- 五、DES解密算法
- DES算法的解密算法是相当简单的，算法本身无显著改变，只需将十六个子密钥顺序颠倒过来，即K_{15}，K_{14},…K_0。轮函数F中L和R互换，即：$R_i=L_{i-1}$,$L_i=R_{i-1}\oplus F(L_{i-1},K_{i-1})$。
- 定理3.1：DES的解密算法是正确的。

- 六、安全性分析
- DES存在下列问题：
- (1)有效密钥位只有56bit，在当前计算能力下，不能保证足够的安全度。
- (2)代换部件S盒的设计虽然不是线性的，但也不是随机的，有些密码学家认为可能隐藏了某些陷门。

- (3)存在弱密钥和半弱密钥。DES算法在每轮中都需使用一个子密钥。如果由给定的主密钥k所产生的子密钥$k_1=k_2=\ldots=k_{16}$，就称k为弱密钥。弱密钥有以下四种(16进制)：
- 01 01 01 01 01 01 01 01
- 1F 1F 1F 1F 0E 0E 0E 0E
- E0 E0 E0 E0 F1 F1 F1 F1
- FE FE FE FE FE FE FE FE

- 若由给定的主密钥k所产生的子密钥只有两种形式，且每种都出现8次，称为半弱密钥。
- 半弱密钥成对出现的，且具有下述性质：若k_1和k_2是一对互逆的半弱密钥，x为明文，则有$DES_{k_2}(DES_{k_1}(x))=DES_{k_1}(DES_{k_2}(x))=x$。半弱密钥共有12个，6对，

- 01 FE 01 FE 01 FE 01 FE
- E 01 FE 01 FE 01 FE 01
- 1F E0 1F E0 0E F1 0E F1
- E0 1F E0 1F F1 0E F1 0E
- 01 E0 01 E0 01 F1 01 F1
- E0 01 E0 01 F1 01 F1 01
- 1F FE 1F FE 0E FE 0E FE
- FE 1F FE 1F FE 0E FE 0E
- 01 1F 01 1F 01 0E 01 0E
- 1F 01 1F 01 0E 01 0E 01
- E0 FE E0 FE F1 FE F1 FE
- FE E0 FE E0 FE F1 FE F1

- 穷举搜索密钥。
- 64bit密钥的DES，其密钥量为$2^{56}\approx 7.2\times 10^{16}$
- 假定DES加密操作需时5μs，则需要1.1×10^{4}年才能穷尽密钥。
- Diffe和Hellman提出专用机的设想，将100万个专用芯片并行运算，以每秒10^{12}密钥的速度，搜索遍2^{56}个密钥，这只需一天即可完成，估计成本在2000万到7000万美元之间。
- 将这么多的芯片连在一起工作，其平均故障时间将是很短的，故这种方法并不现实。

- 随着计算技术和互联网的发展，64bit密钥的DES穷举搜索在今天已成为确实可行之事。
- 1997年，RSA数据安全公司悬赏1万美元破译DES，
- DESCHALL小组经过近四个月的努力，通过互联网搜索了3×10^{16}个密钥找到了密钥。
- 1998 年 5 月 ， 美 国 EFF(Electronic Frontier Foundation)宣布，用一台价值20万美元的计算机改装的专用解密机，用了56小时破译了RSA数据安全公司悬赏1万美元破译了64bit密钥的DES。
- 在目前计算能力下，64bit密钥的DES已难于抗穷举搜索攻击，必须使用新的加密算法。

七、双重DES

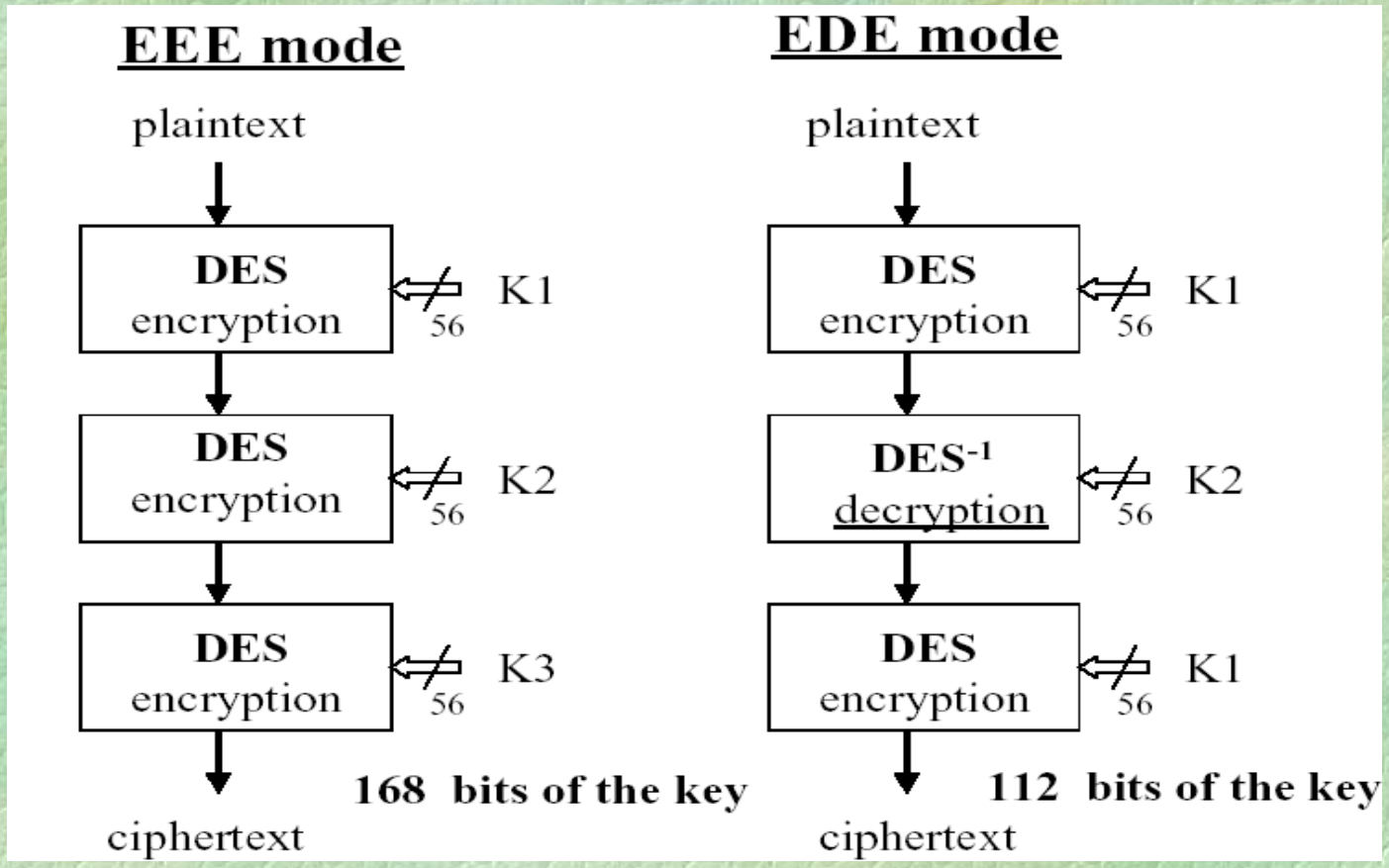
- 双重DES是用两个56位的密钥对明文进行两次DES加密
  - $C = E_{K2}[E_{K1}[P]]$
  - $P = D_{K1}[D_{k2}[C]]$
- 已经证明两重DES不能通过一个DES来进行替代，所以密钥有效长度是112位。

- 中途攻击
- 利用$X = E_{K1}[P] = D_{k2}[C]$
- 首先用2^{56}个可能的密钥加密P，并排序
- 然后用2^{56}个可能的密钥解密C，进行匹配
- 对于匹配成功的再使用新的明文密文对进行攻击。
- 两个已知明文密文对能较少误报率到2^{-16}。
- 攻击总量是2^{56}，近似于DES算法。

- 三重DES加密
- 这种方式里使用三(或两)个不同的密钥对数据块进行三次(或两次)加密。三重DES的强度大约和112-bit的密钥强度相当。三重DES有四种模型：
  - DES-EEE3 使用三个不同密钥顺序进行三次加密变换
  - DES-EDE3 使用三个不同密钥依次进行加密-解密-加密变换
  - DES-EEE2 其中密钥K1=K3 顺序进行三次加密变换
  - DES-EDE2 其中密钥K1=K3 依次进行加密-解密-加密变换

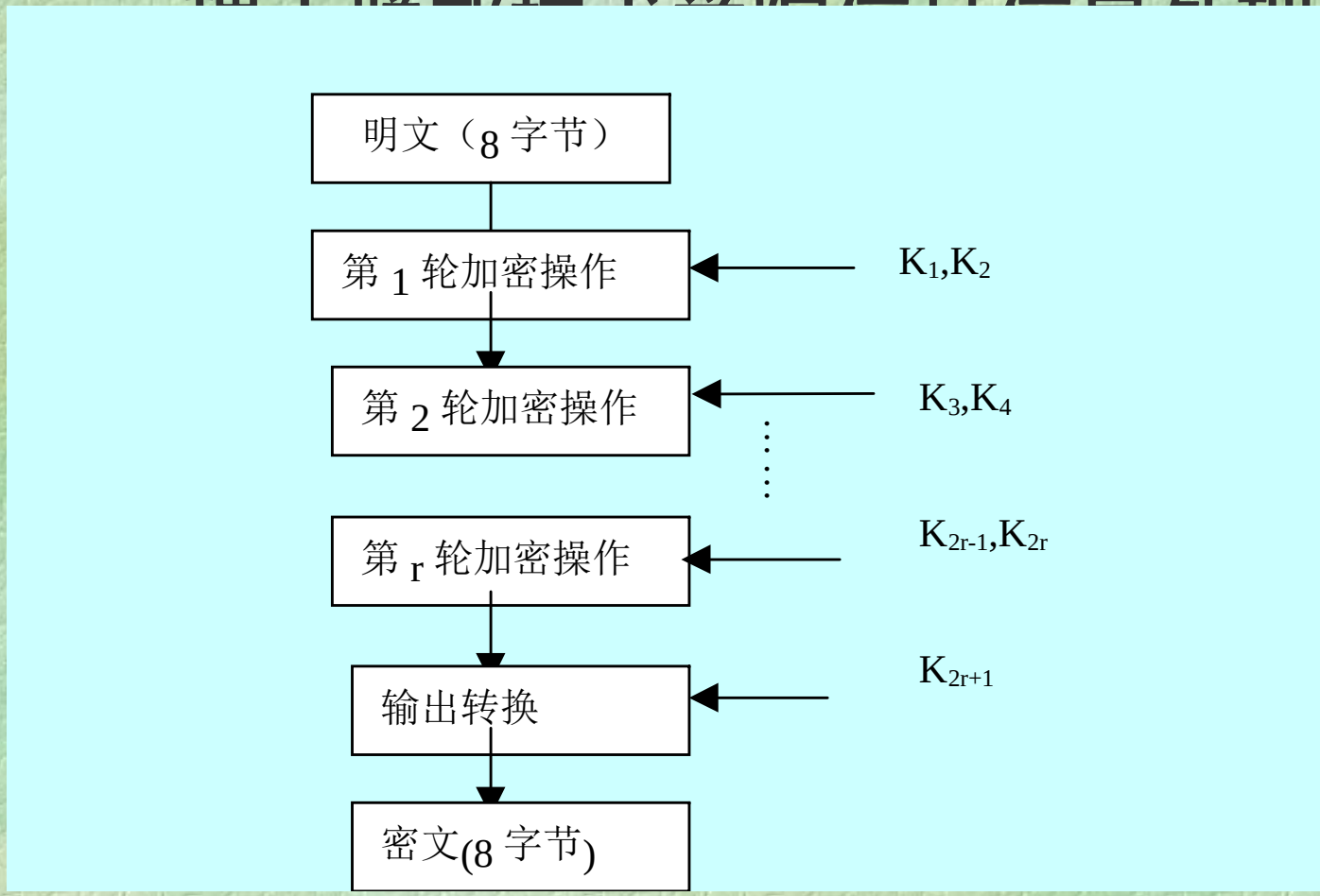
  到目前为止，还没有人给出攻击三重DES的有效方法。对其密钥空间中密钥进行蛮干搜索，那么由于空间太大，这实际上是不可行的。

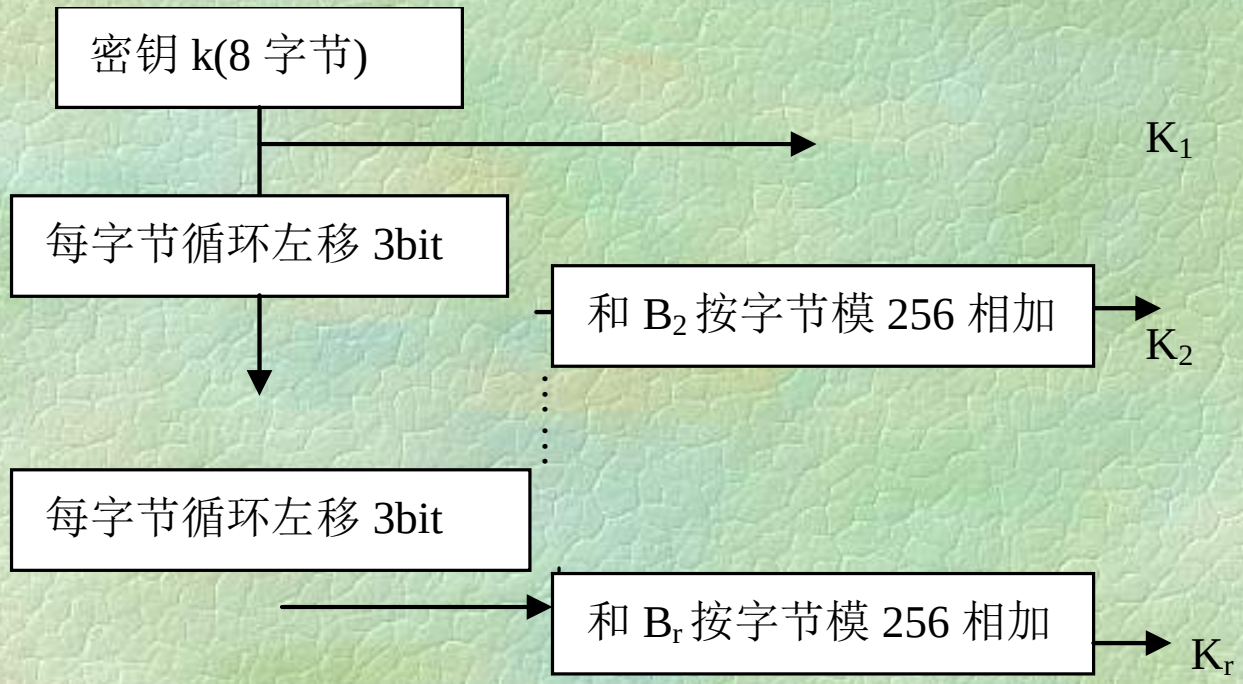
三重DES


EEE mode
plaintext
DES encryption
56
K1
DES encryption
56
K2
DES encryption
56
K3
ciphertext
168 bits of the key
EDE mode
plaintext
DES encryption
56
K1
DES-1 decryption
56
K2
DES encryption
56
K1
ciphertext
112 bits of the key

3.2.3 SAFER加密算法

瑞士联邦技术学院信息信息处理实验室
使用64比
算法：



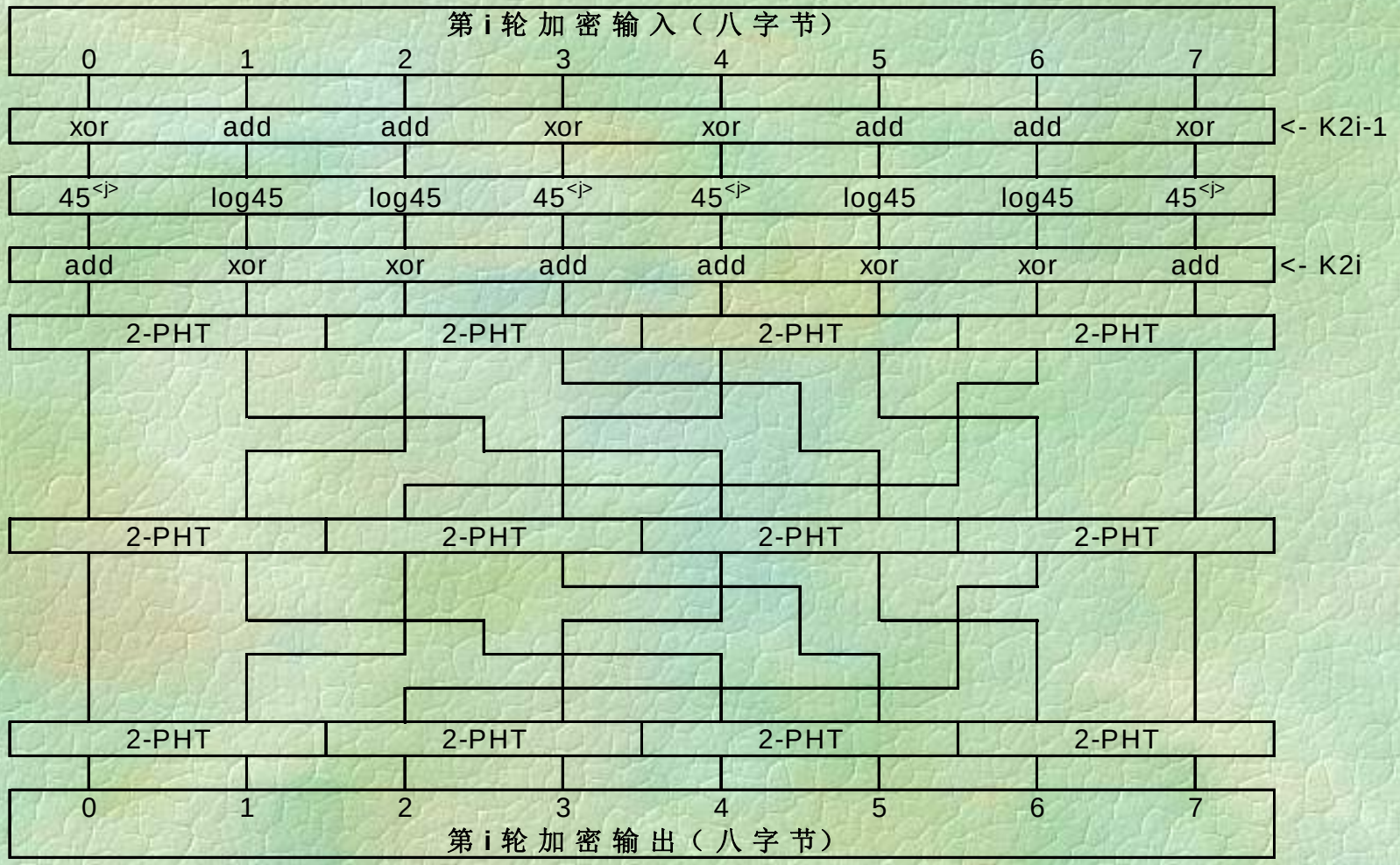


1.用户输入的密钥即赋予第一个子密钥K_1(八字节)。
2.将用户指定的密钥循环左移3比特后与B_2相加，并对256取模，其结果赋予K_2。
3.将已循环左移的密钥重复作第二步操作，直至生成全部2r+1个子密 钥。

1	45	226	147	190	69	21	174
120	3	135	164	184	56	207	63
8	103	9	148	235	38	168	107
189	24	52	27	187	191	114	247
64	53	72	156	81	47	59	85
227	192	159	216	211	243	141	177
255	167	62	220	134	119	215	166
17	251	244	186	146	145	100	131
241	51	239	218	44	181	178	43
136	209	153	203	140	132	29	20
129	151	113	202	95	163	139	87
60	130	196	82	92	28	232	160
4	180	133	74	246	19	84	182
223	12	26	142	222	224	57	252
32	155	36	78	169	152	158	171
242	96	208	108	234	250	199	217
0	212	31	110	67	188	236	83
137	254	122	93	73	201	50	194
249	154	248	109	22	219	89	150
68	233	205	230	70	66	143	10
193	204	185	101	176	210	198	172
30	65	98	41	46	14	116	80
2	90	195	37	123	138	42	91
240	6	13	71	111	112	157	126
16	206	18	39	213	76	79	214
121	48	104	54	117	125	228	237
128	106	144	55	162	94	118	170
197	127	61	175	165	229	25	97
253	77	124	183	11	238	173	75
34	245	231	115	35	33	200	5
225	102	221	179	88	105	99	86
15	161	49	149	23	7	58	40

入每一
证没有
多数分
由以下

..., 8),


第 i 轮 加 密 输 入（ 八 字 节）
0 1 2 3 4 5 6 7
xor add add xor xor add add xor
<- K2i-1
45<j> log45 log45 45<j> 45<j> log45 log45 45<j>
add xor xor add add xor xor add
<- K2i
2-PHT 2-PHT 2-PHT 2-PHT
2-PHT 2-PHT 2-PHT 2-PHT
2-PHT 2-PHT 2-PHT 2-PHT
0 1 2 3 4 5 6 7
第 i 轮 加 密 输 出（ 八 字 节）

128 , 0 , 176 , 9 , 96 , 239 , 185 , 253 ,
16 , 18 , 159 , 228 , 105 , 186 , 173 , 248 ,
192 , 56 , 194 , 101 , 79 , 6 , 148 , 252 ,
25 , 222 , 106 , 27 , 93 , 78 , 168 , 130 ,
112 , 237 , 232 , 236 , 114 , 179 , 21 , 195 ,
255 , 171 , 182 , 71 , 68 , 1 , 172 , 37 ,
201 , 250 , 142 , 65 , 26 , 33 , 203 , 211 ,
13 , 110 , 254 , 38 , 88 , 218 , 50 , 15 ,
32 , 169 , 157 , 132 , 152 , 5 , 156 , 187 ,
34 , 140 , 99 , 231 , 197 , 225 , 115 , 198 ,
175 , 36 , 91 , 135 , 102 , 39 , 247 , 87 ,
244 , 150 , 177 , 183 , 92 , 139 , 213 , 84 ,
121 , 223 , 170 , 246 , 62 , 163 , 241 , 17 ,
202 , 245 , 209 , 23 , 123 , 147 , 131 , 188 ,
189 , 82 , 30 , 235 , 174 , 204 , 214 , 53 ,
8 , 200 , 138 , 180 , 226 , 205 , 191 , 217 ,
208 , 80 , 89 , 63 , 77 , 98 , 52 , 10 ,
72 , 136 , 181 , 86 , 76 , 46 , 107 , 158 ,
210 , 61 , 60 , 3 , 19 , 251 , 151 , 81 ,
117 , 74 , 145 , 113 , 35 , 190 , 118 , 42 ,
95 , 249 , 212 , 85 , 11 , 220 , 55 , 49 ,
22 , 116 , 215 , 119 , 167 , 230 , 7 , 219 ,
164 , 47 , 70 , 243 , 97 , 69 , 103 , 227 ,
12 , 162 , 59 , 28 , 133 , 24 , 4 , 29 ,
41 , 160 , 143 , 178 , 90 , 216 , 166 , 126 ,
238 , 141 , 83 , 75 , 161 , 154 , 193 , 14 ,
122 , 73 , 165 , 44 , 129 , 196 , 199 , 54 ,
43 , 127 , 67 , 149 , 51 , 242 , 108 , 104 ,
109 , 240 , 2 , 40 , 206 , 221 , 155 , 234 ,
94 , 153 , 124 , 20 , 134 , 207 , 229 , 66 ,
184 , 64 , 120 , 45 , 58 , 233 , 100 , 31 ,
146 , 144 , 125 , 57 , 111 , 224 , 137 , 48

- SAFER的解密运算是其加密运算的逆运算，只需对“SAFER 算 法 主流程”中的算法步骤自下而上依次取逆即 可。
- SAFER算法解密算法中子密钥的生成算法仍和加密算法中的生成过程一致。
- 异或运算的逆运算为：
- 设：a XOR b = c，则a＝b XOR c
- 加法运算的逆运算为：
- 设：(a+b)mod256=c，则a＝(c-b+256) mod 256。
- 伪哈德码变换的逆运算为：
- 设：$b_1=(2*a_1+a_2)\bmod 256$, $b_2=(a_1+a_2)\bmod 256$
- 则：$a_1=(b_1-b_2+256)\bmod 256$,
- $a_2=(2b_2-b_1+256)\bmod 256$

- 三、安全性分析
- SAFER算法具有良好的扩散和混乱性能，抗攻击性强。具有明文、密钥的变化能够迅速扩散的特点。
- 六轮SAFER K－64算法能安全抵抗差分密码分析。
- 在SAFER算法的子密钥生成过程中，由于引进了B这个随机因子可以确保馈入每一轮的子密钥均是随机的，并且保证没有全零的子密钥，从而不象DES那样存在明显的弱密钥问题的，

- 作为15个高级加密算法候选之一的SAFER+，就是把原来8个字节的分组扩展为16个字节(即128bit)，每轮均使用2个16字节的子密钥进行运算，密钥K也是16字节，由它产生每轮的工作子密钥。b_{ij}的产生也将相应变化：

$$b_{ij} = 45^{(45^{(17i+j) \bmod 257} \bmod 257)} \bmod 257 \qquad (i=2,3,\cdots, 17,\ j=1, 2, \cdots, 16)$$

$$b_{ij} = 45^{(17i+j) \bmod 257} \bmod 257 \qquad (i=18,19,\cdots, 33,\ j=1, 2, \cdots, 16)$$

输出变换则作了改进，使得可以更好地抗击差分分析

3.2.4 新一代分组迭代加密算法——Rijndael算法

- 由于DES算法已不能提供足够的安全性，1997年4月15日，美国国家标准技术研究所(NIST)发起征集AES(Advanced Encryption Standard)算法的活动。
- AES算法的基本要求是比三重DES快且至少与三重DES一样安全，分组长度为128比特，密钥长度为128/192/256比特。
- 1998年8月20日，NIST召开了第一次AES候选会议，并公布了15个候选算法，研究讨论其安全性

- 5个来自美国，即HPC、MARS、RC6、SAFER+和TWOFISH；
- 2个来自加拿大，即CAST-256和DEAL；
- 其他算法分别来自澳大利亚(LOKI 97)、比利时(RIJNDAEL)、哥斯达黎加(FROG)、法国(DFC)、德国(MAGENTA)、日本(E2)、韩国(CRYPTON)以及由英国、以色列和挪威联合设计的SERPENT。

- 有的是在原有算法基础上提出的，
- 如CAST-256是对CAST-128的改进，
- DEAL则是对DEA的改进，
- LOKI 97则是对LOKI 91的改进，RC6则是对RC5的改进，
- SAFER^{+} 是对SAFER的改进。
- 在15个算法中，有些经过分析攻击后发现存在明显的缺陷，它们是HPC、DEAL、LOKI 97、FROG、DFC和MAGENTA，
- 其余则是AES的有力竞争者。

- 所有算法其基本设计方法无论在理论上还是在技术上都与DES的设计方法类似，
- 或者模仿并自己设计出我们的AES算法，
- 或者在理论上寻求突破，象当初Shannon理论那样，指导构造新一代实用算法。
- 1999年3月22日举行了第二次AES候选会议，宣布对这些算法讨论的结果，并于4月15日公布了从中优选出的5个候选算法，
- 它们是：美国的MARS(IBM)、RC6(RSA实验室)和TWOFISH(部分学者提出的)，比利时的RIJNDAEL，以及由英国、以色列和挪威学者联合设计的SERPENT。

- 2000年10月2日,美国商务部长宣布:最终胜出的是比利时的Rijndael算法,
- Rijndael算法采用的是代替——置换网络结构，每一轮由三层组成：线形混合层，非线性层，和密钥加层。
- 线形混合层的作用是确保多轮之后的高度扩散。
- 非线性层由16个S盒并置而成，起到混淆的作用。S盒选取的是有限域$GF(2^8)$中的乘法逆运算，其差分均匀性和线性偏差都达到了最佳。

- 1总体框架
- Rijndael算法分组长度和密钥长度都是可变的，它们可以独立地指定为128bit,192 bit或256bit。
- Rijndael算法中的许多运算都是按字节进行的，算法的各个中间结果采用字节的矩阵阵列图表示。
- 阵列有4行,Nb列(Nb=分组长度/32，即阵列中每个元素都是1个字节8bit)。
- 密钥同样采用1个4行的定义矩阵阵列图表示，其列数为Nk，且Nk=密钥长度/32。

- **Rijndael算法的迭代轮数用Nr表示，与Nb,Nk有关，如下表所示**

Nr: Nb / Nk	4	6	8
4	10	12	14
6	12	12	14
8	14	14	14

Rijndael算法由以下三部分组成：
密钥加，Nr-1轮变换，结尾轮变换。
先由主密钥生成工作密钥串，
然后进行首次密钥加，
随后进行Nr-1轮变换，
最后是结尾轮变换。

- ## 2.字节的表示与运算
- 字节可以看作为有限域$GF(2^8)$中的元素。
- 有限域中的元素可以用多种不同的方式表示，
- 根据代数知识可知，对于任意素数的幂次构成的有限域，在同构意义下是唯一的。
- 一种传统的表示方式是多项式表示法。
- $GF(2^8)$可看作为$GF(2)$上的8次不可约多项式的扩域，
- 因此其元素可表示为$b_7x^7+b_6x^6+b_5x^5+b_4x^4+b_3x^3+b_2x^2+b_1x+b_0$ ($b_i\in\{0,1\}$, i=0,1,…7)，
- 由$b_7b_6b_5b_4b_3b_2b_1b_0$，构成的字节b就是多项式的系数。

- 例如，十六进制数‘57’(二进制表示为01010111)这样一个字节对应的多项式是$x^6+x^4+x^2+x+1$。
- 在用多项式表示元素的有限域中，其两个元素的和仍是一个多项式。
- 由于GF(2^8)是{0,1}的扩域，故这个多项式的系数是两个元素对应系数模2加(即1⊕1=0)。
- 例：‘57’+‘83’
- ‘57’+‘83’=‘D4’。
- 加法可看作为按字节为单位的比特异或加。
- 本例中就是01010111⊕1000011 =11010100。
- 有限域其加法单位元是‘00’。

- 在多项式乘法中，GF(2^8)中的乘法实际上就是关于{0,1}上的一个8次不可约多项式乘法取摸。
- 在目前采用的Rijndael算法中，所用的多项式是$x^8+x^4+x^3+x+1$，用十六进制表示就是‘11B’。

- 在多项式乘法中，GF(2^8)中的乘法实际上就是关于{0,1}上的一个8次不可约多项式乘法取摸。
- 在目前采用的Rijndael算法中，所用的多项式是$x^8+x^4+x^3+x+1$，用十六进制表示就是‘11B’。
- 例：‘57’•‘83’
- $(x^6+x^4+x^2+x+1)\bullet(x^7+x+1) \bmod (x^8+x^4+x^3+x+1)=x^7+x^6+1$
- 即为11000001=‘C1’。有限域的乘法单位元是‘01’
- 例：求‘$(57)^{-1}$’，实际上就是求$(x^6+x^4+x^2+x+1)$关于模$x^8+x^4+x^3+x+1$的逆，逆元的计算采用欧几里德扩展算法，求得逆为$x^7+x^5+x^4+x^3+x^2+x+1$，即为10111111=‘5F’

- $x^8+x^4+x^3+x+1=(x^2+1)(x^6+x^4+x^2+x+1)+x^4$
- $x^6+x^4+x^2+x+1=(x^2+1)x^4+x^2+x+1$
- $x^4=(x^2+x)(x^2+x+1)+x$
- $x^2+x+1=(x+1)x+1$
- 故$1=(x^2+x+1)-(x+1)x$
- $=(x^2+x+1)-(x+1)(x^4-(x^2+x)(x^2+x+1))$
- $=(1+(x+1)(x^2+x))(x^2+x+1)+(x+1)x^4$
- $=(1+(x+1)(x^2+x))((x^6+x^4+x^2+x+1)-(x^2+1)x^4)+(x+1)x^4$
- $=(x^3+x+1)(x^6+x^4+x^2+x+1)+((x^3+x+1)(x^2+1)+(x+1))x^4$
- $=(x^3+x+1)(x^6+x^4+x^2+x+1)+(x^5+x^2)((x^8+x^4+x^3+x+1)-(x^2+1)(x^6+x^4+x^2+x+1))$
- $=((x^3+x+1)+(x^5+x^2)(x^2+1))(x^6+x^4+x^2+x+1)+(x^5+x^2)(x^8+x^4+x^3+x+1)$
- $=(x^7+x^5+x^4+x^3+x^2+x+1)(x^6+x^4+x^2+x+1)+(x^5+x^2)(x^8+x^4+x^3+x+1)$
- 所以$x^6+x^4+x^2+x+1$关于模$x^8+x^4+x^3+x+1$的逆元是：
- $x^7+x^5+x^4+x^3+x^2+x+1$

- 3.加密算法
- 1)首次密钥加(AddRoundkey(Roundkey))
- 这是一个比特异或运算，$b_{ij}=a_{ij}\oplus k_{ij}$($k_{ij}$为工作密钥)。
- 2)轮变换，共进行Nr轮，其中前Nr-1由4个不同的变换组成：

字节代替变换(ByteSub)、行移位变换(ShiftRow)、列混合变换(MixColumn)和密钥加(AddRoundkey(Roundkey))。

结尾轮则是由3个不同变换组成：

字节代替变换(ByteSub)、行移位变换(ShiftRow)和密钥加(AddRoundkey(Roundkey))。

- (1)字节代替变换(ByteSub)
- 首先用前面介绍的方法表示每个字节(共Nb*4个)，然后对每个字节作下述变换：
- ①在GF(2^8)中求字节的乘法逆，特别‘00’就取为自身‘00’。
- ②把字节表示成($x_0,x_1,x_2,x_3,x_4,x_5,x_6,x_7$)，然后作仿射变换：

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

- 字节代替变换(ByteSub)的逆变换记为InvByteSub，它是先进行仿射变换的逆变换，然后在GF(2^8)中求字节的乘法逆，‘00’仍取为自身‘00’。
- (2)行移位变换(ShiftRow)
- 对阵列

$$\begin{pmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,Nb-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,Nb-1} \\ a_{2,0} & a_{2,1} & \cdots & a_{2,Nb-1} \\ a_{3,0} & a_{3,1} & \cdots & a_{3,Nb-1} \end{pmatrix}$$

第0行不移位，
第1行循环移位C_1字节
第2行循环移位C_2字节
第3行循环移位C_3字节

- 位移量C_1, C_2, C_3与分组长度Nb有关：

Nb	C_1	C_2	C_3
4	1	2	3
6	1	2	3
8	1	3	4

行移位变换(ShiftRow)的逆变换记为InvShiftRow，它是后3行分别循环移位Nb-C_1字节，Nb-C_2字节，Nb-C_3字节。

- (3)列混合变换(MixColumn)
- 在列混合变换中，先将矩阵的阵列图中每个列看作为$GF(2^8)$上的3次多项式的系数(这里系数为十六进制)，
- 然后将这些多项式与多项式c(x)关于模x^4+1乘，
- 其中$c(x)=‘03’x^3+‘01’x^2+‘01’x+‘02’$。
- 由于c(x)与x^4+1互素，因此关于模x^4+1是可逆的。

- 定理：多项式 $b(x)=b_3x^3+b_2x^2+b_1x+b_0$ 在 $GF(2^8)$ 上是模 x^4+1 可逆，当且仅当下列矩阵在 $GF(2^8)$ 上可逆。

$$\begin{pmatrix} b_0 & b_3 & b_2 & b_1 \\ b_1 & b_0 & b_3 & b_2 \\ b_2 & b_1 & b_0 & b_3 \\ b_3 & b_2 & b_1 & b_0 \end{pmatrix}$$

- $\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} c_0 & c_3 & c_2 & c_1 \\ c_1 & c_0 & c_3 & c_2 \\ c_2 & c_1 & c_0 & c_3 \\ c_3 & c_2 & c_1 & c_0 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$ **+1乘法运算，**

列混合变换为：

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

- $b_2=a_2c_0 \oplus a_1c_1 \oplus a_0c_2 \oplus a_3c_3$;
- $b_3=a_3c_0 \oplus a_2c_1 \oplus a_1c_2 \oplus a_0c_3$;

- 列混合变换(MixColumn)的逆变换记为InvMixColumn，
  它是每列关于模x^4+1乘特定的多项式d(x)，d(x)满足$c(x)\otimes d(x)=$‘01’，
  因此$d(x)=$‘0B’x^3+‘0D’x^2+‘09’$x+$‘0E’。
- (4)密钥加(AddRoundkey(Roundkey))
- 此变换与首次密钥加相同。

- 3)生成工作密钥串(Keyexpansion)
- 工作密钥串由主密钥生成。
- 基本原理是：
- 工作密钥总数等于分组长度×轮数+1
- 用主密钥扩展成工作密钥串(keyexpansion)；
- 工作密钥按下述方式从工作密钥串中取出
- 第一个工作密钥由最前面的Nb个字组成，
- 第二个工作密钥由接下来的Nb个字组成，如此继续下去。

- 工作密钥串的生成方式与Nk的取值有关，分为Nk≤6与Nk＞6两类。
- (1)Nk≤6
- 把主密钥按字节表示，记为$k_0,k_1,k_2,\ldots, k_{4*Nk-1}$，并且用w(i)表示4字节的字。
- For (i=0; i<Nk; i++)
- w(i)=($k_{4i},k_{4i+1},k_{4i+2},k_{4i+3}$) 即最前面的Nk个字是由主密钥填充而成
- For (i=Nk; i<Nb*(Nr-1); i++)
- ①如果i是Nk的整数倍，先对w(i-1)进行变换，把字w(i-1)=(a,b,c,d)进行一个字节的循环移位得(b,c,d,a)
- 然后与轮常数Rcon(i/Nk)进行异或运算，得到变换后的w(i-1)。其中轮常数Rcon(i)=(Rc(i),00,00,00)。
- Rc(1)=01, Rc(i)=02·Rc(i-1)
- ②w(i)=w(i-1)⊕ (i-Nk)

- (2)Nk>6
- 把主密钥按字节表示，记为$k_0,k_1,k_2,\ldots,k_{4*Nk-1}$
- For (i=0; i<Nk; i++)
- $w(i)=(k_{4i},k_{4i+1},k_{4i+2},k_{4i+3})$
- For (i=Nk; i<Nb*(Nr-1); i++)
- ①如果i是Nk的整数倍，则同Nk≤6一样先对w(i-1)进行变换，得到变换后的w(i-1)；

  如果i=4 mod Nk，则对w(i-1)进行一个字节的循环移位，得到变换后的w(i-1)。
- ②w(i)=w(i-1)⊕ (i-Nk)

- **4.解密算法**
- **先进行加密算法中最后一轮的逆运算，然后进行Nr-1轮逆运算，最后进行密钥加运算。**
- **其中加密算法中最后一轮的逆运算是：密钥加(AddRoundkey(Roundkey))、行移位逆变换(InvShiftRow)、字节代替逆变换(InvByteSub)。**
- **Nr-1轮逆运算的每轮形式是：密钥加(AddRoundkey(Roundkey))、列混合逆变换(InvMixColumn)、行移位逆变换(InvShiftRow)和字节代替逆变换(InvByteSub)。**

- 由于行移位逆变换对字节仅进行位置移动而不影响字节的值，而字节代替逆变换则是作用在单个字节上且与它们所处位置无关，因此行移位逆变换和字节代替逆变换的先后顺序是可以交换的。
- AddRoundkey(Roundkey)和InvMixColumn可以变换为InvMixColumn和AddRoundkey(InvRoundkey)，
- 其中InvRoundkey将InvMixColumn作用到Roundkey而得到的。
- 线性变换A满足A(x+k)=A(x)+A(k)。

- 先进行密钥加运算AddRoundkey(Roundkey)，然后进行Nr-1轮逆运算，最后进行结尾轮运算。而Nr-1轮逆运算形式也改为：字节代替逆变换(InvByteSub)、行移位逆变换(InvShiftRow)、列混合逆变换(InvMixColumn)和密钥加AddRoundkey(InvRoundkey+Nb*i)(这里i为轮数)。
- 结尾轮运算改为字节代替逆变换(InvByteSub)、行移位逆变换(InvShiftRow)和密钥加AddRoundkey(Roundkey)。

3.3 分组密码的工作方式

- 现在通用的国际标准模式有ECB(电码本)模式、CBC(密码分组链接)模式、CFB(*l*-比特密码反馈)模式和OFB(输出反馈模式)，计数器模式(CTR)
- ECB模式，就是直接利用分组加密算法分别对分组后的各明文组进行加密，各密文组只与所对应的明文组有关，与其他明文组无关，明文扩散效应只对所在的组有效，对其他组无任何作用，无法实现整个明文信息的扩散效应。

- CBC模式方案是:每个明文组X[i]在加密之前先与前一组的密文Y[i-1]按位模2相加得到X‘[i]后再进行加密。
- 由于第一组明文X[1]加密时无反馈密文，通常预制一初始向量IV与X[1]进行按位模2相加，再进行加密运算，
- IV则可作为秘密参数(每次通信都可变)，用ECB模式使用同一密钥加密后传送给收方。
- 为了使接收者知道哪些数字是填充无用数字，通常用最后8位作为填充指示符，表示填充占有的字节数。

- CBC模式通过反馈使输出的密文组与以前各明文组相关，较好实现了隐蔽明文、扩散明文效应的作用，
- 存在错误传播扩散问题，如果某个密文组Y[i]在传输中出错，则X[i]和X[i+1]也会出错，即出错也有扩散。

- CFB模式就是把明文按l长分段(l<密码算法分组长n)$x_1,x_2,\ldots$，令$L=\lceil n/l \rceil$，n比特的初始向量IV可分成$I_1I_2\ldots I_L$，

然后对上述内容加密得到$\tilde{y}_1$，选最左边的 l 比特$\tilde{y}_1'$，计算$y_1=x_1\oplus\tilde{y}_1'$作为第一组密文发出；然后将 $I_1I_2\cdots I_L$ 成左移，并将 y_1 反馈上去，得到 $I_2I_3\cdots I_Ly_1$，

再对它们加密得到$\tilde{y}_2$，选最左边的 l 比特$\tilde{y}_2'$，计算 $y_2=x_2\oplus\tilde{y}_2'$ 作为第二组密文发出；再左移并把 y_2 反馈上去，

继续这样的过程，直到完成所有明文的加密为止。

IV作为秘密参数(每次通信都可变)，用ECB模式使用同一密钥加密后传送给收方

- 该模式的优点是输出的密文组与以前各明文组相关，较好实现了隐蔽明文、扩散明文效应的作用，并且能适应不同数据格式，
- 但存在错误传播扩散问题，如果l=8，一组出错，将导致连续9组出错(n=64)，并且每加密l比特，要进行1次加密运算，运算量增加，效率降低。
- 由于在数据网等低层次中，一般效率都不高，因此这种模式适用于数据网等低层次中。

- OFB模式就是把明文按l长分段(l<密码算法分组长n)$x_1,x_2,\ldots$，令$L=\lceil n/l \rceil$，n比特的初始向量IV可分成$I_1I_2\ldots I_L$，

然后对上述内容加密得到$\tilde{y}_1$，选最左边的 l 比特 $\tilde{y}_1'$，计算$y_1=x_1\oplus\tilde{y}_1'$作为第一组密文发出；

然后将 $I_1I_2\cdots I_L$ 成左移，并将 $\tilde{y}_1'$ 反馈上去，得到 $I_2I_3\cdots I_L\tilde{y}_1'$，再对它们加密得到 $\tilde{y}_2$，选最左边的 l 比特 $\tilde{y}_2'$，计算 $y_2=x_2\oplus\tilde{y}_2'$ 作为第二组密文发出；

再左移并把 $\tilde{y}_2'$ 反馈上去，继续这样的过程，

直到完成所有明文的加密为止。

- IV作为秘密参数(每次通信都可变)，用ECB模式使用同一密钥加密后传送给收方。
- 该模式的优点是没有错误传播扩散问题，
- 但引进了流密码的缺点，当密文被篡改时难以被检测，而CBC模式和CFC模式则容易检测出此种情况。

- OFB模式使用时必须保持严格的同步，否则无法解密。该模式主要在同步信道中运行，由于攻击者难以确定通信的起止而无法实施篡改攻击。
- 四种加密工作模式适用于不同的场合。ECB模式通常用来加密密钥和初始向量IV，CBC模式则可用来对字符加密，可用于认证系统，OFB模式则可用于卫星通信加密。

- 计数器模式(CTR)
- 将计数器从初始值开始计数所得到的值反馈给分组密码算法。
- 输入：CTR_1(初始非保密值)，$m_1,m_2,\ldots m_n$
- 输出： CTR_1，$c_1,c_2,\ldots c_n$
- 这里$c_i=m_i+E(CTR_i),i=1,2\ldots,n$
- 由于没有反馈，CTR模式的加解密能够同时进行，这是比CFB模式和OFB模式优越之处。

3.4 密码攻击方法

- 3.4.1典型密码攻击
- 常见的密码攻击方法有已知密文攻击、已知明文攻击、选择明文攻击、选择密文攻击等。
- 已知密文攻击，就是密码分析者拥有一个或多个用同一个密钥加密的密文，通过对它们分析得到明文或密钥。

- 已知明文攻击，就是除待破译的密文外，密码分析者拥有一些明文和用同一个密钥加密这些明文所对应的密文，
- 通过对它们分析得到密钥和破译的密文所对应的明文。
- 其代表就是穷举搜索方法，对给定的一个明密文对，试验所有可能的密钥，平均需要试验$2^{||k||-1}$次。
- 线性分析攻击也属于这一类，它通过寻找一个给定密码算法的有关明文比特、密文比特和密钥比特的有效线性近似表达式。

- 选择明文攻击，就是指密码分析者拥有所需要的任何明文所对应的密文，
- 这些密文与待破译的密文都是用同一个密钥加密所得到的。
- 查表方法就是其中之一，对给定的明文m，对所有可能的密钥K，预计算密文，然后制成表。
- 差分分析法也是选择明文攻击，它通过分析特定明文差对结果密文差的影响来获得可能性最大的密钥

- 选择密文攻击，就是指密码分析者拥有除待破译的密文外所需要的任何密文所对应的明文，解密这些密文与解密待破译的密文的密钥是相同的。

- 作业: P110 3
- 补充：请给出1个Z_2上的其他8次不可约多项式,并验证你的结论
- Project 1-1:4轮DES的编程实现
- 加密使用64位明文和64位密钥做为输入，产生64位的密文做为输出。加解密使用相同的密钥。要求:采用CBC模式对文本加解密
- 1.对话界面:选择加密或解密,输入密钥,在目录中选择明文或密文文件(内容为二进制,.txt文件,),提示加密或解密完成
- 2.输出:产生密文文本或明文文本
- 3.提供:说明文档,源码,可执行程序,通过加密实验,给出运行结果.
- 递交时间：必须在10月21日之前，网上上传
- 文件名为学号p1-1