第一部分　加解密

02.　对称密钥密码

简介

- 流密码——类似于"一次一密"
  - 相对小规模的密钥
  - 密钥被"扩展"为长的二进制比特流
  - 使用方法与一次一密相同
- 分组密码——基于电码本的思想设计的
  - 电码本是由分组密码密钥确定的
  - 每个密钥都确定一本不同的电码本
  - 同时使用了"混淆" 和"扩散"

流密码

- 曾几何时，不久前，流密码是加密的王者
- 现今，流密码不如分组密码流行
- 两种流密码算法例子：
- A5/1算法
  - 基于位移寄存器
  - 使用于GSM移动通信系统中
- RC4算法
  - 基于置换的查表方式
  - 占用大量空间

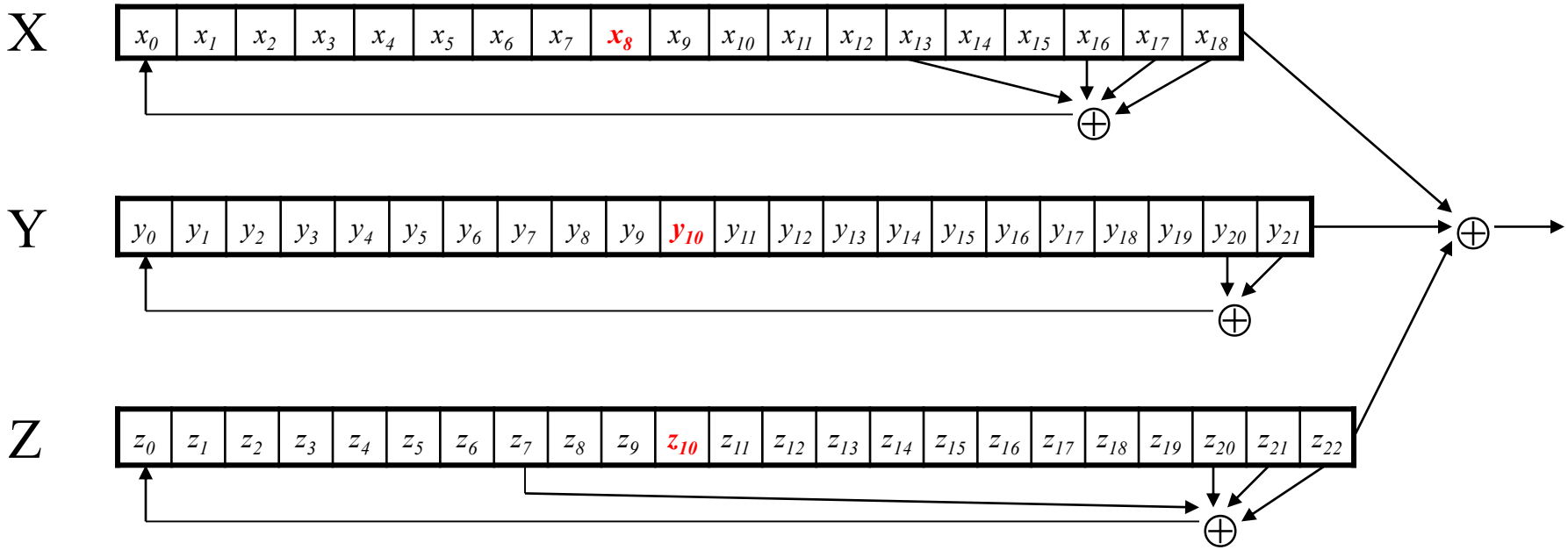
A5/1：移位寄存器

- A5/1 使用 3移位寄存器
  - X: 19 比特, 编号为$(x_0,x_1,x_2, \ldots,x_{18})$
  - Y: 22比特, 编号为$(y_0,y_1,y_2, \ldots,y_{21})$
  - Z: 23比特, 编号为$(z_0,z_1,z_2, \ldots,z_{22})$

A5/1

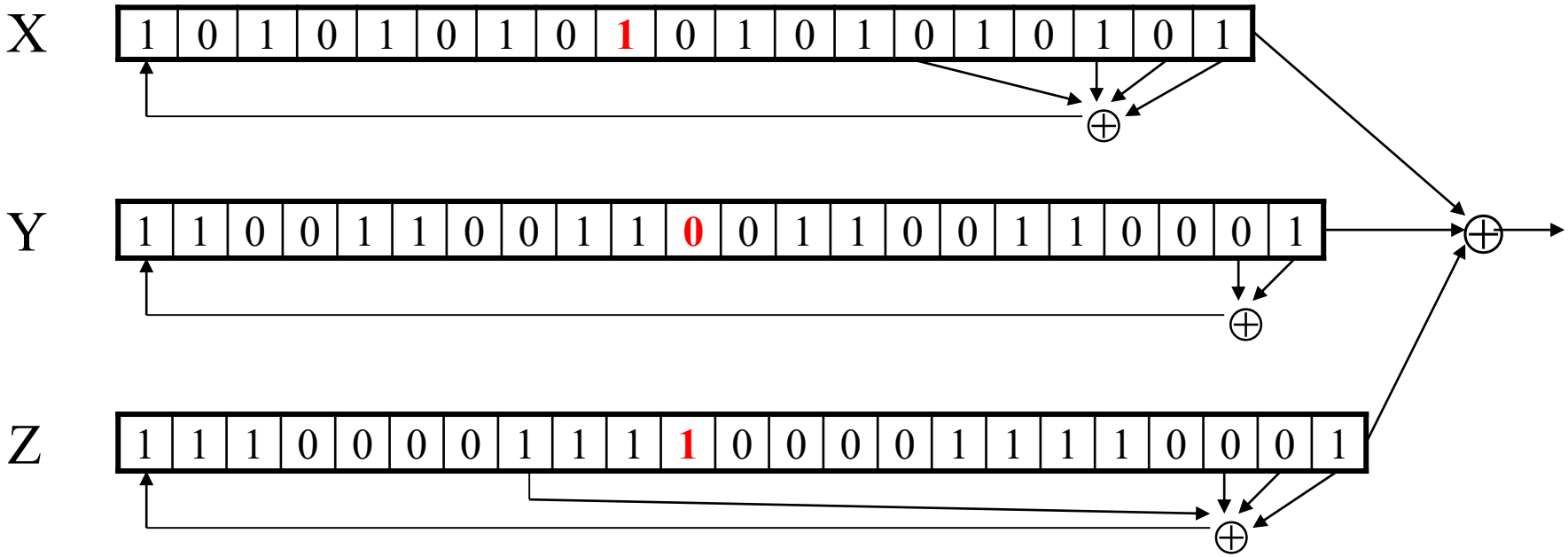
- 每一时钟周期做运算: $m = \text{maj}(x_8, y_{10}, z_{10})$
  - 如: maj(0,1,0) = 0 和 maj(1,1,0) = 1
- 如果 $x_8 = m$ 那么进行X 操作
  - $t = x_{13} \oplus x_{16} \oplus x_{17} \oplus x_{18}$
  - $x_i = x_{i-1}$， $i = 18,17,\ldots,1$和 $x_0 = t$
- 如果 $y_{10} = m$ 那么进行Y 操作
  - $t = y_{20} \oplus y_{21}$
  - $y_i = y_{i-1}$， $i = 21,20,\ldots,1$和 $y_0 = t$
- 如果 $z_{10} = m$ 那么进行Z 操作
  - $t = z_7 \oplus z_{20} \oplus z_{21} \oplus z_{22}$
  - $z_i = z_{i-1}$， $i = 22,21,\ldots,1$和 $z_0 = t$
- 密钥流比特按照关系 $S=x_{18} \oplus y_{21} \oplus z_{22}$ 产生

A5/1



- 1位表示一个值
- 密钥用于初始化寄存器，填充三个寄存器。
- 每个寄存器是否操作取决于 (x_8, y_{10}, z_{10})
- 密钥流比特与寄存器右边做 XOR 运算

A5/1



- 如此例: $m = \text{maj}(x_8, y_{10}, z_{10}) = \text{maj}(1,0,1) = 1$
- 寄存器 X操作, Y不操作,Z操作
- 密钥流比特与寄存器右边做 XOR 运算
- 这里，密钥流比特是：$0 \oplus 1 \oplus 0 = 1$

位移寄存器流密码

- 基于位移寄存器的流密码硬件实现简单，能高速产生密钥流
- 软件实现较困难
- 曾经是主流的密码之一
- 由于处理器性能的提高，当前很多很多流密码采用了软件实现手段（RC4）
- 现今，位移寄存器流密码仍在使用
  - 资源受限的设备

练习

1.这个问题针对的是A5/1加密方案。对于如下每个问题,请回答并给出你的理由。

a.平均而言,寄存器X步骤的执行频率如何?

3/4

b.平均而言,寄存器Y步骤的执行频率如何?

3/4

c.平均而言,寄存器Z步骤的执行频率如何?

3/4

d.平均而言,所有三个寄存器步骤都执行的频率如何?

1/4

e.平均而言,只有两个寄存器步骤执行的频率如何?

3/4

f.平均而言,只有一个寄存器步骤执行的频率如何?

0

g.平均而言,没有寄存器步骤执行的频率如何?

0

RC4

- 自修改的查表方式
- 表始终都包含一个{0,1,…,255}的置换
- 算法的第一阶段是初始化查表使用的密钥
- RC4算法的每一步：
  - 从当前查表中交换元素
  - 从表中选择一个密钥流字节
- RC4每步产生一个密钥字节
  - RC4算法专门为软件实现优化
- A5/1 每步只产生一个位
  - A5/1算法根据硬件实现的

RC4 初始化

- S[i]包含 {0,1,…,255}的置换， 每个S[i]是一个字节
- key[i] 表示密钥，每个key[i]是一个字节

```
for   i = 0 to 255
        S[i] = i   （初始化S，密钥流生成种子1）
        K[i] = key[i (mod N)]   （初始化K，密钥流生成种子2）
next   i
j = 0
for   i = 0 to 255（i保证每个字节都得到处理）
        j = (j + S[i] + K[i]) mod 256
        swap(S[i], S[j])   （j使得S带有随机性）
next   i
```

RC4 流密码

对于每一密钥流字节，都要交换表元素和选择字节：　　i = j = 0

```
while(true)
        i = (i + 1) mod 256
        j = (j + S[i]) mod 256
        swap(S[i], S[j])
        t = (S[i] + S[j]) mod 256
        keystreamByte = S[t]
```

- 密钥流字节的使用和一次一密相似
- 注意: 使用时，生成的前256字节密钥流必须丢弃
  - 否则，黑客将可能恢复密钥!

关于流密码的小结

- 流密码曾经是主流
  - 硬件实现简单、高效
  - 流密码保持与比特流(如语言)同步，要高速尝试密钥流.
  - 现在也有不少基于软件的密码的应用

分组密码

迭代分组密码

- 迭代分组密码将明文分为固定长度的分组，并产生固定长度的密文分组
- 明文通过若干**轮函数(round function)**的迭代 操作来产生密文
- 轮函数的输入是上一轮的输出及密钥
- 通常采用软件实现

Feistel密码

- Feistel密码是参考一类分组密码设计的，而不是一个特殊的密码
- 明文P被分成左右两部分，即$P= (L_0,R_0)$
- 对于每一轮 i=1,2,...,n,计算：

$$L_i= R_{i-1}$$

$$R_i= L_{i-1} \oplus F(R_{i-1},K_i)$$

这里F是轮函数，K_i第i轮的子密钥

- 最后,密文$C = (L_n,R_n)$

Feistel 密码

- 解密: 密文 C= (L_n,R_n)
- 对于每一轮 i=n,n−1,…,1, 计算：

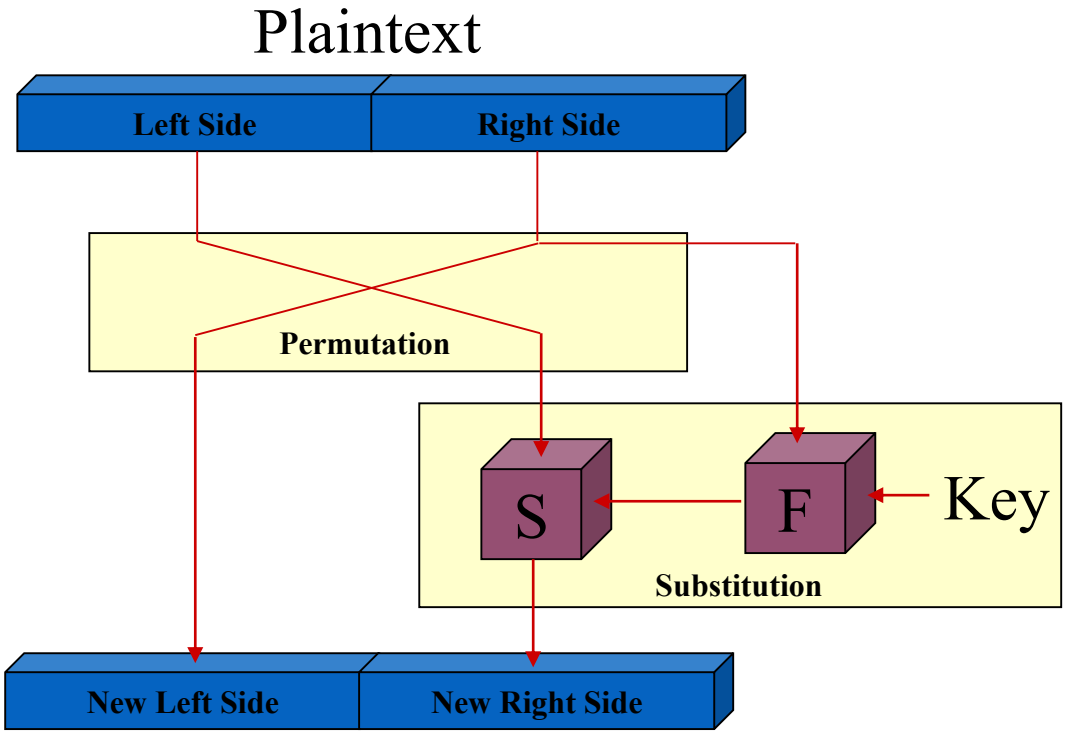
$$R_{i-1} = L_i$$

$$L_{i-1} = R_i \oplus F(R_{i-1},K_i)$$

这里F是轮函数，K_i第i轮的子密钥

- 明文 P= (L_0,R_0)
- 假设函数F输出合适的位数，那么都可以在Feistel密码结构中用作轮函数
  - 但只有特定的轮函数F才是安全的。

Feistel 密码



加密：

$L_i = R_{i-1}$

$R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$

解密：

$R_{i-1} = L_i$

$L_{i-1} = R_i \oplus F(R_{i-1}, K_i)$

练习

2.请考虑含4轮运算的Feistel密码方案。据此考虑，若明文表示为P=(L_0, R_0)，对应的密文表示为C=(L_4, R_4)。依据L_0、R_0以及子密钥，对于如下每一个轮函数，请问密文C分别是什么？

（提示：Feistel密码方案为$L_i=R_{i-1}$　　$R_i=L_{i-1}\oplus F(R_{i-1},K_i)$）

- a. $F(R_{i-1}, K)=0$

C=P

- b. $F(R_{i-1}, K_i)=R_{i-1}$

C=(R_0, $L_0 \oplus R_0$)

- c. $F(R_{i-1}, K_i)=K_i$

C=($L_0 \oplus K_1 \oplus K_3$, $R_0 \oplus K_2 \oplus K_4$)

- d. $F(R_{i-1}, K_i)= R_{i-1}\oplus K_i$

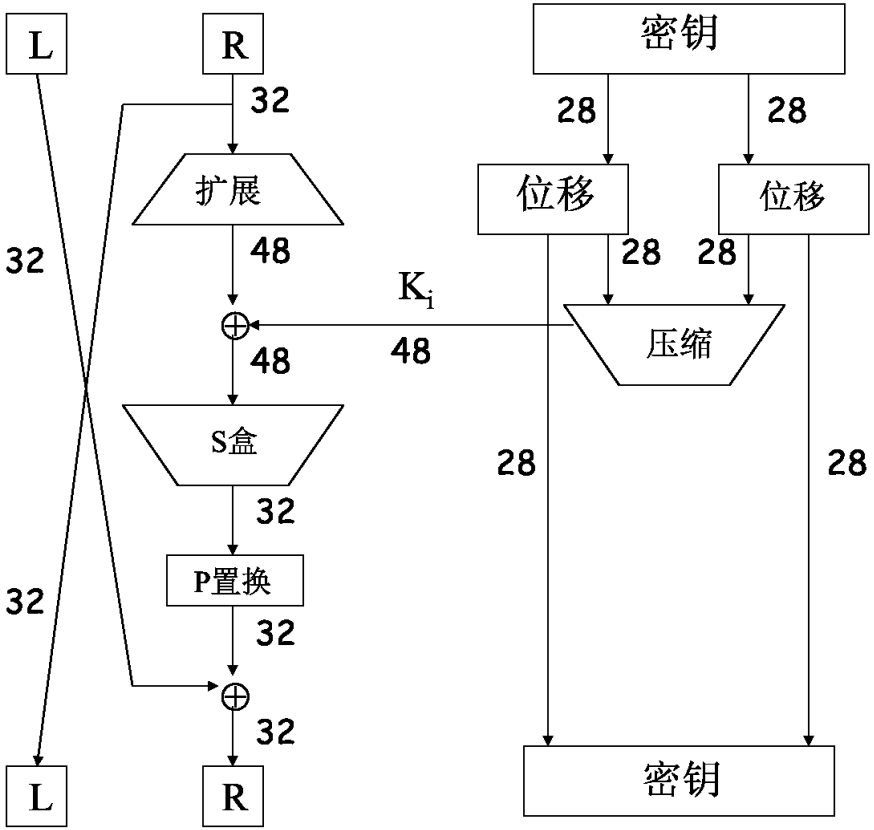
C=($R_0 \oplus K_2 \oplus K_3$, $L_0 \oplus R_0 \oplus K_1 \oplus K_3 \oplus K_4$)

数据加密标准

- DES 于1970’s提出
- 基于IBM 的Lucifer密码
- 作为美国政府的标准
- DES 设计与开发遭遇到一些问题：
  - 国家安全局(NSA)被卷入其中
  - 设计过程是秘密的
  - 密钥长度从128位减少到64位
  - 对Lucifer算法的细节改变(包括S盒的改变)

DES 特点

- DES 是 Feistel结构密码
  - DES的分组长度是64位
  - DES使用56位的密钥
  - DES 是16 轮的Feistel结构密码
  - DES 的每一轮使用48位的子密钥
- DES的每轮都很简单——至少从分组密码的设计标准来看
- "S盒"是其具有的最重要的安全特性之一
  - 每一个S盒将6位输入映射为4位输出


L
R
32
扩展
48
32
K_i
48
48
S盒
32
P置换
32
32
32
L
R
密钥
28
28
位移
位移
28
28
压缩
28
28
密钥

←DES的一轮

DES 扩展置换

- 32位输入：

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

- 48位输出：

31	0	1	2	3	4	3	4	5	6	7	8
7	8	9	10	11	12	11	12	13	14	15	16
15	16	17	18	19	20	19	20	21	22	23	24
23	24	25	26	27	28	27	28	29	30	31	0

DES的"S盒"

- DES的8 个"S盒"
- 每一个S盒都是将6位输入映射为4位输出
- 第一个S盒

输入比特 (b0,b5)

输入比特（b1,b2,b3,b4）

	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
00	1110	0100	1101	0001	0010	1111	1011	1000	0011	1010	0110	1100	0101	1001	0000	0111
01	0000	1111	0111	0100	1110	0010	1101	0001	1010	0110	1100	1011	1001	0101	0011	1000
10	0100	0001	1110	1000	1101	0110	0010	1011	1111	1100	1001	0111	0011	1010	0101	0000
11	1111	1100	1000	0010	0100	1001	0001	0111	0101	1011	0011	1110	1010	0000	0110	1101

DES的P置换

- 输入32位

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

- 输出32位

15	6	19	20	28	11	27	16	0	14	22	25	4	17	30	9
1	7	23	13	31	26	2	8	18	12	29	5	21	10	3	24

DES的子密钥

- DES的56位密钥从左到右以0,1,2,…,55编号
- DES密钥的左部分: LK

```
49 42 35 28 21 14  7
 0 50 43 36 29 22 15
 8  1 51 44 37 30 23
16  9  2 52 45 38 31
```

- DES密钥的右部分:RK

```
55 48 41 34 27 20 13
 6 54 47 40 33 26 19
12  5 53 46 39 32 25
18 11  4 24 17 10  3
```

DES密钥扩展算法

For each round i=1,2,...,16

- Let $LK = (LK$ circular shift left by $r_i)$
- Let $RK = (RK$ circular shift left by $r_i)$
- Left half of subkey K_i is of LK bits

```
13 16 10 23  0  4  2 27 14  5 20  9
22 18 11  3 25  7 15  6 26 19 12  1
```

- Right half of subkey K_i is RK bits

```
12 23  2  8 18 26  1 11 22 16  4 19
15 20 10 27  5 24 17 13 21  7  0  3
```

DES子密钥

- 定义：if i∈{1,2,9,16},ri = 1,else ri = 2
- 每轮忽略LK的8,17,21,24位
- 每轮忽略RK的6,9,14,25位
- 压缩置换 从LK和RK共56位生成48位子密钥 K_i
- 密钥调度生成子密钥

DES 的两个特性

- 第一轮之前对明文进行初始置换
- 最后一轮之后要进行逆初始置换
- 最后对(R_{16},L_{16})进行置换，从而生成密文
- 这两个特性的设计都不是出于安全性的目的

DES的安全性

- DES的安全性取决于S盒
  - S盒是DES中唯一的非线性部件
- 三十多年的详尽分析证明其没有"后门"
- 今天，攻击者的攻击方式采用穷举密钥搜索
- 原因：
  - DES的设计者预见性(预见攻击者的攻击方式)
  - DES的设计者前瞻性

分组密码表示符号

- P = 明文
- C = 密文
- 使用密钥K对明文P加密得到密文C
  - C = E(P, K)
- 使用密钥K对密文C解密得到明文
  - P = D(C, K)
- 注意：
- P = D(E(P, K), K)， C = E(D(C, K), K)

三重DES

- 现今，56位DES密钥已经太短
  - 穷举秘钥搜索是可行的
- 但是DES加密却已广泛应用，怎么办？
- 三重DES (采用112位密钥)
  - $C = E(D(E(P,K_1),K_2),K_1)$
  - $P = D(E(D(C,K_1),K_2),K_1)$
- 为什么在加密-解密-加密(EDE) 使用两个密钥?
  - 为了与DES相兼容: $E(D(E(P,K),K),K) = E(P,K)$
  - 112位已经最够安全

3DES

- 为何不是$C = E(E(P,K),K)$？
  - 其仍然只有56位密钥
- 为何不是$C = E(E(P,K_1),K_2)$？
- 选择明文攻击
  - 对所有可能的密钥值K_1预计算一个大小为2^{56}的表，表内容包括$E(P, K_1)$和K_1
  - 使用密钥K_2对C进行解密，重复尝试不同的直到找到存在的$D(C,K_2)$
  - 一旦匹配，则$E(P,K_1) = D(C,K_2)$ 工作量为2^{55}
  - 因此: $C = E(E(P,K_1),K_2)$

高级加密标准(AES)

- DES的替代方案
- AES 的竞争性 (90’s早期)
  - NSA公开参与
  - 透明过程
  - 很多高质量的提案
  - Rijndael算法最终被选中
    - 发音近似于"Rain Doll"或"Rhine Doll"
- 迭代分组密码(类似于DES)
- 非Feistel结构(与DES不同)

AES概述

- 分组大小：128, 192或256位
- 密钥长度：128, 192或256位(与分组长度的选择无关)
- 轮数从10~14不等，由密钥长度决定
- 每一轮包含四个函数，划分为三个层次
  - ByteSub (非线性层)
  - ShiftRow (线性混淆层)
  - MixColumn (非线性层)
  - AddRoundKey (密钥加层)

AES ByteSub

- 假设分组长度为128位，4x4的字节矩阵

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \longrightarrow \texttt{ByteSub} \longrightarrow \begin{bmatrix} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{10} & b_{11} & b_{12} & b_{13} \\ b_{20} & b_{21} & b_{22} & b_{23} \\ b_{30} & b_{31} & b_{32} & b_{33} \end{bmatrix}.$$

- ByteSub是AES"S盒"
- ByteSub可以视为一个非线性但可逆的两个数学函数复合

AES "S盒"

输入后4位

输入前4位	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

AES ShiftRow操作

- 每一行进行简单的按字节循环位移

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \longrightarrow \texttt{ShiftRow} \longrightarrow \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{11} & a_{12} & a_{13} & a_{10} \\ a_{22} & a_{23} & a_{20} & a_{21} \\ a_{33} & a_{30} & a_{31} & a_{32} \end{bmatrix}$$

AES MixColumn操作

- MixColumn操作是对当前矩阵的每一列进行运算，整个操作是非线性且不可逆的

$$\begin{bmatrix} a_{0i} \\ a_{1i} \\ a_{2i} \\ a_{3i} \end{bmatrix} \longrightarrow \texttt{MixColumn} \longrightarrow \begin{bmatrix} b_{0i} \\ b_{1i} \\ b_{2i} \\ b_{3i} \end{bmatrix} \quad \text{for } i = 0, 1, 2, 3$$

- 查表是最有效的实现方法

AES AddRoundKey操作

- 将子密钥和当前字节矩阵进行XOR运算：

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \oplus \begin{bmatrix} k_{00} & k_{01} & k_{02} & k_{03} \\ k_{10} & k_{11} & k_{12} & k_{13} \\ k_{20} & k_{21} & k_{22} & k_{23} \\ k_{30} & k_{31} & k_{32} & k_{33} \end{bmatrix} = \begin{bmatrix} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{10} & b_{11} & b_{12} & b_{13} \\ b_{20} & b_{21} & b_{22} & b_{23} \\ b_{30} & b_{31} & b_{32} & b_{33} \end{bmatrix}$$

分组　　子密钥

- 密钥扩展算法产生每一轮的子密钥

AES解密

- AES解密操作是可逆的
- 因⊕的逆操作是其本身，所以MixAddRoundKey的逆运算较简单
- MixColumn是可逆的，其逆反操作也是通过查表方式实现的
- Inverse of ShiftRow的逆操作简单，只要向相反方向循环位移既可
- ByteSub操作也是可逆的，其逆反操作也是通过查表方式实现的

其他分组密码

- 三种分组密码算法...
  - 数据加密算法(IDEA)
  - Blowfish
  - RC6
- 详细介绍...
  - 微型加密算法(TEA)

IDEA算法

- James Massey提出
  - 现在已被普遍使用的算法之一
- IDEA采用64位长度的分组，128位的密钥
- IDEA使用了混合模式算术(mixed-mode arithmetic)
- 组合了不同的数学操作
  - XOR、模2^{16}加法、Lai-Massey乘法
  - IDEA采用这种方法
  - 现在这种方法已经被普遍使用

Blowfish算法

- 支持分组长度为64位的加密
- 密钥采用可变长度，增长到448位
- Bruce Schneier设计
- 非Feistel结构密码
  - $R_i = L_{i-1} \oplus K_i$
  - $L_i = R_{i-1} \oplus F(L_{i-1} \oplus K_i)$
- 轮函数F采用4个S盒
  - 每一S盒将8位映射到32位
- 通过密钥产生S盒
  - 由密钥确定的S盒

RC6算法

- Ron Rivest设计的
- 算法的参数：
  - 分组大小
  - 密钥大小
  - 轮数
- RC6是AES的候选算法之一
- 基于数据的循环
  - 由数据确定密码算法的操作

TEA

- Tiny Encryption Algorithm (TEA)
- 64位的分组长度和128位的密钥
- 采用32位字的计算平台
- 轮数是可变的 (一般认为32轮才够安全)
- 使用非常简单的轮函数，但轮数必须足够大

TEA加密过程

假设使用的轮数是32:

(K[0],K[1],K[2],K[3]) = 128 bit key

(L,R) = plaintext (64-bit block)

delta = 0x9e3779b9

sum = 0

for i = 1 to 32

    sum += delta

    L += ((R<<4)+K[0])^(R+sum)^((R>>5)+K[1])

    R += ((L<<4)+K[2])^(L+sum)^((L>>5)+K[3])

next i

ciphertext = (L,R)

TEA解密过程

假设轮数是32:

(K[0],K[1],K[2],K[3]) = 128 bit key

(L,R) = ciphertext (64-bit block)

delta = 0x9e3779b9

sum = delta << 5

for i = 1 to 32

R –= ((L<<4)+K[2])^(L+sum)^((L>>5)+K[3])

L –= ((R<<4)+K[0])^(R+sum)^((R>>5)+K[1])

sum –= delta

next i

plaintext = (L,R)

TEA的分析

- 非Feistel结构密码
  - 使用+和-替代⊕ (XOR)操作
- 有简单且易于实现，内存要求较低的特性.
- "相关密钥"攻击
- TEA的扩展版本 (XTEA)，更为复杂，可以完全抵抗"相关密钥"攻击
- TEA的简化版本 (STEA) — 其强度很弱，常用于演示各种攻击方法

浙江工业大学
ZHE JIANG UNIVERSITY OF TECHNOLOGY

分组密码工作模式

多块明文分组

- 如何加密多块明文分组?
- 每一分组一个新的密钥吗?
  - 将比一次一密糟糕(甚至更糟糕)!
- 对每一分组单独加密?
- 后续分组的加密依靠前面的分组来进行加密?
  - 即将这些分组"链接"在一起?

工作模式

- 讨论是那种工作模式
- 电本码模式(ECB)
  - 对每一分组单独加密
  - 存在一些安全性问题
- 密文链接模式 (CBC)
  - 将分组密文链接在一起
  - 比ECB安全，但没有太多改进
- 计数器模式(CTR)
  - 使用方法同流密码一样
  - 通常在随机存储时使用

ECB模式

- 标记C=E(P,K)
- 假设多块明文分组为：$P_0,P_1,\ldots,P_m,\ldots$
- 在ECB模式中，解密和加密过程如下所示：
- 加密　　解密
- $C_0 = E(P_0, K)$,　　$P_0 = D(C_0, K)$,
- $C_1 = E(P_1, K)$,　　$P_1 = D(C_1, K)$,
- $C_2 = E(P_2, K)$,…　　$P_2 = D(C_2, K)$,…
- 对于固定的密钥K，分组密码就是一个电码本（没有附加本）
  - 每个密钥都确定一部不同的电码本！

ECB 剪切攻击

- 假设明文是：

  Alice digs Bob. Trudy digs Tom.

- 假设分组长度是64位和八位ASCII:

  P_0="Alice di", P_1="gs Bob. ",

  P_2="Trudy di", P_3="gs Tom. "

- 密文是: C_0,C_1,C_2,C_3
- Trudy"剪切-复制"攻击，得到: C_0,C_3,C_2,C_1
- 解密后得到：

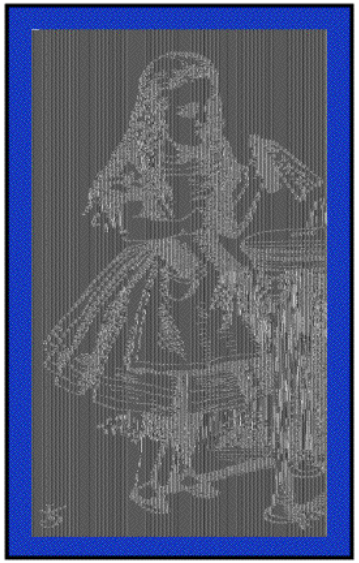
  Alice digs Tom. Trudy digs Bob.

ECB的缺点

- 假设：$P_i = P_j$
- 那么：$C_i = C_j$，且Trudy可推得：$P_i = P_j$
- 即使Trudy不知晓P_i或P_j的情况下，一些信息也会泄露出去
- Trudy可得知两个明文分组相同
- 这是安全问题吗?

Alice痛恨ECB模式

- 左边是Alice的图像，右边图像是使用ECB模式加密后的图像



- 为何产生这样的结果?
- 一样明文分组 ⇒一样的密文!

CBC模式

- 分组时"链接"在一起的
- 随机初始化一向量IV,并用它来对CBC模式进行初始化
- IV是随机产生的，且不需要保密

加密

$C_0 = E(IV \oplus P_0, K)$,

$C_1 = E(C_0 \oplus P_1, K)$,

$C_2 = E(C_1 \oplus P_2, K)$,…

解密

$P_0 = IV \oplus D(C_0, K)$,

$P_1 = C_0 \oplus D(C_1, K)$,

$P_2 = C_1 \oplus D(C_2, K)$,…

- 类似于经典的附加码本

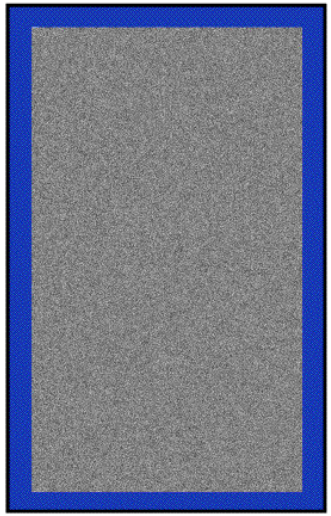
CBC模式

- 对指定的明文分组将产生不同的密文分组----这是很好的！
- 如果C_1篡改成G，那么：

  $P_1 \neq C_0 \oplus D(G, K)$, $P_2 \neq G \oplus D(C_2, K)$
- 但 $P_3 = C_2 \oplus D(C_3, K)$, $P_4 = C_3 \oplus D(C_4, K)$,…
- 可以从错误中自动恢复过来!
- 仍然会被“剪切-复制”攻击，只不过略微复杂些（并且可能会导致混淆）

Alice欣赏CBC模式

- 左边是Alice的图像，右边图像是使用CBC模式加密后的图像



- 为何产生这样的结果?
- 同样的明文产生不同的密文

CTR模式

- CTR模式通常在随机存取时使用
- 使用分组密码同流密码一样

加密	解密
$C_0 = P_0 \oplus E(IV, K)$,	$P_0 = C_0 \oplus E(IV, K)$,
$C_1 = P_1 \oplus E(IV+1, K)$,	$P_1 = C_1 \oplus E(IV+1, K)$,
$C_2 = P_2 \oplus E(IV+2, K)$,…	$P_2 = C_2 \oplus E(IV+2, K)$,…

- CBC也可以方便地实现随机存取！
  - 有一个明显的限制

浙江工业大学
ZHE JIANG UNIVERSITY OF TECHNOLOGY

完整性

数据完整性

- 完整性 ----阻止未授权的修改
  - 例如数据的修改
- 例如: 电子转账
  - 机密性很好,但完整性也很重要
- 加密将有效地保护交易的机密性 (阻止非授权信息的公开)
- 但只进行加密不能保证完整性
  - 一次一密乱数本，ECB复制黏贴等

- 消息认证码(MAC)
  - 用于确保数据的**完整性**
  - 完整性和机密性是完全不同的两个概念
- MAC与使用***CBC*模式**加密数据类似
  - 与*CBC*加密类似，但只保留最后一块密文分组作为*MAC*

MAC的计算方法

- MAC计算过程 (假设有N块分组)

$C_0 = E(IV \oplus P_0, K)$,

$C_1 = E(C_0 \oplus P_1, K)$,

$C_2 = E(C_1 \oplus P_2, K)$,…

$C_{N-1} = E(C_{N-2} \oplus P_{N-1}, K) = MAC$

- 将MAC和明文发送给接收者
- 接收者重复该计算过程并比较新计算出的"MAC"与接收到的MAC
- 接收者必须知道密钥K

MAC的工作原理

- 假设Alice有4个明文块
- Alice将计算

  $C_0 = E(IV \oplus P_0, K)$, $C_1 = E(C_0 \oplus P_1, K)$,

  $C_2 = E(C_1 \oplus P_2, K)$, $C_3 = E(C_2 \oplus P_3, K) = MAC$
- Alice将IV, P_0, P_1, P_2, P_3 给Bob，并附带上MAC
- 如果Trudy将P_1篡改成X
- Bob进行MAC验证时将计算

  $C_0 = E(IV \oplus P_0, K)$, $\mathbf{C_1} = E(C_0 \oplus X, K)$,

  $\mathbf{C_2} = E(\mathbf{C_1} \oplus P_2, K)$, $\mathbf{C_3} = E(\mathbf{C_2} \oplus P_3, K) = \mathbf{MAC} \neq MAC$
- 对于明文分组的任何改动都将通过随后的分组扩散到计算出的**MAC** 上(与CBC解密不同)
- 在没有密钥K时，Trudy不能将**MAC**篡改成MAC

秘密性与完整性

- 采用一密钥进行加密,而计算MAC时使用另一密钥
- 为何不使用相同密钥?
  - 对最后一个密文分组重复发送了两次?
  - 并没有实现安全性的增强!
- 尽管密钥是相关的，但在加密和计算MAC中仍要使用不同的密钥
  - 但仍是单独加密的工作量的两倍
  - 效果更好----大约1.5倍的“加密”
- 采用一次加密来确保机密性和完整性是研究热点

对称加密的用途

- 秘密性
  - 不安全通道中怎样进行数据传输
  - 不安全媒介上的数据怎样进行安全存储
- 完整性 (MAC)
- 认证协议
- Hash函数处理的MAC方法

练习

3.假设我们给出使用168位密钥的三重DES（3DES）的定义如下：

C=E(E(E(P,K1),K2),K3)

假设我们能够计算并存储为2^{56}的表，而且还可以选择使用明文攻击。请证明，这个三重DES（3DES）并不比常规的3DES更安全，常规的三重DES仅仅使用112位的密钥。

选择明文P并得到对应密文C，对所有的K1维护一个大小为256的表，表中包括E(P,K1)和K1，使用K2，K3对C解密，直至找到E(P,K1)=D(D(C,K3),K2)，工作量的期望为2^{111}。

关注我，下节内容更精彩：
03：公钥密码