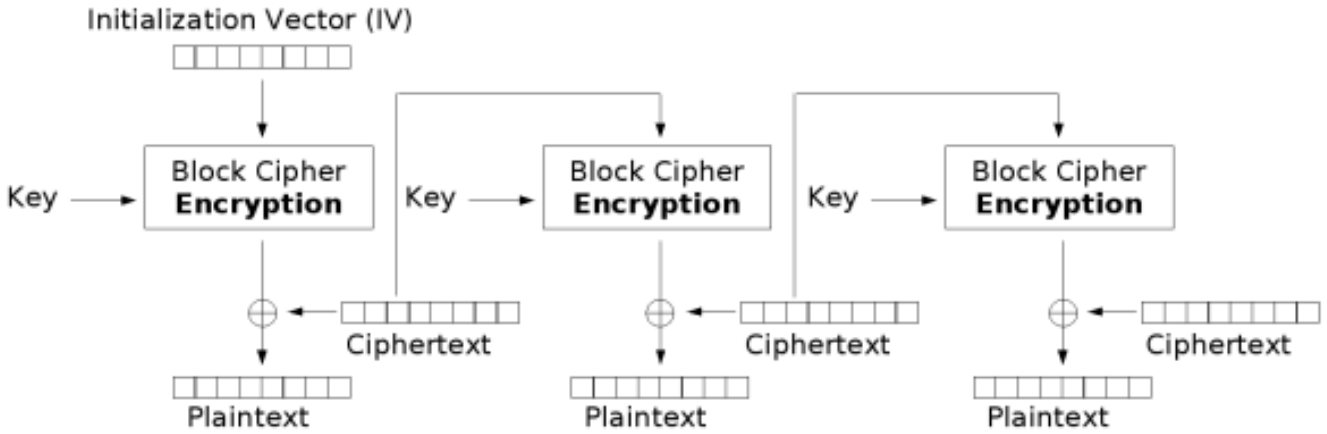
➢ 加密过程：$C_j = E_k(C_{j-1}) \oplus P_j$



Cipher Feedback (CFB) mode encryption

密文反馈模式（CFB）- 解密

- 解密过程： $P_j = E_k(C_{j-1}) \oplus C_j$



Cipher Feedback (CFB) mode decryption

密文反馈模式（CFB）- 错误恢复示例

- 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，...，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

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

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

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

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

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

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。

RC4-流程

- RC4 主要包含三个流程：

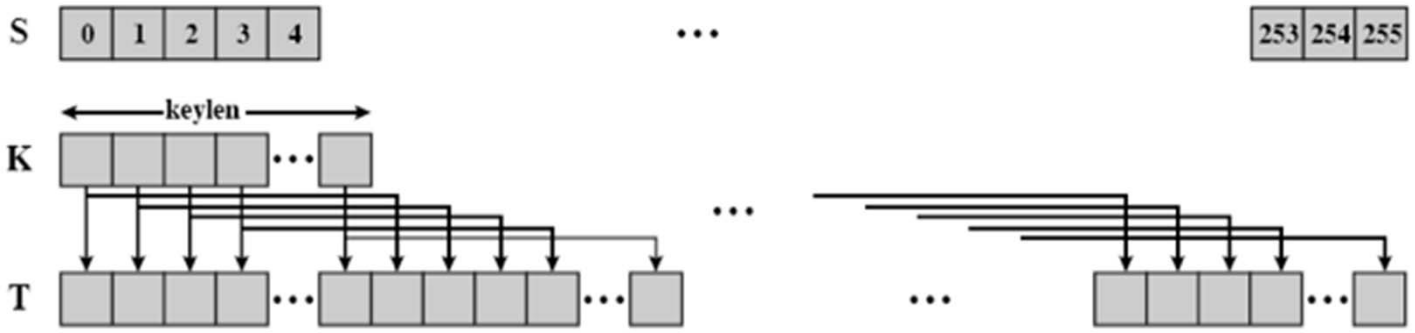
- 1. 初始化 S 和 T 数组。
- 2. 初始化置换 S。
- 3. 生成密钥流并加密。

- 一般称前两部分为 KSA（Key-scheduling algorithm），最后一部分是 PRGA（Pseudo-random generation algorithm）。

RC4-流程

➢ 1. 初始化 S 和 T 数组。

for i = 0 to 255 do
  S[i] = i
  T[i] = K[i mod keylen])

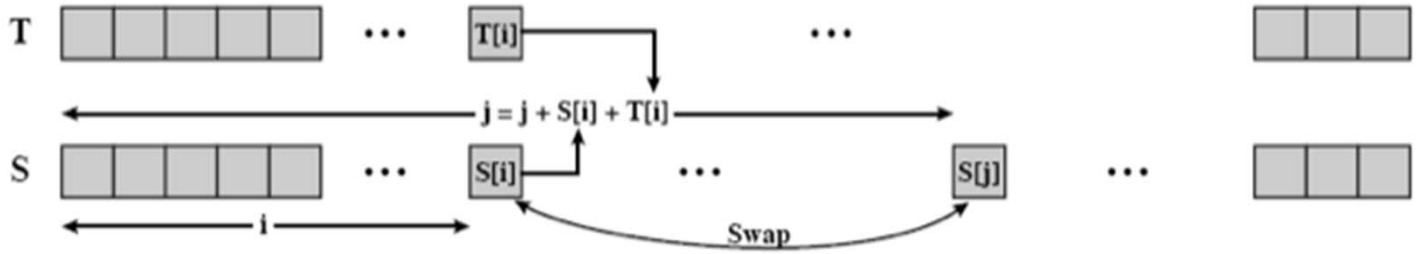


(a) Initial state of S and T

RC4-流程

➢ 2. 初始化置换 S。

```
j = 0
for i = 0 to 255 do
   j = (j + S[i] + T[i]) (mod 256)
   swap (S[i], S[j])
```



(b) Initial permutation of S

RC4-流程

➢ 3. 生成密钥流并加密。

$i = j = 0$

for each message byte M_i

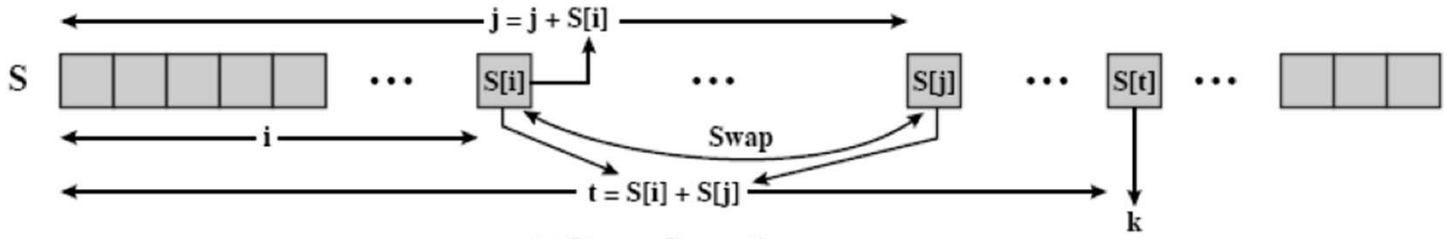
$i = (i + 1) \pmod{256}$

$j = (j + S[i]) \pmod{256}$

swap($S[i]$, $S[j]$)

$t = (S[i] + S[j]) \pmod{256}$

$C_i = M_i$ XOR $S[t]$



(c) Stream Generation

DES算法流程

(1)64位明文在进入加密前有个打乱的过程，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;

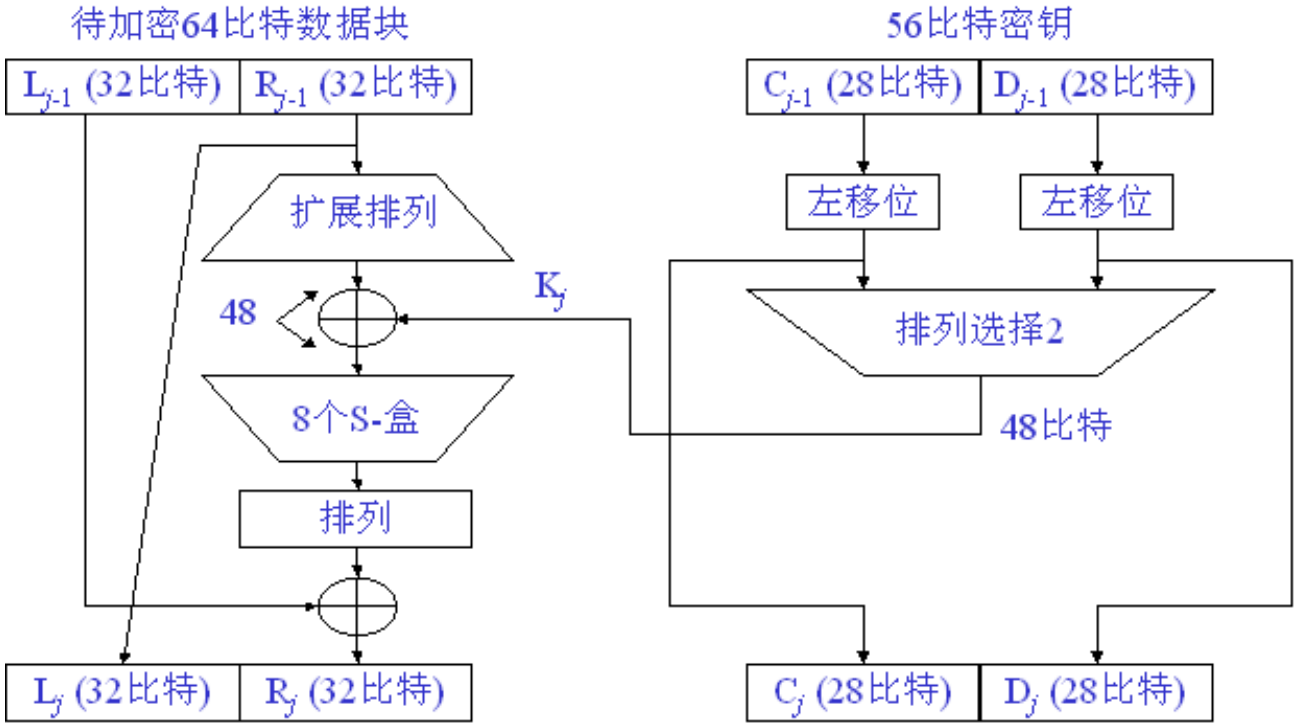


图2.6 DES算法每轮处理过程

DES算法流程

(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];

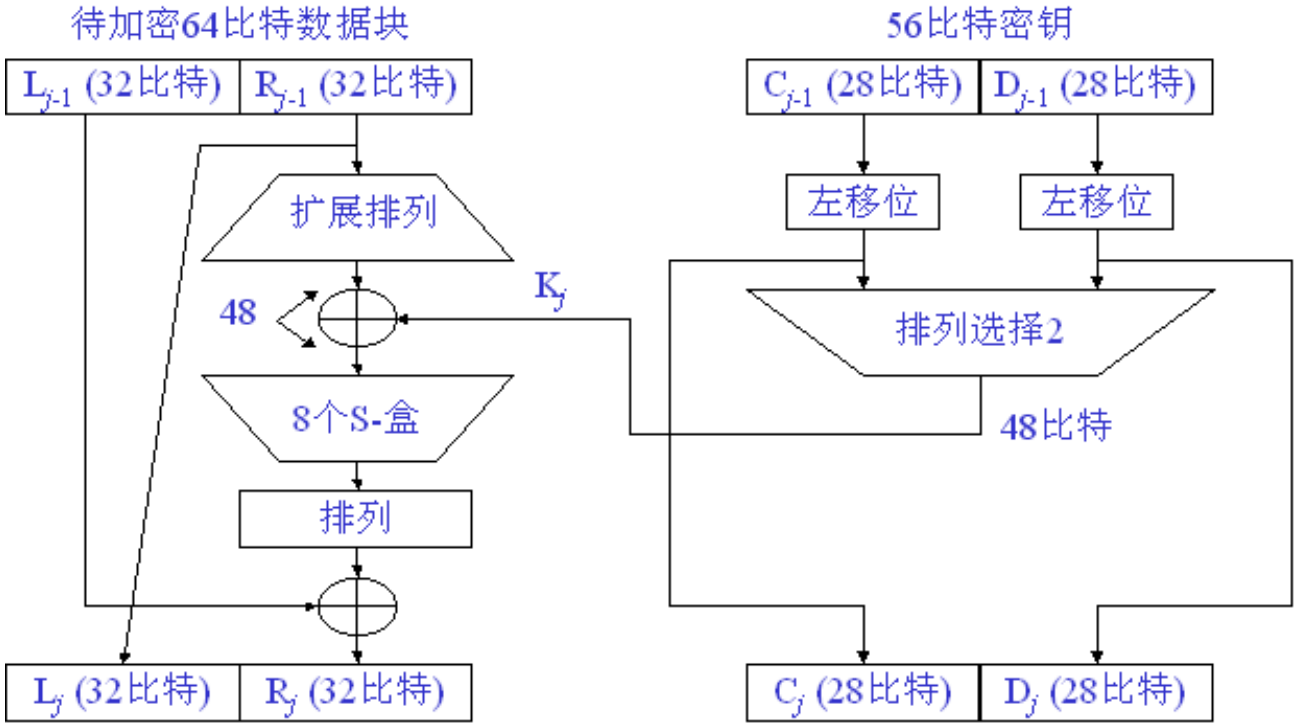


图2.6 DES算法每轮处理过程

DES算法流程

(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];

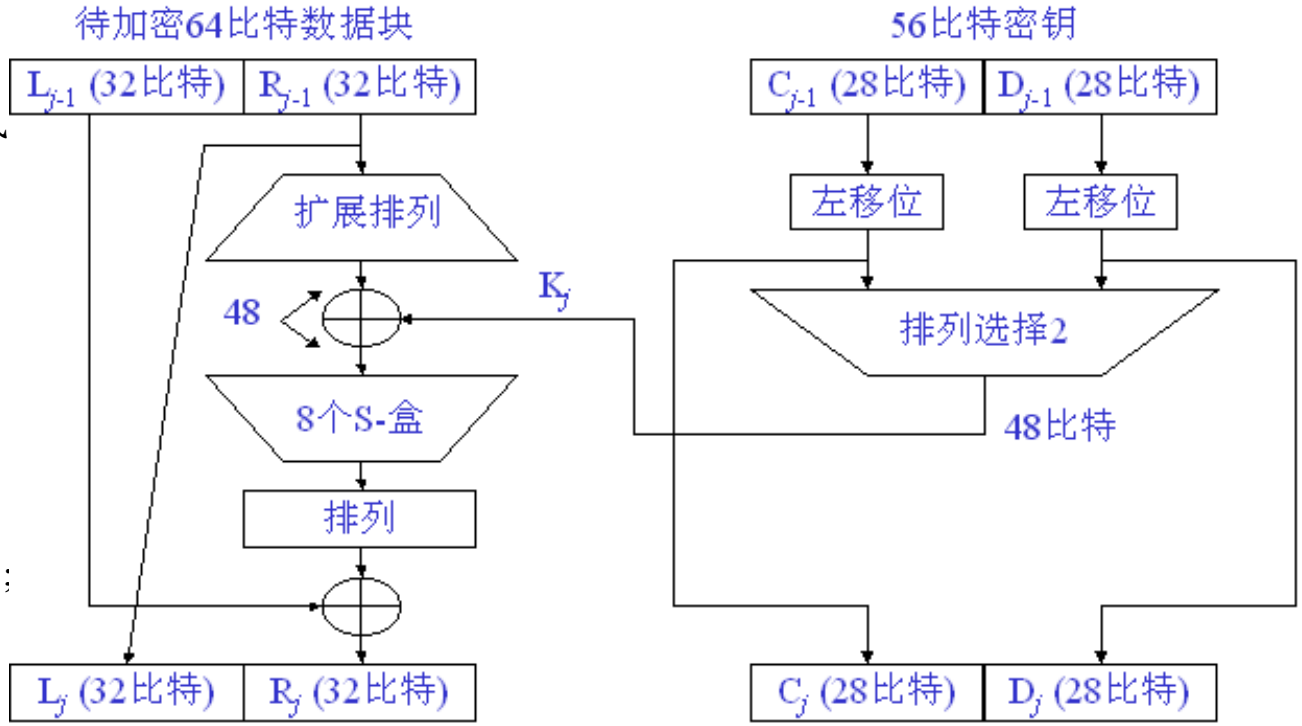


图2.6 DES算法每轮处理过程

DES －源代码分析

- 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。

DES －源代码分析

- 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。

DES －源代码分析

- 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。

DES －源代码分析

- 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。

DES －源代码分析

- 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。

DES －源代码分析

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

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

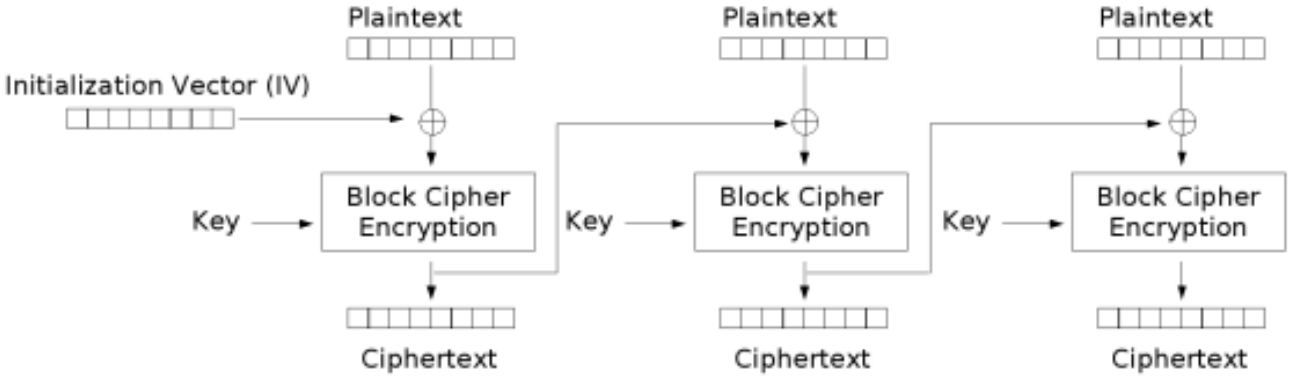
进入8个sbox(每6位进入一个sbox并出来4位)

sbox共输出32位,打乱后生成一个新的32位数N32

最后L32' = L32 xor N32

DES －分组加密模式 － CBC

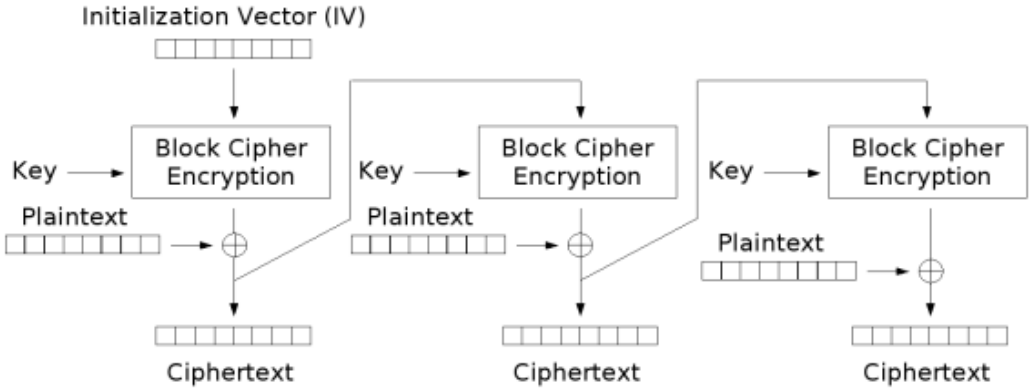
- 需要一个8字节的初始向量(initialization vector)或初始变量(starting value)。令iv=这8个字节。现要加密block[0], block[1]：
- c[0] = des_encrypt(block[0] ^ iv);
- c[1] = des_encrypt(block[1] ^ c[0]);
- c[0]和c[1]就是密文。



Cipher Block Chaining (CBC) mode encryption

DES - 分组加密模式 - 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 */

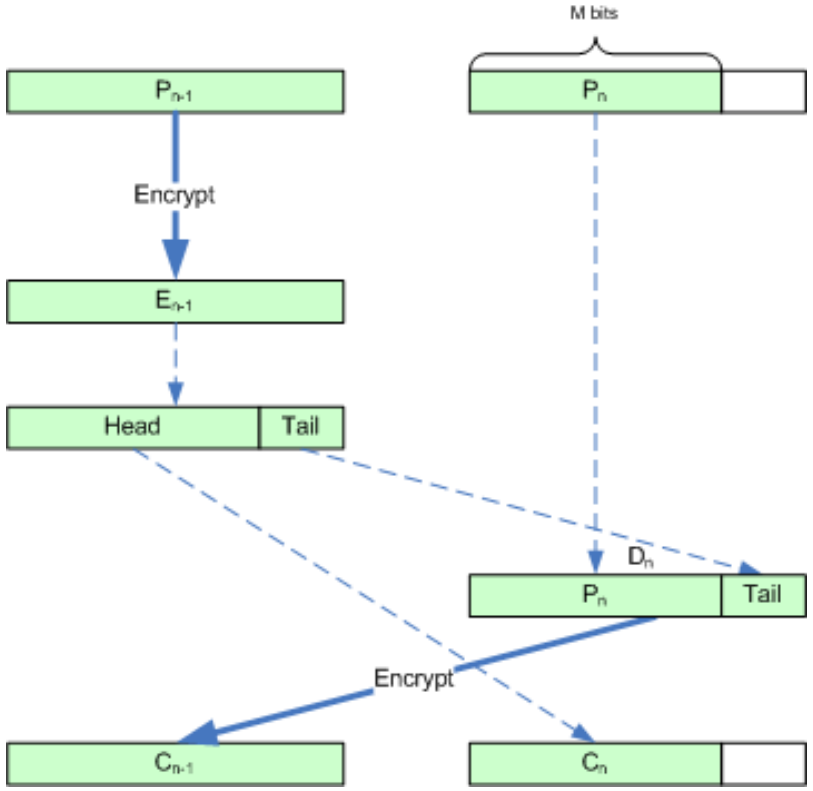


Cipher Feedback (CFB) mode encryption

DES －ciphertext stealing － ECB

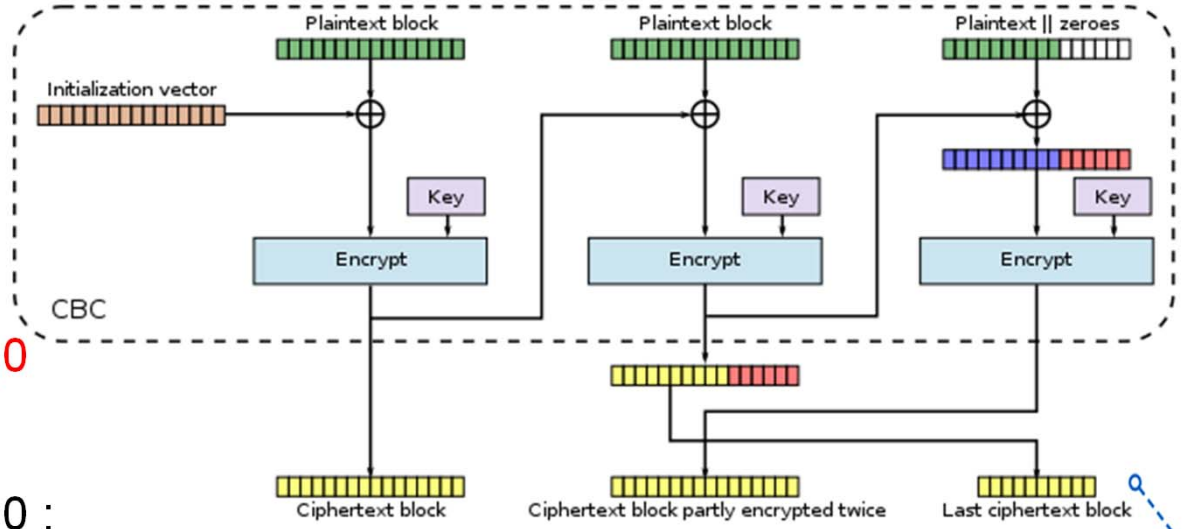
- E_{n-1}= Encrypt（K， P_{n-1}）
- C_n = Head (E_{n-1}, M)
- D_n = P_n || Tail (E_{n-1}, B−M)
- C_{n-1}= Encrypt (K, D_n)

- 举例：
- 12345678 12345
- ~~12345678~~
- 12345~~678~~ ~~12345~~
- 12345~~678~~ ~~12345~~



DES －ciphertext stealing － CBC

- $X_{n-1}=P_{n-1}$ XOR C_{n-2}
- E_{n-1}= Encrypt (K, X_{n-1})
- C_n = Head (E_{n-1}, M)
- P = P_n || 0^{B-M} 在Pn的末端加上零，以产生长度为B的P。
- D_n = E_{n-1} XOR P
- C_{n-1} = Encrypt (K, D_n)
- 举例：
- IV
- 12345678 12345,0,0,0
- ~~12345678~~
- 12345~~678~~ ~~12345~~,0,0,0 ;
- 12345~~678~~ ~~12345~~



三重DES

- 因此必须使用3重加密
  - 似乎需要3把不同的钥匙
- 但可以使用两个Key和E-D-E序列
  - $C = E_{K1}(D_{K2}(E_{K1}(P)))$
  - $P = D_{K1}(E_{K2}(D_{K1}(C)))$
  - nb加密与解密在安全性方面的等效性
  - 如果K1=K2，则使用单DES
- 使有效密钥长度加倍
  - 112位密钥非常强大
  - 即使是今天的电脑，所有可行的已知攻击都无法攻破
  - $O(2^{112})$ 穷举攻击 / 10^{52} 差分密码分析
- 三重DES目前的应用
  - 金融

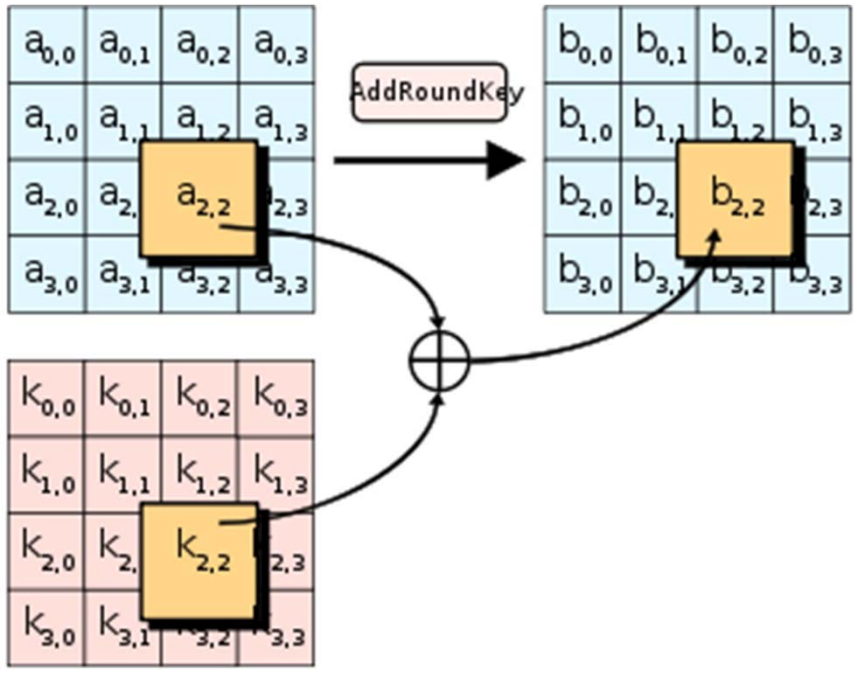
AES算法流程

➢ 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));
}
```

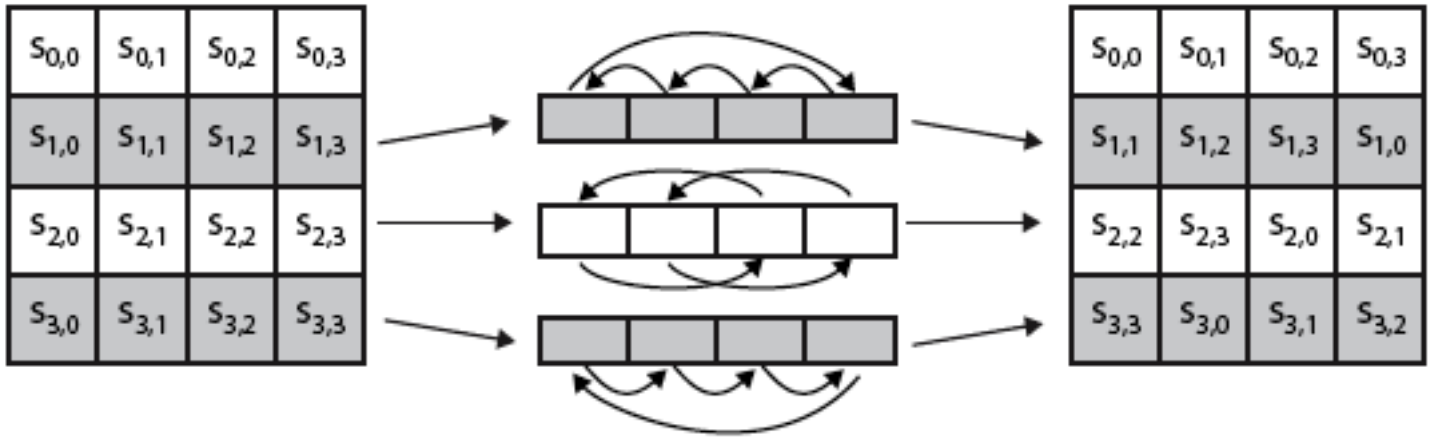
AES算法流程 - AddRoundKey

- 回合密钥将会与原矩阵合并。
- 在每次的加密循环中，都会由主密钥产生一把回合密钥（透过Rijndael密钥生成方案产生），这把密钥大小会跟原矩阵一样，以与原矩阵中每个对应的字节作异或（⊕）加法。



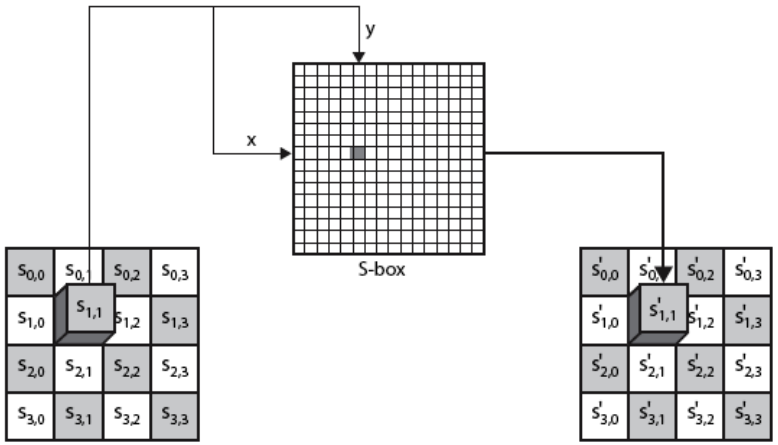
AES算法流程 - ShiftRow

- 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



AES算法流程 - ByteSub

- 逐字节的简单替换
- 使用一个16x16字节的表，其中包含所有256个8位值的排列
- 每个字节都由按行（左4位）和列（右4位）索引的字节替换
  - 例如，byte{95}被第9行第5列中的字节替换
- 使用GF（2^8）中定义的值转换构造的S盒
- 旨在抵抗已知的攻击



MixColumn中的8位数乘法运算 -用农夫算法计算8位数乘法 mod 0x11B

- 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

MixColumn中的8位数乘法运算

- 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:
- $(x^7 + x^3) * (x^2 + 1) = x^9 + x^7 + x^5 + x^3$

$$= x^7 + x^4 + x^3 + x^2 + x \mod (x^8 + x^4 + x^3 + x + 1)$$

MixColumn中的8位数乘法运算

- 用手工除法求模:

$$
\begin{array}{r|l}
 & x \\
\hline
x^8 + x^4 + x^3 + x + 1 & x^9 + x^7 + x^5 + x^3 \\
 & x^9 + x^5 + x^4 + x^2 + x \\
\hline
 & x^7 + x^4 + x^3 + x^2 + x
\end{array}
$$

- 把余式转化成二进制就是:
- 1001 1110
- 结论:
- 0x88 * 0x05 = 0x9E mod 0x11B

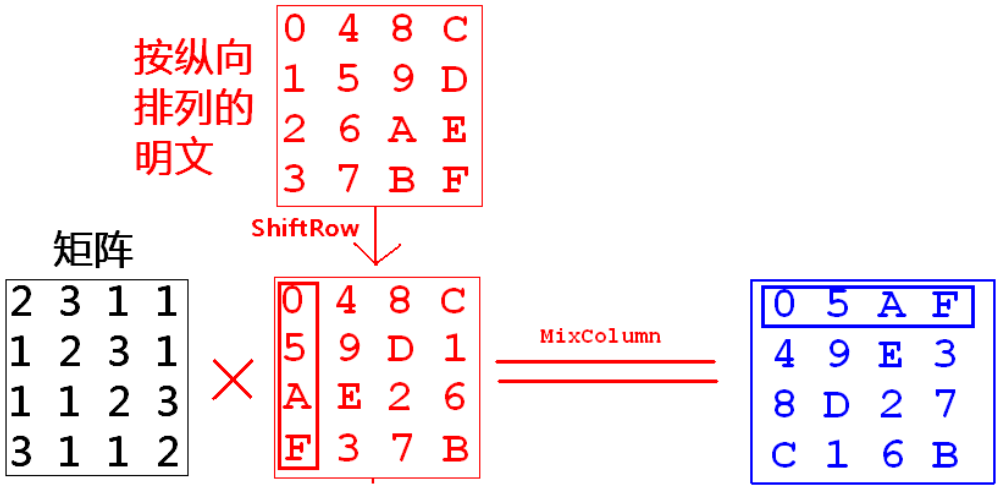
MixColumn的具体步骤

- 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;
- (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};

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

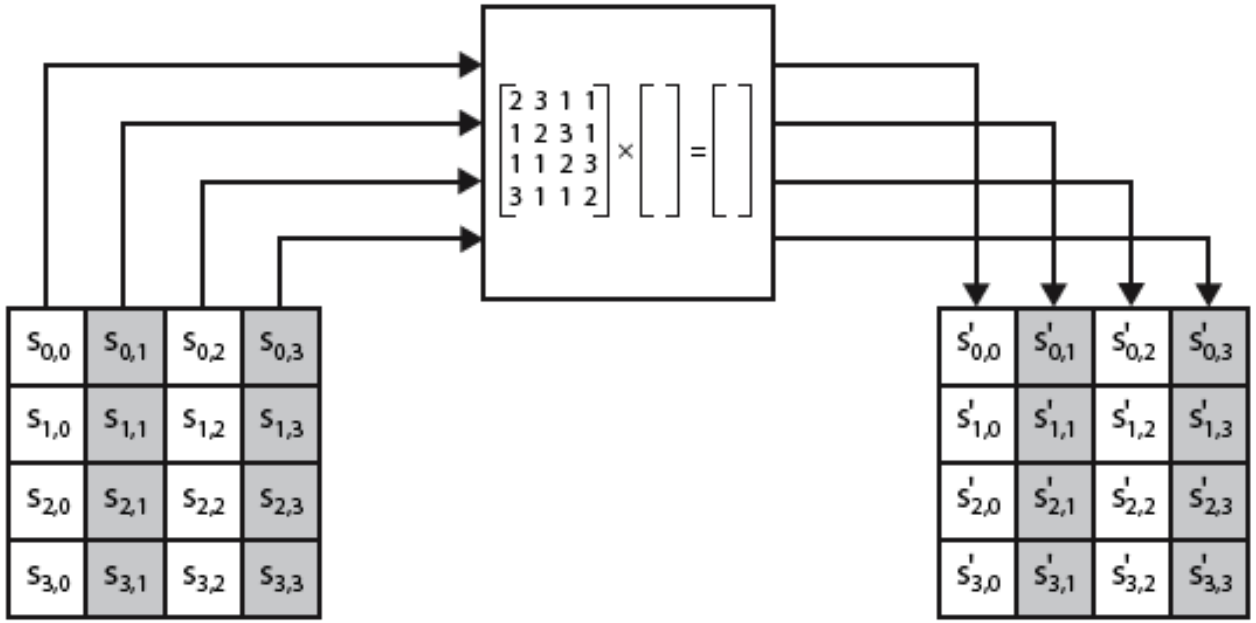
MixColumn的具体步骤

(5) 矩阵m中的4列按以下顺序分别与a做乘法运算:



MixColumn的具体步骤

- (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;



AES密钥的生成过程

- 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 = 21 = 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];

AES密钥的生成过程

- 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()。

RSA算法

- 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 mod ((p-1)*(q-1))

RSA算法

- 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：

$$c=m^e(\text{mod } n)$$

- 按下面的公式进行解密：

$$m=c^d(\text{mod } n)$$

RSA算法-具体流程

1、随机选择两个不等素数p和q。

2、计算出n=p*q。

3、选择一个数e使它和(p-1)(q-1)互素。

4、计算e在模(p-1)(q-1)的逆元d。

5、公开 (e,n)作为RSA公钥。

6、保留 (d,n)作为RSA私钥。

RSA算法的数学基础 – Euler函数

- φ(n): 小于n且与n互素的整数个数。

- 例如φ(5) = 4, 因为与5互素的整数有: 1, 2, 3, 4
- 例如φ(10)= 4, 因为与10互素的整数有: 1, 3, 7, 9

Euler

RSA算法的数学基础 – Euler定理

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

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

RSA算法的数学基础 – Fermat小定理

- 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$\varphi(p) = p-1$。

Fermat

RSA算法的数学基础 – 中国剩余定理

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

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

模 $M=m_1m_2m_3\cdots m_r$ 的唯一解为

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

证明中国剩余定理

- 中国剩余定理：
- 设$m_1, m_2, m_3 \cdots, m_r$两两互素，则以下同余方程组

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

模 $M=m_1m_2m_3\cdots m_r$ 的唯一解为

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

证明中国剩余定理

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

- 对于任意$1 \leqslant j \leqslant r$，都有$\sum_{i=1}^{r} a_i * M_i * (Mi^{-1} \bmod mi) \bmod m_j = a_j$，所以(c)是(a)的一个解。

证明中国剩余定理

(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
- $x_1 = a_i \mod m_i$， $i=1,2,3\cdots,r$
- $x_2 = a_i \mod m_i$， $i=1,2,3\cdots,r$
- 则$x_1-x_2 = 0 \mod m_i$， $i=1,2,3,\cdots,r$
- 即$m_i \mid (x_1-x_2)$， $i=1,2,3,\cdots,r$
- 又因为m_1,m_2,m_3,m_r两两互素，所以 $M \mid (x_1-x_2)$
- 即 $x_1 = x_2 \mod M$
- 因此(b)是(a)模M的唯一解。

RSA算法的数学基础 – Euler函数的乘法性质

- 若n1,n2互素, 则$\varphi(n_1*n_2)=\varphi(n_1)*\varphi(n_2)$
- 例如: $\varphi(3*5)=\varphi(3)*\varphi(5)=2*4=8$
- 与15互素的数包括: 1,2,4,7,8,11,13,14

RSA算法的数学基础 – Euler函数的乘积公式

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

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

RSA算法证明

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

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

RSA算法证明

- 方法1:

为什么$(m^{\phi(n)})^k = (1)^k \bmod n$?

a = a' mod n

b = b' mod n

则一定有a*b = a'* b' mod n, 这是因为:

a = kn+a'

b = jn+b'

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

RSA算法证明

- 方法1：

上述证明的前提是gcd(m, n)=1。

当gcd(m, n) ≠ 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：

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

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

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

RSA算法证明

- 方法2:

$ed = 1 \bmod \phi(p*q)$ ➔

$ed - 1 = k(p-1)(q-1)$ ➔

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

②设$m \neq 0 \bmod p$，则

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

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

RSA算法证明

- 方法2：

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)

RSA算法证明

➢ 方法2：

根据(a)(b)可得：

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

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

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：

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

上述结论其实可以由中国剩余定理得出：

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

RSA算法证明

- 方法2:

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:

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

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

➔ $k_1*P + m = k_2*Q + m$ ➔ $k_1 * P = k_2 * Q$ ➔

$k_1 * P = 0 \bmod Q$

$k_2 * Q = 0 \bmod P$

RSA算法证明

- 方法2:

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

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

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

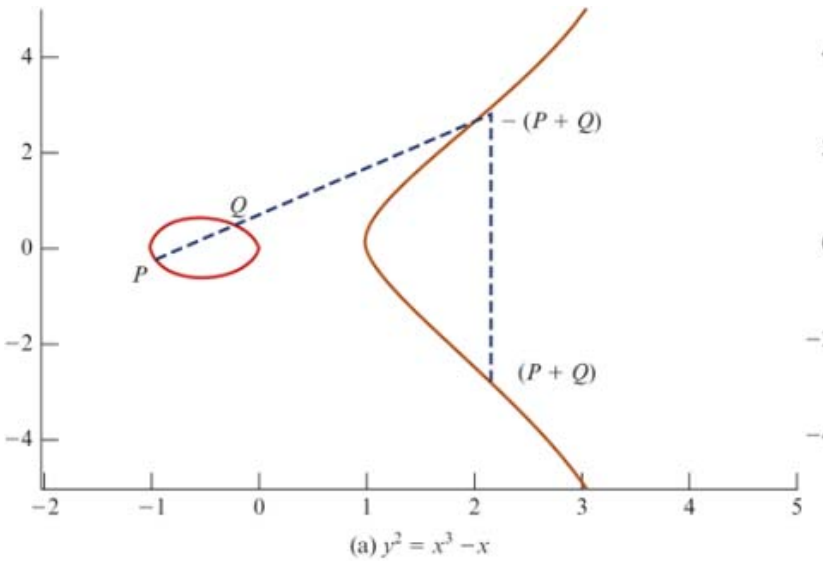
1. RSA_public_encrypt()　公钥加密
2. RSA_private_decrypt() 私钥解密
3. RSA_private_encrypt() 私钥加密
4. RSA_public_decrypt()　公钥解密
5. RSA_new() 分配一个RSA结构指针
6. RSA_free() 释放一个RSA结构指针
7. BN_new() 分配一个大数; BN:Big Number

调用openssl库函数 – 大数处理

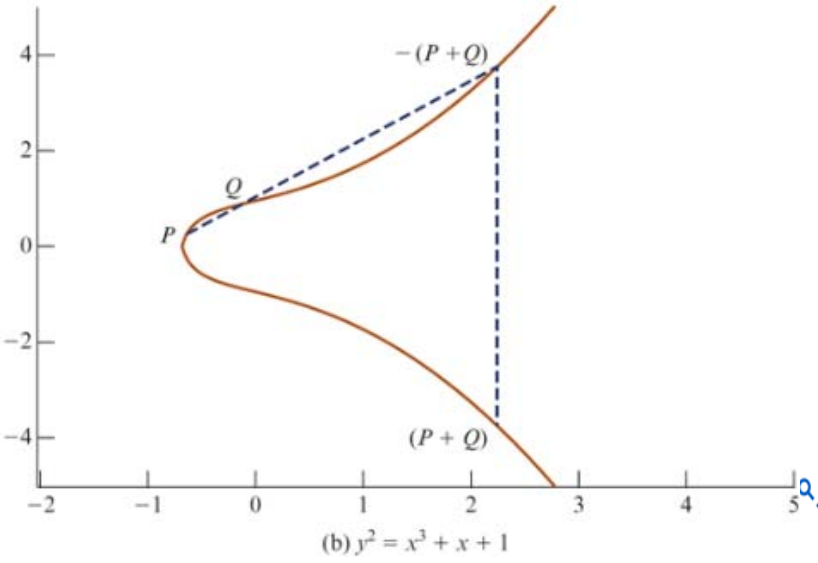
- BN指128位、256位、512位、1024位整数。
- 密钥N为128位即16字节的情况下，明文长度必须是16字节，并且明文的值一定要小于N，密文长度也只能是16字节。
- 设明文char m[]={'A','B','C','D',0,0,…}（共12个0）
- 则实际上该明文是被当成如下这个大数来处理：
- 0x41424344000000000000000000000000

椭圆曲线算法（ECC算法）

- 椭圆曲线(Elliptic Curve) 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。



(a) $y^2 = x^3 - x$



(b) $y^2 = x^3 + x + 1$

ECC算法的数学基础 - 椭圆曲线在素域Z_p上的运算规则

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

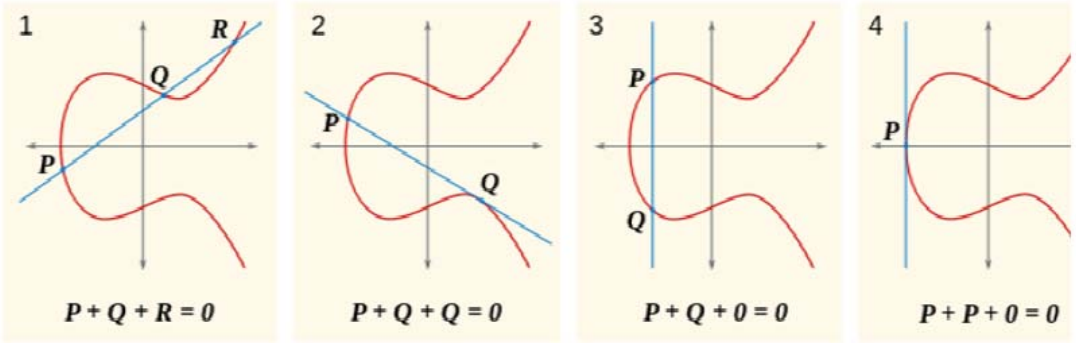
(2) 如果$P=(x_1, y_1)$，$Q=(x_2, y_2)$，且有$x_1=x_2$及$y_1=y_2=0$，或有$x_1=x_2$及$y_1=-y_2\neq 0$，则P+Q=O；

(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，$\lambda=(3x_1^2+a)/(2y_1)$；



ECC算法的数学基础 - Euler准则

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

ECC算法的数学基础 - 点加运算

- 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。

用ECC算法加密解密 -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

用ECC算法加密解密 - 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

用ECC算法加密解密 – 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

用ECC算法加密解密 – 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

用ECC签名验证-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。

用ECC签名验证 - ecnr(Elliptic Curve Nyberg-Rueppel Signature)

- (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

证明gcd(n, u)=an+bu

- 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)

证明gcd(n, u)=an+bu

- 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：

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

$x_i = a2_i * n + b2_i * u$ (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。

证明gcd(n, u)=an+bu

- 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。

证明Euler准则

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

证明Euler准则

- (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$Z_p=\{0, 1, 2, \cdots, p-1\}$是有限域，$Z_p^*=\{1, 2, 3, \cdots p-1\}$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$\pm b^{i/2} \bmod p$。

证明题

- 证明RSA
- 证明中国剩余定理
- 证明Euler准则
- 证明gcd(n, u)=an+bu

Thank you!