密码学复习整理

考试题型

单选题 20分 10道

简答题 20分 4题

计算题 40分 3题

证明题 20分 2题

考试内容

Enigma, md5, sha, rc4, ecb, cbc, cfb, **des, aes, rsa, ecc**【4大算法】

ecc 加密解密，两种签名

公式不用背（比如ECC的签名得到的r、s的公式会提供，但是RSA不会给），会计算就行

Enigma一道计算题，最多敲两下键盘

md5考初始化的过程（结构64位的长度）/ 分块，填在最后的报文的位数是不包括填充内容在内

简答题：md5报文填充过程\设计一个函数，十行左右搞定\

des只能考sbox，进去啥出去啥

md5和sha对照，散列长度不同，填充相同

rc4基本原理看看

分块ecb（每次加密八个字节） cbc（每次加密八个字节） cfb（每次加密一个字节）

要从程序（写程序实现加密）

aes的矩阵运算，旋转，乘法和除法 3112 BD9E

密钥生成 三种宽度的生成代码看看

证明题 - 四选二

- a,b为整数，ab≠0，令d=gcd(a,b),一定存在整数x,y，使d=ax+by

设 $d = gcd(a,b)$,则 $d|a, d|b$。由整除的性质，$\forall x, y \in Z$，有d|(ax+by)。

设s为ax+by最小正值，令 $q = \lfloor \frac{a}{s} \rfloor$，则, $r = amods = a - q(ax+by) = a(1-qx) + b(-qy)$。

可见 r 也为 a, b 的线性组合。

由于 r 为 $amods$ 所得,所以 $0 \le r < s$。

由于 s 为 a, b 线性组合的最小正值,可知 $r = 0$。

因此有 $s|a$,同理 $s|b$，因此，s是a与b的公约数，所以d≥s......①。

因为d|a，d|b，且s是a与b的一个线性组合，所以由整除性质知d|s。

但由于d|s和s＞0，因此d≤s......②。

由①②得d=s，命题得证

- 欧拉准则

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

  比如1,2,4是模7的平方剩余 $1^3=1 \bmod 7$ $2^3=1 \bmod 7$

  比如3,5,6是模7的非平方剩余：$3^3= -1 \bmod 7$

  必要性这里，gcd(y,p)要么是1要么是p，若为p则p应整除y^2，模不会为x，故gcd(y,p)=1

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 $\pm b^{i/2} \bmod p$。

- 中国剩余定理

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

$$\underbrace{\begin{cases} x \equiv a_i \pmod{m_i},\ i=1,2,\ldots,r \\ M = m_1 m_2 \ldots m_r \end{cases}}_{(a)}$$

的模 M 唯一解为 $x = \left(\sum_{i=1}^{r} a_i M_i (M_i^{-1} \bmod m_i)\right) \bmod M$ (b)

其中 $M_i = M/m_i$

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

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

(2) 证明是模 M 唯一解，

假设 x_1, x_2 是 (a) 的两个不同解，

$x_1 = a_i \bmod m_i$，$x_2 = a_i \bmod m_i$ $(i=1, 2, \ldots r)$

$x_1 - x_2 = 0 \bmod m_i$

$\therefore m_i \mid (x_1 - x_2)$

$\because m_i$ 两两互素 $\therefore M \mid (x_1 - x_2)$

$\therefore x_1 = x_2 \bmod M$ $\therefore$ (b) 是 (a) 模 M 的唯一解

- RSA算法证明过程

$c = m^e \bmod n$

证明 $m = c^d \bmod n$

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

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

① 设 $m = 0 \bmod p$，即 $m = ap$，

则 $m^{ed} = (ap)^{ed} = 0 \bmod p = m \bmod p$

② 设 $m \neq 0 \bmod p$，即 $\gcd(m, p) = 1$

$\therefore m^{ed} = m^{ed-1+1} = m^{ed-1} \cdot m$

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

$= \left(m^{(p-1)}\right)^{k(q-1)} \cdot m$

$\because \gcd(m, p) = 1$

$\therefore$ 令 $a = a' \bmod p$，$b = b' \bmod p$

$ab = (a' + k_1 p)(b' + k_2 p)$

$= a'b' + k_1 k_2 p^2 + (k_1 a' + k_2 b')p$

$\therefore ab = a'b' \bmod p$ $\quad \because m^{\phi(p)} = 1 \bmod p$，即 $m^{p-1} = 1 \bmod p$

$\therefore (m^{(p-1)})^{k(q-1)} \bmod p = 1^{k(q-1)} \bmod p = 1 \bmod p$

$\therefore (m^{(p-1)})^{k(q-1)} \cdot m \bmod p = m \bmod p$

$\therefore$ 综上，$m^{ed} = m \bmod p$

$\therefore$ 同理，$m^{ed} = m \bmod q$

$\therefore m^{ed} = k_1 p + m = k_2 q + m$

$\therefore k_1 p = k_2 q$

$\therefore k_1 p = 0 \bmod q$，$k_2 q = 0 \bmod p$

$\because \gcd(p, q) = 1$

$\therefore k_1 = aq$，$k_2 = bp$

$\therefore m^{ed} = apq + m$

$\therefore m^{ed} = m \bmod (p*q)$

$= m \bmod n$

第一章 数学基础

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**

素数的定义：整数 p 只有因子 ±1 和 ±p，则 p 为素数

互素的定义：对于整数a、b，**若gcd(a,b)=1，则称a、b互素。**【gcd最大公约数】

gcd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 a*x + b*y = d。**【当a、b互素时，则一定存在整数x、y使得a*x+b*y=1成立】**

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，记作: a ≡ b (mod n)

- 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)

加法模逆元的定义：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

乘法模逆元的定义：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}。

- 怎么求乘法模逆元（辗转相除法）

  一个整数a有乘法模逆元X（a * X ≡ 1 mod N）的充要条件是 gcd(a,N) = 1，则存在X和Y，使得 a * X + N * Y = 1 成立。

  举例说明：求13模35的乘法逆元，即求13*x + 35*y = 1中的y

  ① 35=2*13+9　② 13=1*9+4　③ 9=2*4+1

  1 = 9-2*4 = 35-2*13 – (13-1*9)*2　【利用①将9换掉，利用②将4换掉】

  = 35-2*13 – (13-(35-2*13))*2　【继续利用①将9换掉】
  = (35-2*13)*3 – 13*2　【将35的和13的合并同类项】

  = 35*3 – 8*13

  解得：x=-8，y=3
  其中-8 ≡ (-35+27) ≡ 27 (mod 35)　【在模35的情况下，-8和27等价】

第二章 古典密码

encrypt加密 - decrypt解密 - C密文 - M明文

单表密码：明文字母和密文字母有固定对应关系，即相同明文对应相同的密文，可以用频率分析攻击。

多表密码：每个明文用不同的单标代替，即同一明文对应不同的密文

- demo - Vigenere

  C = (M+k_i) % n ；　M = (C-k_i) % n

  n是26，k是密钥的位置　19+2%26=21；19+15%26=8

  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,...)

加密密码：demo - 凯撒密码 C=(M-'A'+3)%26+'A'　密文=明文+3

乘法密码：C=M * K % N ； M=C * K^{-1} % N 【K和N一定互质，不然K没有乘法逆元】

仿射密码：C = (M*k_1 + k_2) % n ； M = (C-k_2)*k_1^{-1} % n （C+k2的加法逆元）x(k1的乘法逆元)%n

enigma【重点】

部件

- 接线板**plugboard**：可以**人为**的连接X对字母，使得连接起来的字母对交换（例如，A和B连接，则人输入‘A’，进入机器的实际上是‘B’
- 5个齿轮：每个齿轮有一个ringsetting（内部初始状态，在齿轮转动时不会更改）和messagekey（可以**人为**设置且会随着齿轮转动更改），这两个是用来计算△的（messagekey - ringsetting）；有一张表，可以根据输入字母查到真实输入字母
  - ```
    2. rotor I
    char rotor[27]="EKMFLGDQVZNTOWYHXUSPAIBRCJ";
                    // ABCDEFGHIJKLMNOPQRSTUVWXYZ
    3. rotor II
    AJDKSIRUXBLHWTMCQGZNPYFVOE
    ABCDEFGHIJKLMNOPQRSTUVWXYZ

    4. rotor III
    BDFHJLCPRTXVZNYEIWGAKMUSQO
    ABCDEFGHIJKLMNOPQRSTUVWXYZ

    rotor IV

    ESOVPZJAYQUIRHXLNFTGKDCMWB
    ABCDEFGHIJKLMNOPQRSTUVWXYZ

    rotor V
    VZBRGITYUPSDNHLXAWMJQOFECK
    ABCDEFGHIJKLMNOPQRSTUVWXYZ
    ```

    注意：当数据流向是从右往左的时候，即数据进来的时候，输入的数据是下面这行，然后把上面那行对应的字母传给下一个齿轮/反射板；当数据流向从左往右的时候，即数据出去的时候，输入的数据是上面那一行，然后把下面那行对应的字母传出去
- 反射板：和接线板差不多【那一行作为输出都可以，因为是两两交换的】
  - char reflector[27]="YRUHQSLDPXNGOKMIEBFZCWVJAT";
    //   ABCDEFGHIJKLMNOPQRSTUVWXYZ
  - **反射板上的映射是两两交换的。由于反射板上没有一个字母对应自己（但其他齿轮上允许存在自己对应自己的情况），所以一个明文字母的密文永远不会是自己【这是个缺席】。**

    S<======= A   F<========W

    | |

    V   V

    F========>W   S========> A

    如果W变成A，那么输入A，他到反射板的时候既有可能是S，又有可能是F，这样就不对了。

流程（举例说明）

- 分别使用III、II、I号齿轮【左----右；输入从右边进入】
  齿轮设定为：
  CBA ringsetting内部
    ```

EDB　messagekey外部
输入A后，外部变成EDC【是先转动齿轮，再计算△，再查表的】
因此三位的Δ=222，
输入A（这边假设接线板上A没有接线。如果接线板上A和B之间有线，那么加上△1的是B），加上Δ1，得到I号轮真实输入A+Δ1=C，进入I号轮，**(查下面那行，输出上面)**得到输出M，减去Δ1得到I号轮真实输出M-Δ1=K；K加上Δ2，得到II号轮真实输入M，进入II号轮，得到输出W，再减去Δ2，得到II号轮真实输出U；U加上Δ3，得到真实输入W，进入III号轮，得到输出U，减去Δ3得到真实输出S；进入反射板，得到F；加上Δ3，得到真实输入H，**(查上面那行，输出下面)** III号轮输出D，减去Δ3得到B；B加Δ2得到D，进入II号轮，得到C，C减去Δ2等于A；A加上Δ1得到C，进入I号轮，得到Y，Y减去Δ1等于W。最后亮起来的灯泡是字母W（也假设接线板上W没有接线），W是密文。

齿轮转的问题

- 卡口：当前齿轮处于卡口时，会带动更高位的齿轮旋转。

  Q E V J Z; 齿轮的当前位置,从左到右对应齿轮I II III IV V
  R F W K A; 齿轮的下一步位置

  举例：① I号轮从Q到R的时候，II会转

  ② 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。

- double stepping

  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)，同时由于II从E变成F，所以还会带动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（进位的位置）, I不管在什么位置
  注：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。若上述两个条件同时满足，II只会转1次**

messagekey的传递

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。接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。

第三章 hash函数

MD5：不论文件多长，都得到**128位**的hash值

- 碰撞：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)。
- md5碰撞可以达到骗取数字签名的目的。
  - 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。
    - 计算md5(m1) == md5(m2)碰撞时，md5的种子应该等于md5(letter), 而且md5(letter)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。
- **md5计算过程中的分块和填充**

- md5分块计算，每块大小是**64字节**
- 当最后一块大小n小于64字节时（当原文件大小整除64B时，最后一块视作0字节），要按以下步骤补充数据凑成64字节（**一开始先凑到56个字节，然后再加8个字节表示message总共的位数**）
  - 步骤一（选项一）：假定块大小n<56，即[0,56))字节, 则在末尾补上以下数据:
    0x80 0x00 0x00 0x00 ... 0x00; 共**56-n**个字节
    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。（其实就是填了一个1，后面全是0）
  - 步骤一（选项二）：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（相当于补到下一个块里了）。例如当n==56时, 应该补上64个字节; 当n==57时, 应该补上63个字节。**补的还是0x80、0x00、0x00、...、0x00**。【这是规定，必须填充至少一个字节**的填充物，所以正好56时就要填到下一个块里了】
  - 步骤二：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)，单位为**bit**。这8个字节以**小端**格式保存。
  - 位数！！！小端！！！
- 举例:

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

再如，长度为 55 字节时，就在原文后面一个字节的 0x80，再加 B8 01 00 00 00 00 00 00。其中 01B8h=440=55*8。

- md5

```c
typedef struct _MD5_CTX {
        unsigned long  state[4]; /* 128位摘要 */
        unsigned long  count[2]; /* 已处理的报文的二进制位数,最大值=2^64-1 */
        unsigned char  data[64]; /* 64字节message块 */
}MD5_CTX;
int Init_MD5(MD5_CTX *MD5_ctx);
int Update_MD5(MD5_CTX *MD5_ctx, unsigned char *buffer, unsigned long count);
int Final_MD5(MD5_CTX *MD5_ctx);
```

  - init：count 归零，state设置初始值
  - update：增加count值即报文位数，分块（64字节）计算并更新到state，多余的存在data里等待之后的填充
  - final：为最后一块填充若小于56字节则补充到56，若大于补充完当前块并补充到下一个56字节，最后增加8字节报文长度

SHA

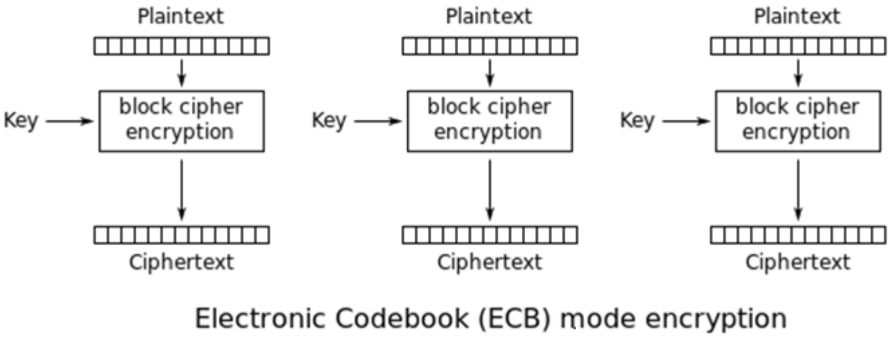
- sha-1散列算法计算出来的hash值达160位，即20字节，比md5多了32位（md5 16个字节，128位）。
- sha-1也是分块计算，每块也是64字节，当最后一块不足64字节也按照md5的方式进行填充。数据块的最后一定要补上表示报文总共位数的8个字节。
- 大端存储位数
~~~~~~~~~~~~~~~~

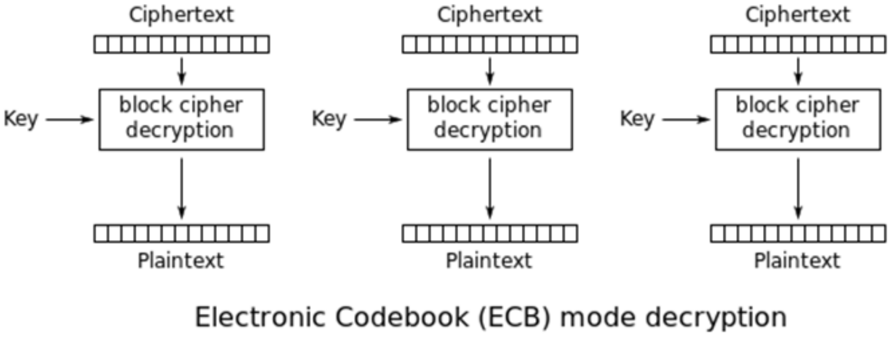
第四章 分组密码工作模式与流密码

分组密码 ECB

把长串明文分成合适大小的block(64bit，8B)，对每块进行并行的加密

- 加密过程：Cj = Ek(Pj)
- 解密过程：Pj = Dk(Cj)
- 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。
- 优点是:加密和解密过程均可以并行处理。

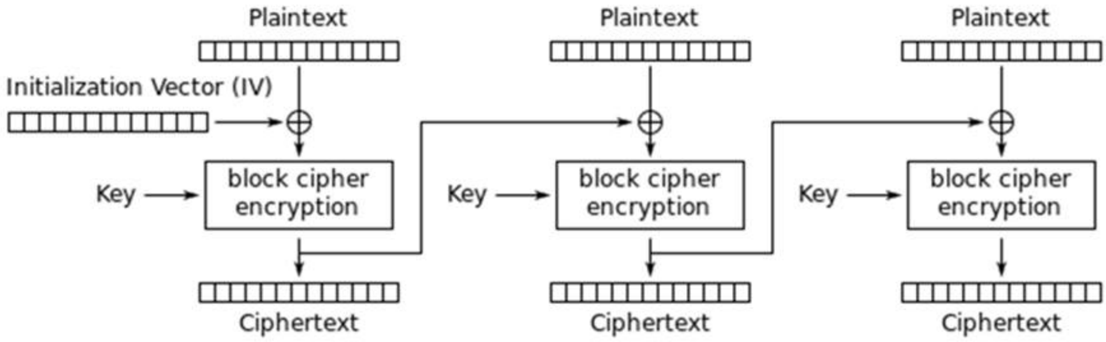




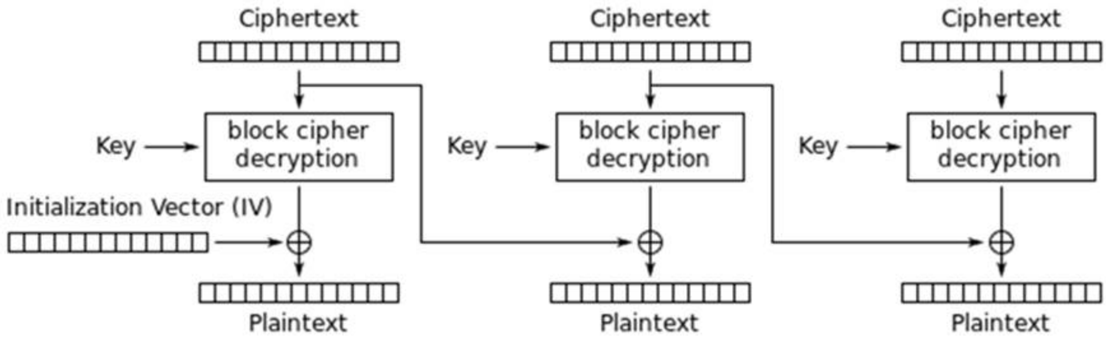
密文块链接模式CBC

把长串明文分成合适大小的block(64bit，8B)，明文、种子异或后再加密，得到密文，并将密文当作下一轮的种子

密文先解密，然后跟种子异或得到明文，种子是上一轮的密文



Cipher Block Chaining (CBC) mode encryption

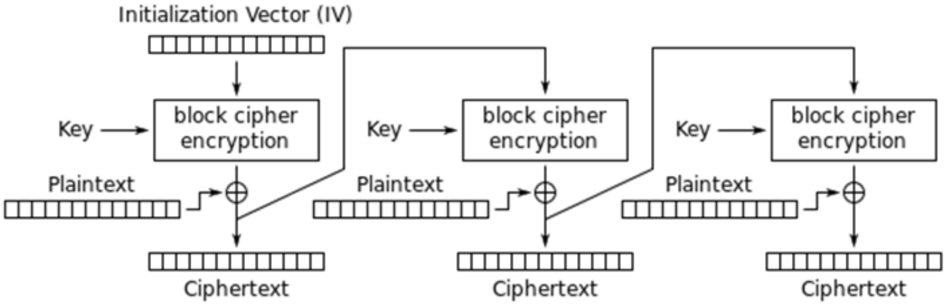


Cipher Block Chaining (CBC) mode decryption

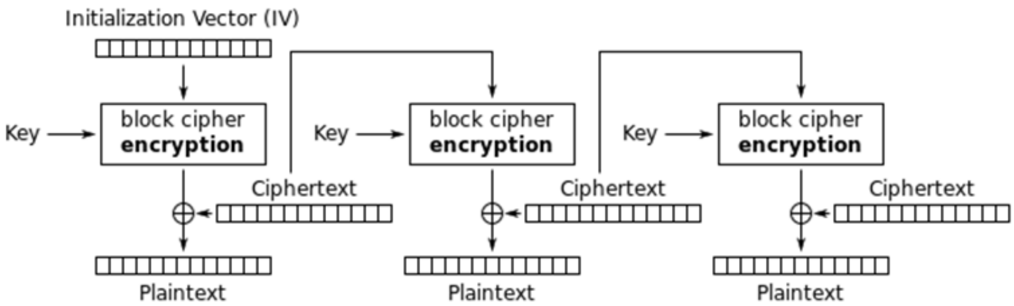
- 将上一个加密的密文当成下一个加密的种子
- C0就是初始化向量IV（宽度和明文一样宽）
- 加密过程： $C_j = Ek(P_j \oplus C_{j-1})$
  解密过程： $P_j = Dk(C_j) \oplus C_{j-1}$
  CBC模式的特点是：当前块的密文与前一块的密文有关;加密过程只能串行处理；解密过程可以并行处理，C_i解密后与C_{i-1}异或即可；不同位置的相同的明文产生的密文是不一样的。

密文反馈模式 CFB

种子加密后，拿第一字节和明文异或，得到密文（8 bit）。原种子左移8位，把密文拼到原种子右边形成新种子



Cipher Feedback (CFB) mode encryption



Cipher Feedback (CFB) mode decryption

流模式，明文被分成8位的小块

- CFB（密文反馈模式）**每次加密一个字节**。种子数是一个**64位**的数。种子数加密后，取种子的第一字节，与明文异或，产生一个字节的密文。加密下一段密文时，把种子数左移8位，拼接上刚才加密产生的密文（1个字节），产生新的种子数。
- **每次把iv[0]移出去，然后iv[i]用iv[i+1]代替，iv[7]用密文代替**
- 加密过程：
  - $C_j = P_j \oplus L_8(E_k(X_j))$　X_j实际是一个64位种子数
    $X_{j+1} = R_{56}(X_j) \| C_j$
  - L_8是取低8位的意思，R_{56}是取高56位的意思，||是拼接的意思，X的高56位放在低56位上，C_j放在高8位上
  - ```
    //des_iv是64位的种子
    for(i=0;i<8;i++) iv[i]=des_iv[i];
    for(i=0;i<n;i++)
    {
        des_encrypt(des_iv, des_key);
        //通过异或进行加密
        p[i] ^= des_iv[0];
        for(j=0;j<7;j++) iv[j]=iv[j+1];
        iv[7]=p[i];//左移8位（1字节）

        //得到新的des_iv
          for(j=0;j<8;j++) des_iv[j]=iv[j];
    }
    ```
- 解密过程：
  - $P_j = C_j \oplus L_8(E_k(X_j))$

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

- 优点：**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P1有错。若X1记作（*, *, *, *, *, *, *, *），则
  X2 = (*, *, *, *, *, *, *, C_1')
  X3 = (*, *, *, *, *, *, C_1', C_2)
  X4 = (*,*,*,*, *, C_1', C_2, C_3)
  ...
  X9 = (C_1', C_2, C_3, C_4, C_5, C_6, C_7, C_8)
  X10 = (C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9)
  使用密钥X1,X2,X3,...,X9解密C1'\C2\C3\C4\C5\C6\C7\C8\C9还原得到的P1,P2,P3,...,P9全部有错，但是从P10开始的解密还原是正确的。即：**错误只会影响局部解密，并不会导致全部解密错误。**

流密码算法RC4

```c
//加密和解密算法相同，RC4运行速度很快，但是不安全
//程序开始
#include<stdio.h>
#include<string.h>
typedef unsigned long ULONG;

/*初始化函数*/
//在初始化的过程中，密钥key的主要功能是将S-box s搅乱，i确保S-box s的每个元素都得到处理，j保证S-box的搅乱是随机的
void rc4_init(unsigned char*s, unsigned char*key, unsigned long Len)
{
    int i = 0, j = 0;
    char k[256] = { 0 };
    unsigned char tmp = 0;
    for (i = 0; i<256; i++)
    {
        s[i] = i;
        k[i] = key[i%Len];
    }
    for (i = 0; i<256; i++)
    {
        j = (j + s[i] + k[i]) % 256;
        tmp = s[i];
        s[i] = s[j];//交换s[i]和s[j]
        s[j] = tmp;
    }
}

/*加解密*/
//s是sbox，data是明文，len是明文长度
void rc4_crypt(unsigned char*s, unsigned char*Data, unsigned long Len)
{
    int i = 0, j = 0, t = 0;
    unsigned long k = 0;
    unsigned char tmp;
    for (k = 0; k<Len; k++)
    {
        i = (i + 1) % 256;
        j = (j + s[i]) % 256;
        tmp = s[i];
        s[i] = s[j];//交换s[x]和s[y]
        s[j] = tmp;
        t = (s[i] + s[j]) % 256;
```

```
        Data[k] ^= s[t];
    }
}

//加解密要用同一个sbox
int main()
{
    unsigned char s[256] = { 0 }, s2[256] = { 0 };//S-box
    char key[256] = { "justfortest" };
    char pData[512] = "这是一个用来加密的数据Data";
    unsigned long len = strlen(pData);
    int i;

    printf("pData=%s\n", pData);
    printf("key=%s,length=%d\n\n", key, strlen(key));
    rc4_init(s, (unsigned char*)key, strlen(key));//已经完成了初始化
    printf("完成对S[i]的初始化，如下：\n\n");
    for (i = 0; i<256; i++)
    {
        printf("%02X", s[i]);
        if (i && (i + 1) % 16 == 0)putchar('\n');
    }
    printf("\n\n");
    for (i = 0; i<256; i++)//用s2[i]暂时保留经过初始化的s[i]，很重要的！！！
    {
        s2[i] = s[i];
    }
    printf("已经初始化，现在加密:\n\n");
    rc4_crypt(s, (unsigned char*)pData, len);//加密
    printf("pData=%s\n\n", pData);
    printf("已经加密，现在解密:\n\n");
    //rc4_init(s,(unsignedchar*)key,strlen(key));//初始化密钥
    rc4_crypt(s2, (unsigned char*)pData, len);//解密
    printf("pData=%s\n\n", pData);
    return 0;
}
```

第五章 DES算法和AES算法

DES：**明文和密文64位**（8字节），**密钥64位**（8字节）【但是密钥每个字节都被去掉1位，所以事实上是56位】；加密和解密的密钥相同

流程：

每轮示意图：

最开始：

- 对明文64位打乱；对密钥64位砍成56位。【查表key_perm_table】

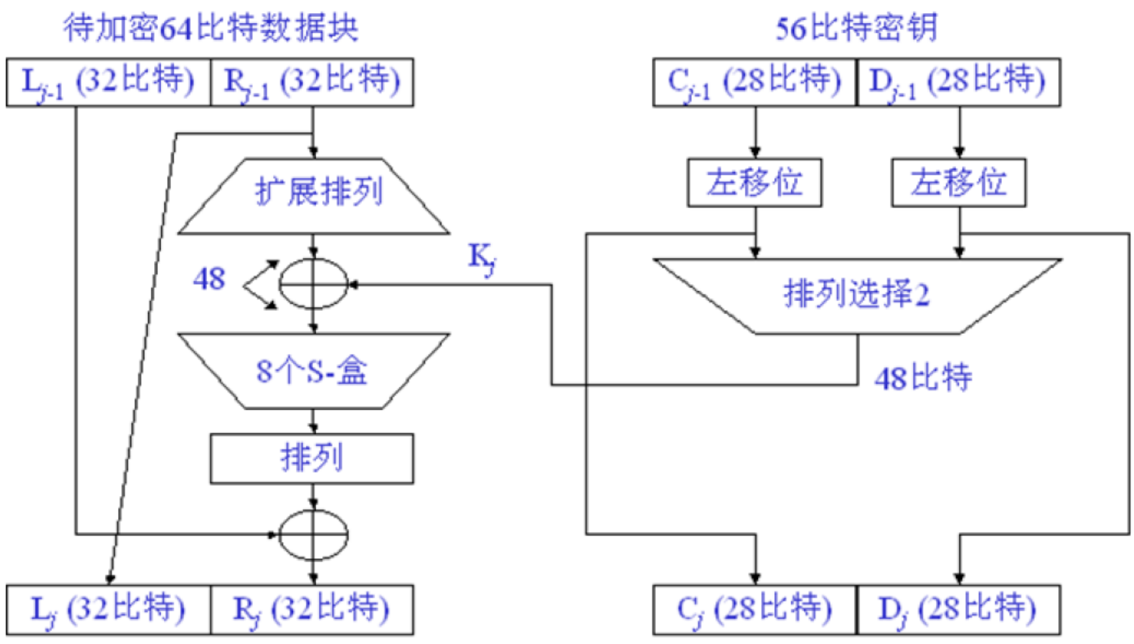


图2.6 DES算法每轮处理过程

- 每一轮：（明文的左右每轮需要互换）
  - 明文分成左右32位，复制右32位成为下一轮的左32位，然后查表将右32位拓展成48位；
  - 打乱56位密钥，分成左右28位，根据表格进行循环左移位1或2位，成为下一轮的密钥。
  - 根据表格对密钥进行排列选择，得到48位【用表：key_56bit_to_48bit_table】
  - 两个48位进行异或，分成8组6位，进入sbox，得到8组4位，共32位
  - 对sbox得到的32位结果打乱【sbox_perm_table】
  - 这32位与左32位异或，得到下一轮的右32位；
- 出去的时候：
  - 左右互换
  - 打乱密文【fp】
- DES算法中用的明文需按照**大端**规则放置

```
//输入是r（按照小端，数值低位放地址低位），要把他按照大端放在s里
for(i = 0;i < 4;i++) strcpy(s+i,(unsigned char*)(&r) + (3 - i));
```

- 明文被分成L32和R32，然后32位需要扩展成48位，怎么扩？
  - plaintext有8个字节（每个字节左边2位恒为0，所以有用的实际上只有48位）
  - 其中plaintext是目标数组（下标从0开始），s是源数组（下标从1开始）。假设i=0，j=0的时候，index=8，则其实是

    目标[0*6+0]=源[8-1]（以bit为单位） **【凡是用循环查找做打乱都是这么做的】**
  - 代码实现：ip[0]=58，表示目标第0位=源第(58-1)=57位

    if(p[57>>3]&(1<<(7-57&7))) t[0]|=(1<<7-0);

    //加减法比移位优先级高

    //第0位其实在最左边（这边的位都是从左往右数的）

```
//根据plaintext_32bit_expanded_to_48bit_table这张表，把s中的4字节共32位扩展成48位并保存到数组plaintext中；plaintext每个元素的左边2位恒为0，右边6位用来保存数据32位转48位的过程要求使用双重循环来做，外循环8次，内循环6次，其中内循环每次只提取1位;注意提取某1位的时候只能从数组s中取(要计算该位是第几个字节中的第几位)，不得从参数r中获取；判断某位是否为1的时候可以使用表bytebit。
//左边两位恒为0，指每个字节的低两位都是0，所以下面是j+2
```

```c
memset(plaintext, 0, sizeof(plaintext));
for (int i = 0; i < 8; i++) {
    for (int j = 0; j < 6; j++) {
        int index = plaintext_32bit_expanded_to_48bit_table[6 * i + j];
        index--;
        int byte, bit;
        byte = index / 8;
        bit = index % 8;
        if (s[byte] & bytebit[bit]) {
            plaintext[i] |= bytebit[j + 2];
        }
    }
}
```

- 用循环做打乱比较慢，打乱64位就要做64次循环，所以可**事先生成表来加快打乱**。
  根据打乱用的那张表建立一张新的表，假设是char t[16][16][8]，第一个16表示有16组（每组4位，共64位），第二个16表示4个位的16种变化（0~15），8表示单独将这4位打乱后得到的64位结果**（通过查表的方式将4位打乱，得到的是64位）**。
  将64位源分为16组，每组4位，记为a[0]a[1]a[2]...a[15]，第i组打乱的结果是t[i][a[i]][0~7]。比如，

  假设64位数据为1011 0110 1001 1111 ...
  假设t[0][11]为 xx1x01x1　xxxxxxxx　...(8个字节)
  假设t[1][6]为　xxxxxxxx　x0xx11xx　xx0xxxxx ...(8个字节)

  注：上面的x均为0。
  注：对所有的t[i][0~15]，其中x的位置都是确定的。若t[0][11]为xx1x01x1 xxxxxxxx ...，则t[0][12]为xx1x10x0 xxxxxxxx ...。
  注：t[0]~t[15]的非x位（每一个有4个非x位）分别在不同的位置(共64个位置)上（每一组（4位一组的那个）得到的64位结果中非X位分别在不同的位子，并且合起来看64个位子正好全被占且仅被占1次（互不重叠）。因此，将得到的16组64位数或一下，就得到打乱后的结果。
- 明文加密后也是48位，然后需要缩减为32位，怎么缩减?
  - 精简版：**进6位出4位（8组6位，共48位，出来32位）**
  - **给出6位二进制制，第一位和最后一位拼起来作为行号，中间4位作为列号，读取一个4位二进制数。**输入的流为48位，拆为8部分，每部分查一张表（总共有8个sbox，虽然放在一个数组里面），共得到4*8=32位数
  - sbox的第一个下标是代表这是第几组6位输入，第二个下标是 行号*16+列号，然后根据这个index，得到一个4位二进制数

```c
/*
static char sbox[8][64] =
{
   /* 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,
   ......
*/
```

```
//plaintext[i]&1 代表最低位（第0位）
//plaintext[i] >> 5 &1 代表第五位
//第五位乘2+第0位是行号
//plaintext[i] >> 1 & 0x0f 代表中间的4位，列号
//行号*16+列号
m[i]=sbox[i][(((plaintext[i] & 1) + ((plaintext[i] >> 5 & 1) << 1))*16)
+ (plaintext[i] >> 1 & 0x0f)];
```

- 重要函数
  - 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])　　//生成打乱表
    (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)　//通过查表，将输出打乱
    (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。
    (4) 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分组加密模式:ecb, cbc, cfb
  - ecb

```
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
}
```

  - cbc

```
c[0] = des_encrypt(block[0] ^ iv);
c[1] = des_encrypt(block[1] ^ c[0]);
//c[0]和c[1]是密文，iv是8字节的初始化向量
```

- cfb 把加密过的左边1字节放到没加密的种子右边

```
//iv是8字节的初始向量
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 */
iv = iv << 8 | c[1]
```

AES算法

先把明文按列排好，然后做行的循环左移（第0行移0个字节，。。。第i行移i个字节），然后用3 1 1 2的矩阵（行的循环左移，第i行移3-i个字节）跟每一列进行矩阵乘法，结果按行存放（第i列结果存第i行）。矩阵乘法的每一项均为8位数乘法（转成多项式乘然后mod($x^8+x^4+x^3+x+1$)，其中除的时候-可以直接转换成+）,余式转成二进制再转成16进制即得每一项的乘法答案，四项八位数乘法结果的相加为异或，得到的结果即为目标矩阵的一个值

- **明文长度 = 密文长度 = 128位（16字节）**
- **密钥长度 128位（16字节） 192位（24字节） 256位（32字节）**

密钥长度	**128位（16字节）**	**192位（24字节）**	**256位（32字节）**
key_rounds	10	12	14

- AES算法加密流程

```c
void aes_encrypt(unsigned char *bufin, unsigned char *bufout, unsigned char *key)
{
   int i;
   unsigned char a[4] = {0x03, 0x01, 0x01, 0x02}; /* 定义多项式 3*X^3+X^2+X+2 */
   unsigned char matrix[4][4];
   memcpy(matrix, bufin, 4*4); /* 复制明文16字节到matrix */
   AddRoundKey((unsigned char *)matrix, key); /* 第0轮只做AddRoundKey() */
   for(i=1; i<=key_rounds; i++)
   {  /* 第1至key_rounds轮, 做以下步骤:
        ByteSub, ShiftRow, MixColumn, AddRoundKey */
      ByteSub((unsigned char *)matrix, 16);
      MixColumnInverse((unsigned char *)matrix, a, 0);
       /* 不做乘法, 只做矩阵行转列 */
      ShiftRow((unsigned char *)matrix);
      if(i != key_rounds)
         MixColumn((unsigned char *)matrix, a, 1); /* do mul */
      else
         MixColumn((unsigned char *)matrix, a, 0); /* don't mul */
       //只做列转行
      AddRoundKey((unsigned char *)matrix, key+i*(4*4));
   }
   memcpy(bufout, matrix, 4*4); /* 密文复制到bufout */
}
```

- sbox：各不相同的256个的char类型元素数组
- **ShiftRow(p)**：对明文16字节构成的4*4矩阵逐行做循环左移（1~F只是代表数组的下标）

  0 4 8 C; 不移动　　　　0 4 8 C
  1 5 9 D; 循环左移1次　　5 9 D 1
  2 6 A E; 循环左移2次　　A E 2 6
  3 7 B F; 循环左移3次　　F 3 7 B

- MixColumn 把列取出来做乘法运算，然后以行的形式保存回p中，加密用3112,解密用BD9E **列转行**

```
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) 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;
      (6) 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, 仅做列转行;
    */
   unsigned char b[4];
   unsigned char t[4][4];
   int j;
   for(j=0; j<4; j++)
   {
      get_column(p, j, b); /* 从p所指矩阵m中提取第j列, 保存到数组b中. */
      if(do_mul) /* 在加密最后一轮以及解密第一轮的MixColumn步骤中不需要做乘法; */
         aes_polynomial_mul(a, b, b); /* 其余轮都要做乘法: b = a*b mod (X^4+1); */
      memcpy(t[j], b, 4); /* 把乘法所得结果复制到t中第j行 */
   }
   memcpy(p, t, 4*4); /* 复制t中矩阵到p, 替换掉p中原有矩阵 */
}
```

  - **加密使用的被乘数a是{3,1,1,2}。手动模拟矩阵乘法过程**

    char plain[16]={4,3,2,1,11,22,33,44,...}
    $(3x^3 + x^2 + x + 2) * (x^3 + 2x^2 + 3x + 4) \bmod (x^4 + 1)$
    (系数相乘) =
    2 3 1 1　　4　　0次 3次 2次 1次　　0次　　　　　　0次+4次+4次+4次　0次
    1 2 3 1　*　3　　1次 0次 3次 2次　*　1次　相乘得到：　1次+1次+5次+5次　经过mod x^4+1后，得到：　1次
    1 1 2 3　　2　　2次 1次 0次 3次　　2次　　　　　　2次+2次+2次+6次　2次
    3 1 1 2　　1　　3次 2次 1次 0次　　3次　　　　　　3次+3次+3次+3次　3次

    - 上面解释了这么排的合理性
    - mod $(x^4 + 1)$的作用，就是将6次变2次，5次变1次，4次变0次
  - 上面矩阵相乘得到的结果

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

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

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

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

注意：**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

- **8位数乘法mod 0x11B**是什么？
  - 8位数乘法实质上是多项式乘法 mod ($x^8+x^4+x^3+x+1$) 不可约就行，作者选择了这个（为了有乘法逆元）

    乘数的每一个bit都是多项式某一项的系数

    举例：

    求1000 1000（x^7+x^3）* 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：

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

    把余式转化成二进制就是：

    1001 1110

    结论：

    0x88 * 0x05 = 0x9E mod 0x11B

    - 可以发现4次、2次和1次前面本来都应该是 - 号，但是这边都是 + 号。是因为在$GF(2^8)$中，a的加法逆元还是a，所以正负没有区别
  - 也可以用**农夫算法**，计算P=X*Y mod 0x11B

    重复八次
  - 先判断Y最低位是否为1
    - 为1，则P=P^X
    - 为0，无事发生
  - Y右移一位
  - 判断X最高位是否为1，X左移一位（有九位）
    - 为1，X=X^0x11B
    - 为0，无事发生
- x左移进位后为什么要做x = x ^ 0x11B？

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。

- 练习题：用农夫算法分步计算0x05 * 0x43 mod 0x11B【结果是 0x05 * 0x43 = 0x54 mod (x8+x4+x3+x+1)】
- **解密用的被乘数a是{0x0B, 0x0D, 0x09, 0x0E}**，矩阵为

  E B D 9

  9 E B D

  D 9 E B

  B D 9 E

- 密钥的拓展

  以128位种子密钥为例, 假定当前机器为大端，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:

```
种子密钥是k[0]~k[3] 一个long 32位
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(进去8位出来8位)
r = 2^((i-4)/4) mod 0x11B; 计算轮常数r, 其中i是k的下标
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[i]跟k[i-4]异或
k[5] = k[4] ^ k[1]; //k[i]=k[k-1]^k[i-4]
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。

- 不同长度密钥的生成

  根据种子密钥生成真正密钥:
    128bit种子密钥(16字节): 生成(1+10)*4个long32, step=4, loop=10
    192bit种子密钥(24字节): 生成(1+12)*4个long32, step=6, loop=8
    256bit种子密钥(32字节): 生成(1+14)*4个long32, step=8, loop=7

```
 memcpy(key, seed_key, bits/8);
   pk = (ulong32 *)(key + 4*step);
   for(i=step; i<step+step*loop; i+=step)
   {
      unsigned int r;
      /* 假定生成的密钥k是long32类型的数组, i是其下标,
         则当i!=0 && i%step==0时, k[i]在计算时必须做以
         下特殊处理:
       */
      pk[0] = pk[-1];
```

```
        rol_a_row(key+i*4, 1);
        ByteSub(key+i*4, 4);
        r = 1 << ((i-step) / step);
        if(r <= 0x80)
            r = aes_8bit_mul_mod_0x11B(r, 1);
        else
            r = aes_8bit_mul_mod_0x11B(r/4, 4);
        key[i*4] ^= r;
        pk[0] ^= pk[-step];
        for(j=1; j<step; j++)
            /* i+j是密钥k的下标, 当(i+j)%step != 0时,k[i+j]只需做简单的异或处理 */
        {
            if(step == 6 && i == step*loop && j>=step-2 || step == 8 && i == step*loop && j>=step-4)
                break; /* 128-bit key does not need to discard any steps:
                          4 + 4*10 - 0= 4+40 = 44 == 4+4*10
                          192-bit key should discard last 2 steps:
                          6 + 6*8 - 2 = 4+48 = 52 == 4+4*12
                          256-bit key should discard last 4 steps:
                          8 + 8*7 - 4 = 4+56 = 60 == 4+4*14
                        */
            if(step == 8 && j == 4) /* 对于256bit密钥, 当(i+j)%4==0时需做特殊处理 */
            {
                ulong32 k;
                k = pk[3];
                ByteSub((unsigned char *)&k, 4); /* k = scrambled pk[3] */
                pk[4] = k ^ pk[4-8];
            }
            else /* 当(i+j)%step != 0时, k[i+j]只需做以下异或处理 */
                pk[j] = pk[j-step] ^ pk[j-1];
        }
        pk += step;
    } /* for(i=step; i<step+step*loop; i+=step) */
```

第6章 RSA算法

算法简介

```
随机选择2个不相等的大素数：p和q
计算乘积：n=p*q     //当n小的时候，很容易被质因数分解，得到p和q
选择一个数e，使得e和(p-1)*(q-1)互为素数,e是公钥 （这样才能有乘法逆元）
//当p和q很大的时候，很难根据n猜出(p-1)*(q-1)，也就很难根据e来算出d。现在128位的n已经不安全了，1024位的是相对安全的
找到一个d，使得d满足：e*d = 1 mod ((p-1)*(q-1))，d是私钥
【d其实是e在模(p-1)(q-1)的逆元】
公开(e,n)作为RSA的公钥
保留(d,n)作为RSA的私钥
```

加密过程：c=m^e(mod n) 公钥加密，私钥解密

解密过程：m=c^d(mod n) **其中e*d = 1 mod ((p-1)*(q-1))**

- **明文m的长度和n的长度需要保持一致**（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。）

- （私钥d是某个小于1024bits的大整数）。所以RSA很少对一个字节等短明文进行加密（n越小，越不安全）。比如n有1024bit，则m[0]=0x12,m[1]=0x34,……,m[127]=0xff，m实际上被拼成一个大数m=0x1234……ff，他会被当成这个大数来处理

  （**大端规则**）

- m要比n略小，不然解密得到的结果肯定是错误的，mod n只会得到一个比n小的数

  虽然m的长度和n一样，但还是不能保证m比n小，所以引入2种方法

  - 将m[127]=0x00，浪费一个字节，1次只加密127个字节
  - 多保留几个高字节，比如m[127]~m[120]，随机填写（但还是要保证高位较小）。这样做一个相同的明文可以对应不同的密文，增加了扰乱

AES vs 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。

数学基础

- 欧拉函数φ(n)：小于n且与n互素的整数个数

  φ(5) = 4, 因为与5互素的整数有: 1, 2, 3, 4

- 欧拉定理：若gcd(x,n)=1, 则$x^{\varphi(n)}$ = 1 (mod n)。
  例如$3^{\varphi(5)} = 3^4$ = 81 = 1 (mod 5)

- feimat小定理：设p为素数，且gcd(x,p)=1，则x^{p-1} = 1 (mod p)

  根据欧拉定理直接推，因为p为素数时，φ(p)=p-1

- **中国剩余定理（详见证明题）**

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

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

  模$M=m_1m_2m_3\ldots m_r$的唯一解为

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

  其中$M_i = M/m_i$

  使用拓展欧几里得算法算出Mi的逆元

- 欧拉函数的乘法性质：若n1,n2互素, 则φ(n1*n2)= φ(n1)*φ(n2)

  例如: φ(3*5) =φ(3)*φ(5) = 2*4 = 8

- 欧拉函数的乘法公式：φ(n) = n * ∏ (1- 1/p)   p是n的所有不重复的素因子（若重复，只算1次）

  例如: φ(10) = 10 * (1-1/2) * (1-1/5) = 4

RSA应用——注册码

```
(1)软件打开时显示一个机器码
其中机器码     m'=rsa(mac, 作者公钥)
(2)软件作者: mac=rsa(m', 作者私钥)
注册码        sn=(mac, 作者私钥)
(3)软件验证注册码:
              rsa(sn, 作者公钥)==mac
```

RSA应用——数字签名:

- 作用：别人无法假冒A，A也无法抵赖自己发过（因为使用了A的私钥）；消息在传输过程中有无被别人更改

假定A要发一封信给B，信的内容L="Hello, I'm A."
该如何对信进行加密？
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)** 【为什么要先MD5？若直接对信签名，得到的签名会很长，影响传输效率，使用MD5则不论信多长，得到的摘要长度固定】
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)，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)，然后RSA解密得出信件L = RSA(L', B的私钥)

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。

为了对抗旁路攻击，采取下列办法： （保护d）

私钥加密时本来是：$c = m^d \bmod n$

现在转换成以下两步
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

第7章 椭圆曲线算法

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。【a，b为整数，x^3的系数恒为1】

欧拉准则

见证明题

点的代数意义

(1) P+O=O+P=P
(2) 如果P=(x1,y1)，Q=(x2, y2)，且有x1=x2及y1=y2=0，或有x1=x2及y1=-y2≠0，则P+Q=O;
(3) 如果P=(x1,y1)，Q=(x2, y2)，且排除(1)(2)，则 P+Q=(x3,y3)由下列规则决定:

$x3 = \lambda^2 - x1 - x2$
$y3 = \lambda(x1-x3) - y1$
当P≠Q时, $\lambda=(y2-y1)/(x2-x1)$;
当P=Q时, $\lambda=(3x1^2+a)/(2y1)$;

λ为斜率，同一个点就切线（求导），不同的点就斜率

点运算

- **椭圆曲线六要素**

```
椭圆曲线y^2 = x^3 + x + 6 (mod 11)上的点。
                    a=1    b=6    p=11
       加上          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。若余因子的值不为1，则G的阶一定是曲线的阶的质因数。不为1的时候，不大安全（G的阶会变小）。
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)。n必须是素数。基点可以选择满足方程的任意点
通过(n+1)α=α得出阶数n
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。
```

- 点加运算：直接套 点的代数意义 的公式
- 点乘运算，其实就是点加运算 除法换成模p的乘法逆元，减法换成模p的加法逆元

  举例：设α=(2,7), 计算2α=α+α（曲线还是上面介绍六要素里面的那条）

  $\lambda=(3x_1^2+a)/(2y_1)=(3*2^2+1)/(2*7)=13/14=13*14^{-1}=2*3^{-1}=2*4=8 \mod 11$

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

  $y_3 = \lambda(x_1-x_3) - y_1=8*(2-5)-7=8*(2+6)-7=8*8-7=64+4=2 \mod 11$

  因此2α=(5,2)

公钥及私钥 R和G都是公开的

公钥点R=d*G，公钥是个点

私钥d是一个随机数，且d<n,n是G的阶

加解密以及签名

- 加密

  r=(k*G).x   k*G是一个新的点   r是这个点的x坐标

  s=m*(k*R).x mod n   m是明文，R是公钥

  密文包括r和s两部分

- 解密

  m=s/(dr).x r=k*G(一个点)

- 签名

1. ecdsa(elliptic curve digital signature algorithm)

(1) 签名

r = (k*G).x mod n ; k 是随机数，且 k<n

s = (m+r*d)/k mod n ; m 是明文或 hash, d 是私钥

(2) 验证

需要证明这个((m/s mod n)*G+(r/s mod n)*R).x == r

省掉了.x 和 mod n

(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.x+m mod n

s = k-r*d mod n

(2) 验证

r - (s*G+r*R).x mod n == m mod n

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

= r - (kG-rdG+rdG).x = r - kG.x

= kG.x+m-kG.x = m