



第3章：构建私钥密码原语

Practical Constructions of Symmetric-Key Primitives

赵俊舟

西安交通大学网安学院

junzhou.zhao@xjtu.edu.cn

2025年12月20日

目录

- 1 伪随机生成器与流密码
- 2 伪随机置换与分组密码
- 3 多重加密
- 4 工作模式

目录

1 伪随机生成器与流密码

- 流密码的基本概念
- 线性反馈移位寄存器及非线性化
- 流密码的实现

2 伪随机置换与分组密码

3 多重加密

4 工作模式

目录

1 伪随机生成器与流密码

- 流密码的基本概念
- 线性反馈移位寄存器及非线性化
- 流密码的实现

2 伪随机置换与分组密码

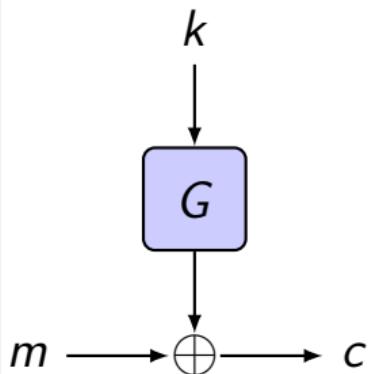
3 多重加密

4 工作模式

回顾：基于伪随机生成器构建窃听安全私钥密码

设计 (满足窃听安全的私钥密码)

- G 是一个扩展因子为 $\ell(n)$ 的伪随机生成器，定义密码 $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ 为：
- **密钥生成函数 Gen**：输入 1^n ，输出密钥 $k \leftarrow \{0, 1\}^n$
- **加密函数**： $\text{Enc}_k(m) \triangleq G(k) \oplus m$
- **解密函数**： $\text{Dec}_k(c) \triangleq G(k) \oplus c$



定理

使用伪随机生成器构建的私钥密码满足窃听安全。

问题

？那么在实际中究竟如何构建高效的伪随机生成器？

回顾：伪随机生成器

定义 (伪随机生成器, Pseudorandom Generator)

如果一个确定性多项式算法 $G: \{0, 1\}^n \mapsto \{0, 1\}^{\ell(n)}$ 满足：

- **扩展性**： $\forall n, \ell(n) > n$
- **伪随机性**：对于任意 PPT 区分器 D ，随机串 $r \leftarrow \{0, 1\}^{\ell(n)}$ ，随机种子 $s \leftarrow \{0, 1\}^n$ ，有

$$|\Pr[D(r) = 1] - \Pr[D(G(s)) = 1]| \leq \text{negl}(n)$$

则称 G 为**伪随机生成器**，称 $\ell(n)$ 为**扩展因子**。

问题

- 如果消息长度 $\ell' > \ell(n)$ ，怎么办？
- 如果消息长度 $\ell' < \ell(n)$ ，怎么办？截断？浪费！

流密码

- 在实际中，使用流密码来高效构建伪随机生成器。
- 流密码每次以比特或字节为单位输出一个比特串，通过重复调用，让流密码产生足够长的随机串。

定义 (流密码, Stream Ciphers)

一个流密码由两个确定性算法 (Init , Next) 组成，其中

- Init : 输入一个种子 s 和一个初始向量 IV (可选)，输出一个初始状态 st_0 ；
- Next : 输入当前状态 st ，输出一个比特 y 和下一个状态 st' 。

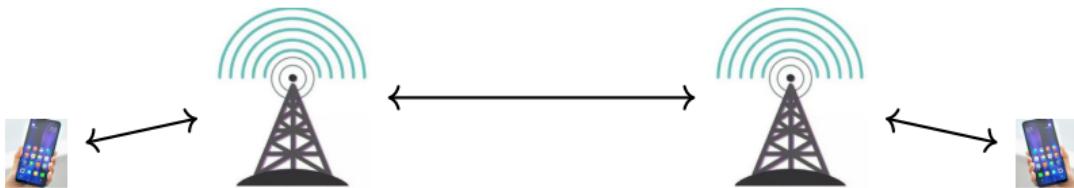
为了方便，往往定义 GetBits 函数来产生 ℓ 长的随机串：

- For $i = 1, \dots, \ell$, compute $(y_i, \text{st}_i) = \text{Next}(\text{st}_{i-1})$.
- Return the ℓ -bit string $y = y_1 \dots y_\ell$ and st_ℓ .

使用流密码来构建伪随机生成器和伪随机函数

- 使用不带初始向量 IV 的流密码 ($Init$, $Next$) 构建伪随机生成器。
- 给定希望的输出串长度 $\ell = \ell(n) > n$, 定义确定性函数 G^ℓ 为
$$G^\ell(s) \triangleq \text{GetBits}(\text{Init}(s), 1^\ell)$$
- 如果 G^ℓ 为伪随机生成器, 则称流密码是安全的。
- 也可以使用带初始向量 IV 的流密码 ($Init$, $Next$) 构建伪随机函数。
- 定义带密钥函数 $F^\ell: \{0, 1\}^n \times \{0, 1\}^n \mapsto \{0, 1\}^\ell$ 为
$$F_s^\ell(IV) \triangleq \text{GetBits}(\text{Init}(s, IV), 1^\ell)$$

流密码的其他用途：同步模式



- 通信双方 S 和 R 持有密钥 k , 初始化 $st_0 = \text{Init}(k)$
- 假设 S 的当前状态为 st_S , S 需要发送消息 m , 计算
$$(y, st'_S) = \text{GetBits}(st_S, 1^\ell)$$
将 $c = m \oplus y$ 发送给 R , 更新状态为 st'_S 。
- 假设 R 的当前状态为 st_R , 当收到密文 c 时, 计算
$$(y, st'_R) = \text{GetBits}(st_R, 1^\ell)$$
得到消息 $m = c \oplus y$, 更新状态为 st'_R 。
- 如果双方还持有密钥 k' , 就可以实现 R 到 S 的通信。

流密码的其他用途：异步模式

- 使用流密码 (Init, Next) 实现对任意长消息的私钥加密：
- Gen: 输入 1^n , 输出随机密钥 $k \leftarrow \{0, 1\}^n$
- Enc: 输入密钥 k 和任意长消息 $m \in \{0, 1\}^*$, 选择随机 $IV \leftarrow \{0, 1\}^n$, 计算

$$c = (IV, \text{GetBits}(\text{Init}(k, IV), 1^{|m|}) \oplus m)$$

- Dec: 输入密钥 k 和密文 (IV, c) , 计算

$$m = \text{GetBits}(\text{Init}(k, IV), 1^{|c|}) \oplus c$$

- 可以看到这种密码显然也满足 CPA 安全。

目录

1 伪随机生成器与流密码

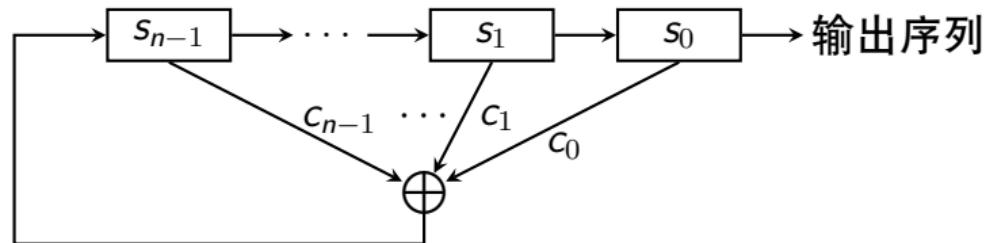
- 流密码的基本概念
- 线性反馈移位寄存器及非线性化
- 流密码的实现

2 伪随机置换与分组密码

3 多重加密

4 工作模式

线性反馈移位寄存器 (Linear Feedback Shift Register, LFSR)



- 一个 n 级线性反馈移位寄存器包含 n 个寄存器 s_{n-1}, \dots, s_0 ，构成当前状态 $s_{n-1} \dots s_0$ 。
- 反馈系数 $c_i \in \{0, 1\}$ ，状态更新满足递推关系

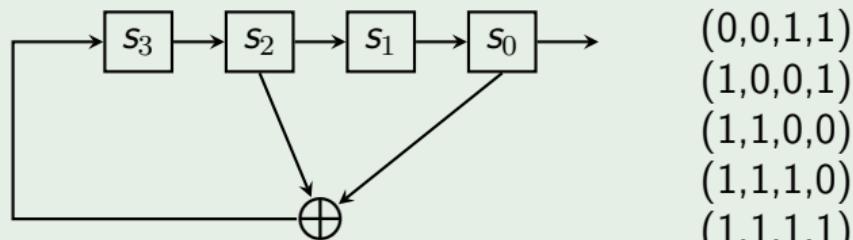
$$s_i^{(t+1)} = s_{i+1}^{(t)}, \quad i = 0, \dots, n-2 \quad s_{n-1}^{(t+1)} = \bigoplus_{i=0}^{n-1} c_i s_i^{(t)}$$

- 每个时钟输出最右侧寄存器的值。
- LFSR 实现简单、速度快、理论成熟。

LFSR 举例

例 (四级线性反馈移位寄存器)

初始状态 $(s_3, s_2, s_1, s_0) = 0011$

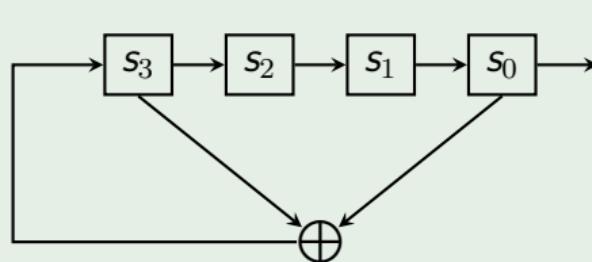


- 输出: 110011110... $(0,0,1,1) \leftarrow$ 开始重复
- 周期: 6

LFSR 举例

例 (四级线性反馈移位寄存器)

初始状态 $(s_3, s_2, s_1, s_0) = 0011$



(0,0,1,1)	(1,1,1,1)
(1,0,0,1)	(0,1,1,1)
(0,1,0,0)	(1,0,1,1)
(0,0,1,0)	(0,1,0,1)
(0,0,0,1)	(1,0,1,0)
(1,0,0,0)	(1,1,0,1)
(1,1,0,0)	(0,1,1,0)
(1,1,1,0)	(0,0,1,1) ← 开始重复

- 输出: 11001000111101011...
- 周期: 15

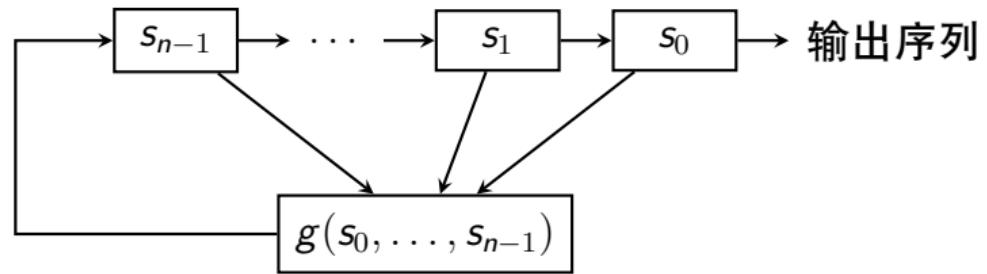
LFSR 的性质

- LFSR 输出序列的性质完全由反馈系数 c_i 确定。
- 假定 c_i 中至少有一个不为 0, 否则输出恒为 0。
- n 级 LFSR 的寄存器值 $s_{n-1} \cdots s_0$ 构成其状态, 共 2^n 个状态。
- n 级 LFSR 的状态周期小于等于 $2^n - 1$ 。
- 输出序列的周期等于状态周期, 也小于等于 $2^n - 1$ 。
- LFSR 输出序列的周期由反馈系数 c_i 确定, 选择合适的系数可使序列的周期达到最大值 $2^n - 1$ 。

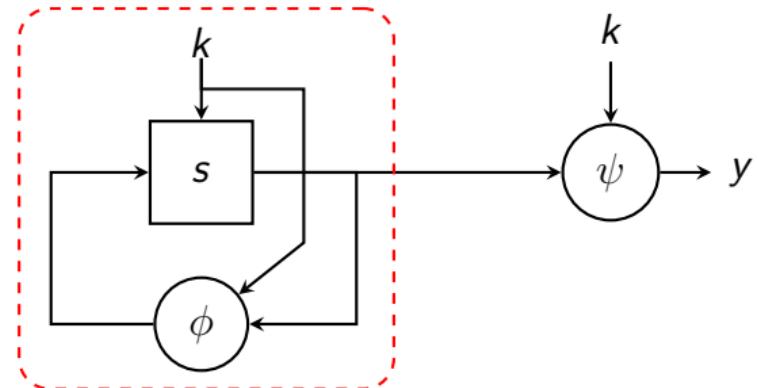
LFSR 的安全性

- 达到最大周期的 LFSR 产生的序列具有很好的统计特性：输出序列中 0 和 1 的个数近似相等。
- 但是 LFSR 产生的序列不能作为密钥流使用，有安全问题。
- 如果 LFSR 的反馈系数公开（根据 Kerckhoffs 准则），那么 LFSR 输出的前 n 个比特其实是 LFSR 的初始状态。
- 密码分析者根据初始状态和反馈系数可以完全预测 LFSR 未来的输出序列，使密钥流完全可预测。
- 如果不公开反馈系数，密码分析者可以由接下来 n 个输出构建 n 个线性方程，由线性方程组计算出反馈系数。
- 为了抵抗以上攻击，需要使输出序列非线性化。

增加非线性：非线性反馈



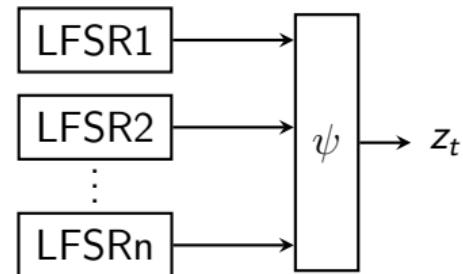
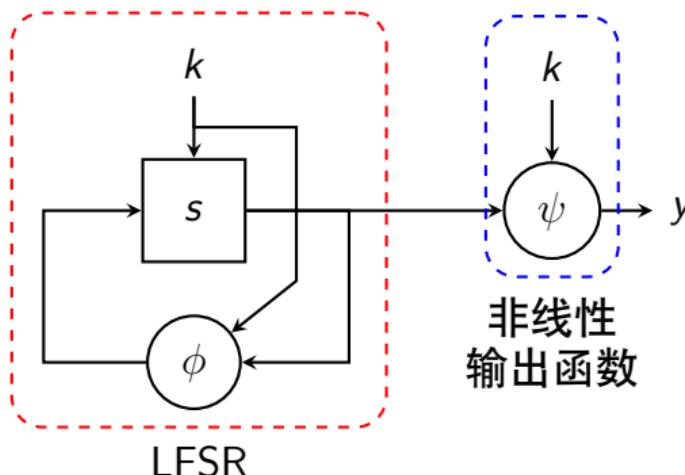
方式一：非线性反馈 + 简单输出函数



非线性反馈移位寄存器 (FSR)

增加非线性：非线性输出

- 方式二：线性反馈移位寄存器 + 非线性输出函数
- 方式三：多个线性反馈移位寄存器 + 非线性输出函数



- 方式二和方式三比方式一更易于分析和实现，是目前最为流行和实用的密钥流生成器工作方式。

目录

1 伪随机生成器与流密码

- 流密码的基本概念
- 线性反馈移位寄存器及非线性化
- 流密码的实现

2 伪随机置换与分组密码

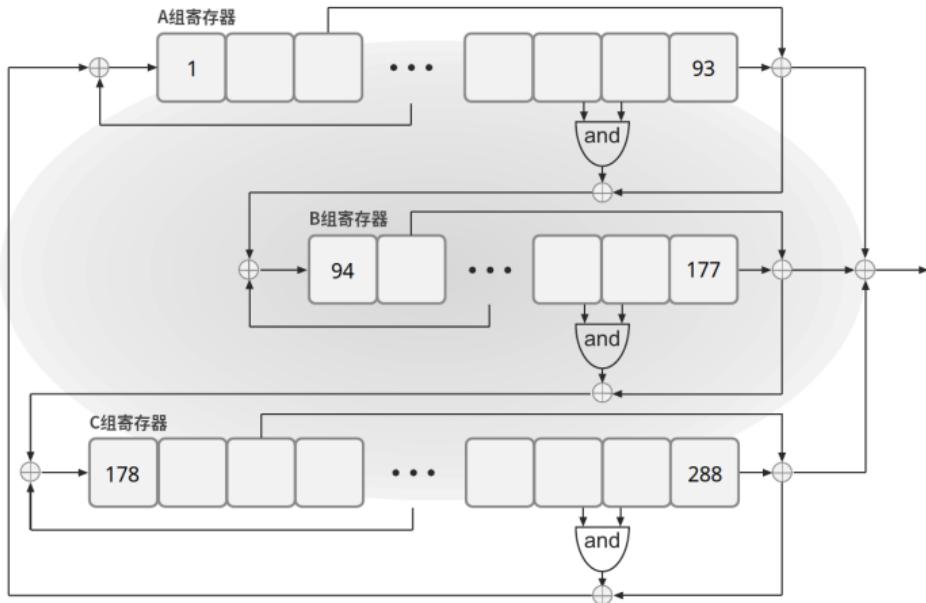
3 多重加密

4 工作模式

Trivium 流密码

- Trivium 流密码由比利时密码学家于 2008 年提出。
- 是一种轻量级流密码，可以使用较少的门电路实现，需要较少的计算、存储等资源，适宜应用于嵌入式等硬件系统。
- 包含三组相互耦合的非线性反馈移位寄存器 A , B 和 C ，级数分别为 93, 84 和 111，所以 Trivium 的状态共包含 288 比特。
- 密钥长度为 80 比特，初始向量 IV 为 80 比特。
- **反馈函数**：每组寄存器的部分寄存器值经过非线性运算，然后和下一组中某个寄存器值异或，作为下一组寄存器最左边的寄存器值。
- **输出函数**：每组寄存器最右边的寄存器值和本组中某个寄存器值分别异或后，再异或，作为输出密钥流。

Trivium 流密码



- **初始化**: 用 80 位密钥填充 A 最左边 80 位; 用 80 位 IV 填充 B 最左边 80 位; C 最右边 3 位置 1, 其余位置 0。
- **安全性**: 目前尚无比穷举攻击更好的攻击方法。

其他流密码设计

- RC4
 - 易于软件实现，由 Ron Rivest 与 1987 年提出。
 - 曾普遍应用于 802.11 无线网保密通信中的 WEP 加密方案，但现在已经不安全。
- ChaCha20：作为 RC4 的替代，目前仍可以安全使用。

目录

- 1 伪随机生成器与流密码
- 2 伪随机置换与分组密码
 - 理想分组密码
 - 扩散混淆范式及其实现
 - 数据加密标准 DES
 - 高级数据加密标准 AES
 - 差分分析和线性分析
- 3 多重加密
- 4 工作模式

目录

- 1 伪随机生成器与流密码
- 2 伪随机置换与分组密码
 - 理想分组密码
 - 扩散混淆范式及其实现
 - 数据加密标准 DES
 - 高级数据加密标准 AES
 - 差分分析和线性分析
- 3 多重加密
- 4 工作模式

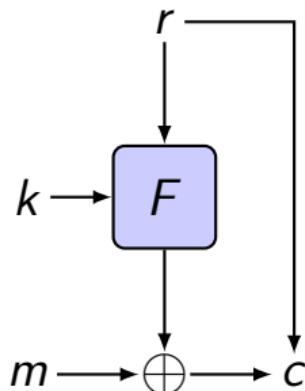
回顾：基于伪随机函数的 CPA 安全私钥密码

设计 (满足 CPA 安全的密码)

- F 为伪随机函数，按如下方式定义密码体制
 $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ ：
- **密钥生成函数**：输入 1^ℓ ，输出 $k \leftarrow \{0, 1\}^\ell$
- **加密**：输入密钥 k 和消息 $m \in \{0, 1\}^\ell$ ，采样一个随机数 $r \leftarrow \{0, 1\}^\ell$ ，输出密文

$$\text{Enc}_k(m) = (r, F_k(r) \oplus m)$$
- **解密**：输入密钥 k 和密文 $c = (r, s)$ ，输出明文

$$\text{Dec}_k(c) = F_k(r) \oplus s$$



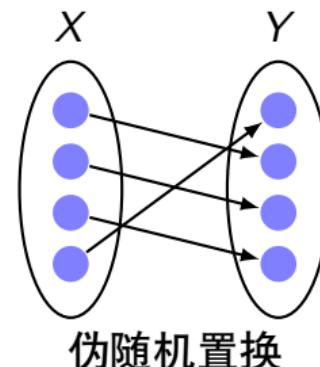
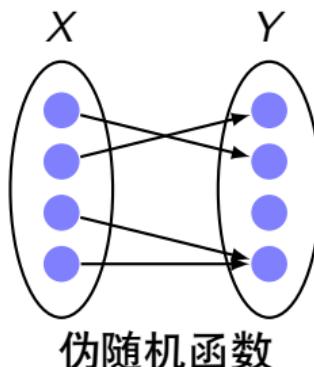
问题

？那么在实际中如何构造高效的伪随机函数（或伪随机置换）？

分组密码与伪随机置换

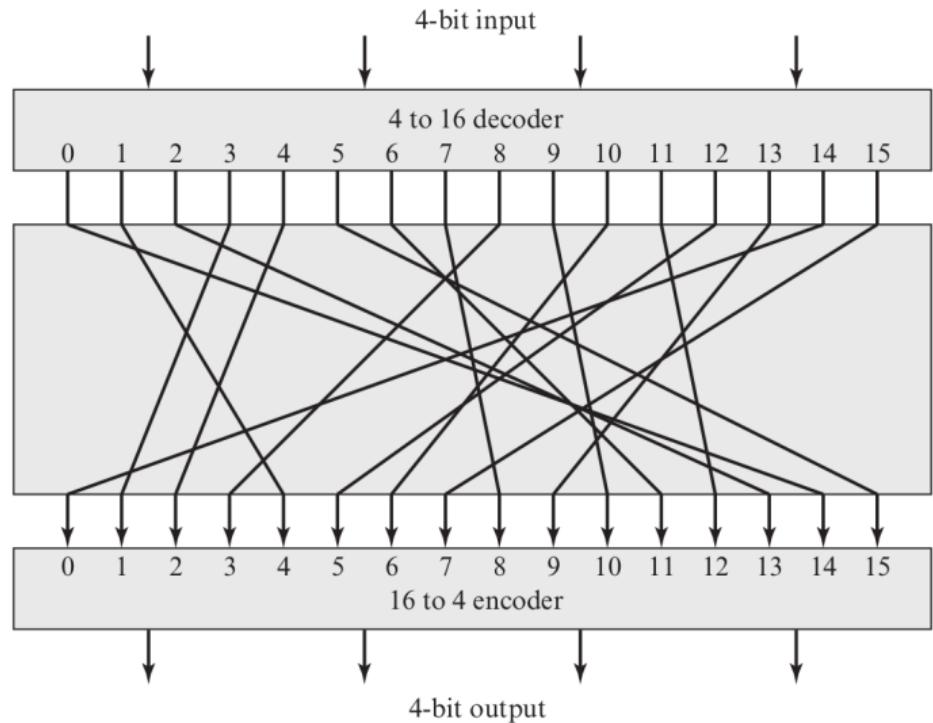
定义 (分组密码)

- 分组密码在本质上来说是一个高效的伪随机置换，记为 $F: \{0, 1\}^\ell \times \{0, 1\}^\ell \mapsto \{0, 1\}^\ell$ 或 $F_k(x) \triangleq F(k, x)$ 。
- 要求 $F_k(x)$ 是双射，当密钥 k 给定时，可高效计算 F_k^{-1} 。
- ℓ 是分组密码的分组长度，也是密钥长度。



分组密码不是密码体制，不能直接用于数据加密。

理想分组密码：4 位到 4 位的一种置换



？这里一共存在多少种不同置换方法？

理想分组密码

- 一共有 $2^\ell!$ 种置换方法。
- 表的第二列定义了 $2^\ell!$ 种置换中的某个特定置换，这个置换即为理想分组密码的密钥。
- 理想分组密码拥有最大的密钥空间 $2^\ell!$
- 密钥大小为 $\log(2^\ell!) \approx \ell \cdot 2^\ell$ 比特。
- 也就是保存表的第二列需要的存储空间。

Plaintext	Ciphertext
0000	1110
0001	0100
0010	1101
0011	0001
0100	0010
0101	1111
0110	1011
0111	1000
1000	0011
1001	1010
1010	0110
1011	1100
1100	0101
1101	1001
1110	0000
1111	0111

理想分组密码存在的问题

- 分组长度 ℓ 比较小，例如 $\ell = 8$ ，密码系统等价于单表代换密码，容易利用明文的统计信息攻击它。
- 如果 ℓ 充分大，那么明文的统计特征将被掩盖，从而不能利用明文的统计信息攻击这种分组密码。
- 对于 ℓ 位分组，密钥大小为 $\ell \cdot 2^\ell$ 比特。
 - 例如一个 64 位理想分组密码，密钥大小为 $64 \times 2^{64} = 2^{70}$ bit = 1Zb
 - 1ZB: global yearly Internet traffic in 2016.

问题

? 如何有效缩短密钥长度，使置换看起来仍像一个随机置换？

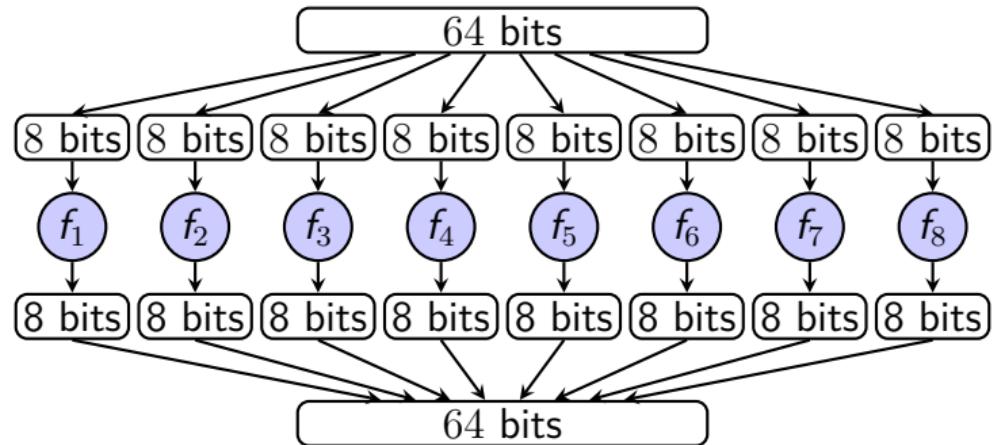
目录

- 1 伪随机生成器与流密码
- 2 伪随机置换与分组密码
 - 理想分组密码
 - 扩散混淆范式及其实现
 - 数据加密标准 DES
 - 高级数据加密标准 AES
 - 差分分析和线性分析
- 3 多重加密
- 4 工作模式

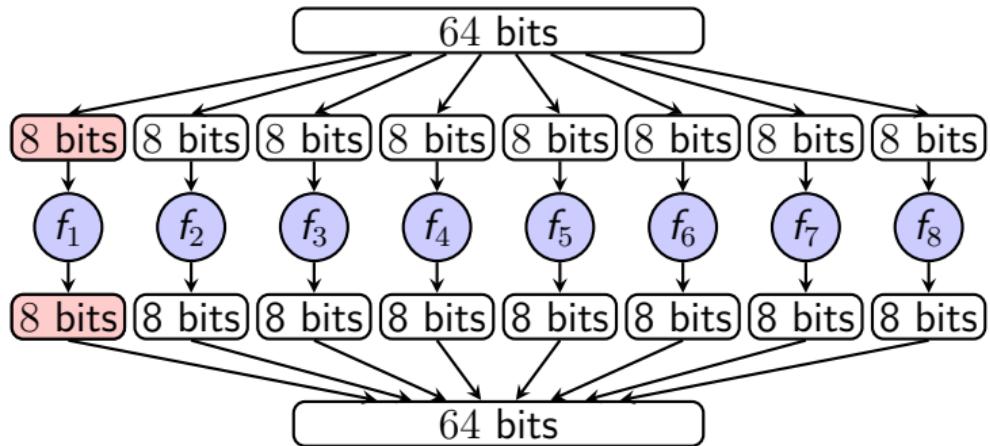
混淆扩散范式 (Confusion-Diffusion Paradigm)

- 假设目标是构建一个分组长度 $\ell = 64$ 比特的伪随机置换 F 。
- 使用 8 个分组长度是 8 比特的短分组伪随机置换 $\{f_i\}_{i=1}^8$

$$F_k(x) \triangleq f_1(x_1) \parallel f_2(x_2) \parallel \cdots \parallel f_8(x_8), \quad x = x_1 \cdots x_8$$
- 称 f_i 为轮函数，含参数 k ，作用是向 F 的计算结果引入混淆。
- 优点：密钥长度缩短为 $8 \times 8 \times 2^8 = 16\text{Kb}$ 。 

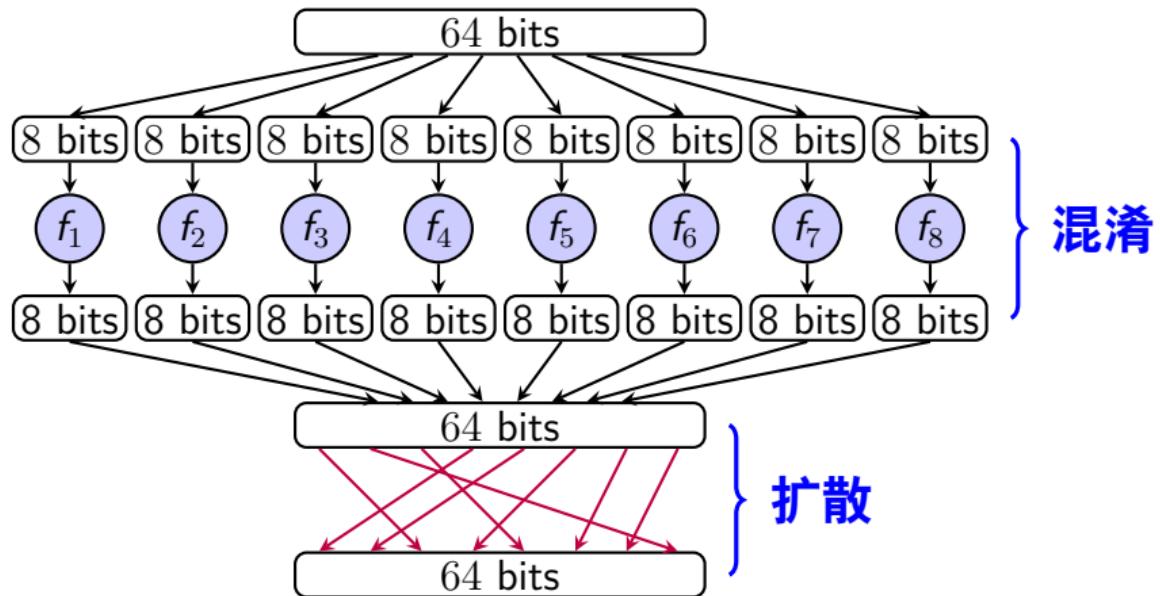


混淆扩散范式 (Confusion-Diffusion Paradigm)



- **缺点**: 如果 x 和 x' 只相差一个比特, 那么 $F_k(x)$ 和 $F_k(x')$ 只相差一个字节, 变化太集中, F_k 不是好的伪随机置换。
- 因此还需要**扩散**: 进一步打乱混淆后的结果, 使变化扩散开。
- 反复执行混淆和扩散, 执行多轮, 使 F_k 无限接近伪随机置换。这就是香农提出的**混淆扩散范式**。

混淆扩散范式 (Confusion-Diffusion Paradigm)

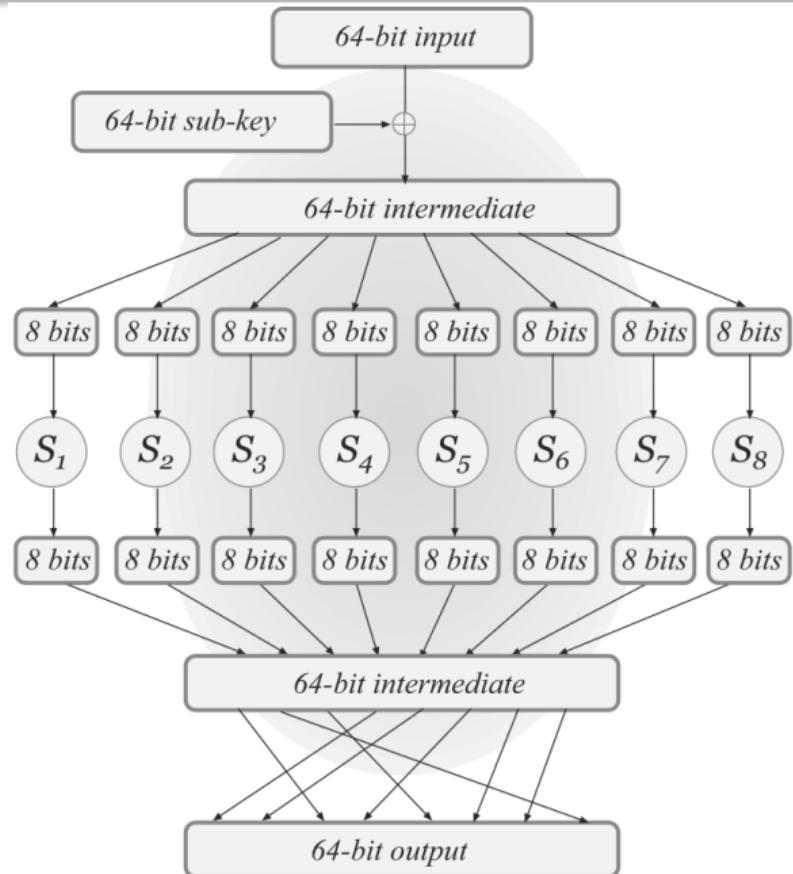


- 反复执行混淆和扩散，执行多轮，使 F_k 无限接近伪随机置换。这就是香农提出的混淆扩散范式。

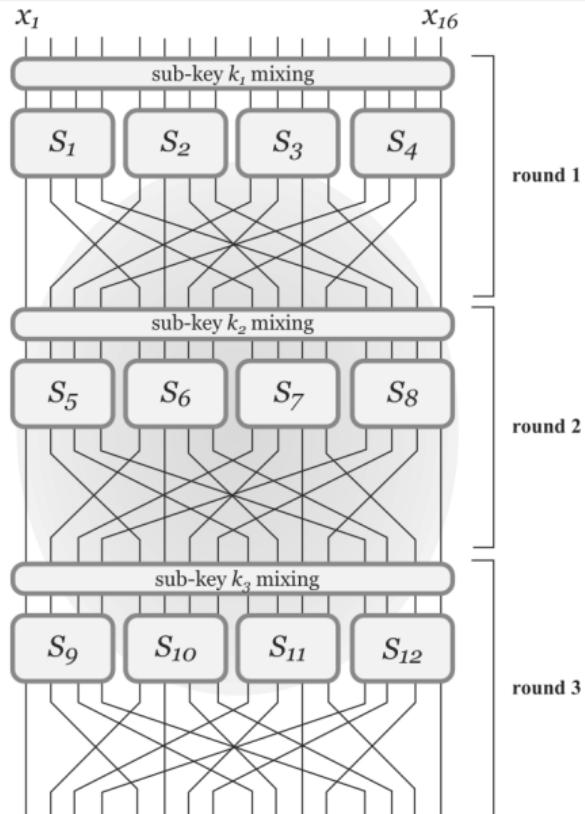
代替置换网络 (Substitution-Permutation Networks, SPN)

- 香农 1949 年提出的代替置换网络可以看作是混淆扩散范式的一种实现。
- 区别在于 SPN 中的轮函数 f 与密钥 k 无关, 是公开的置换函数, 称为 S 盒。
- 混淆扩散范式中的轮函数 f 和 SPN 中的 S 盒的关系为
$$f(x) = S(k \oplus x)$$
- SPN 同样进行多轮运算, 每轮的运算包括:
 - 轮密钥加: $x = x \oplus k$, k 是轮密钥, 由主密钥推导得到。
 - 代换操作: $x = S_1(x_1) \parallel \dots \parallel S_8(x_8)$, x_i 是 x 的第 i 字节。
 - 置换操作: 进一步打乱 x 各比特的位置。
- 经过最后一轮运算后, 还需进行一次额外轮密钥加运算, 然后得到最终密文 (否则最后一轮的代换和置换操作白费)。

代替置换网络的一轮运算

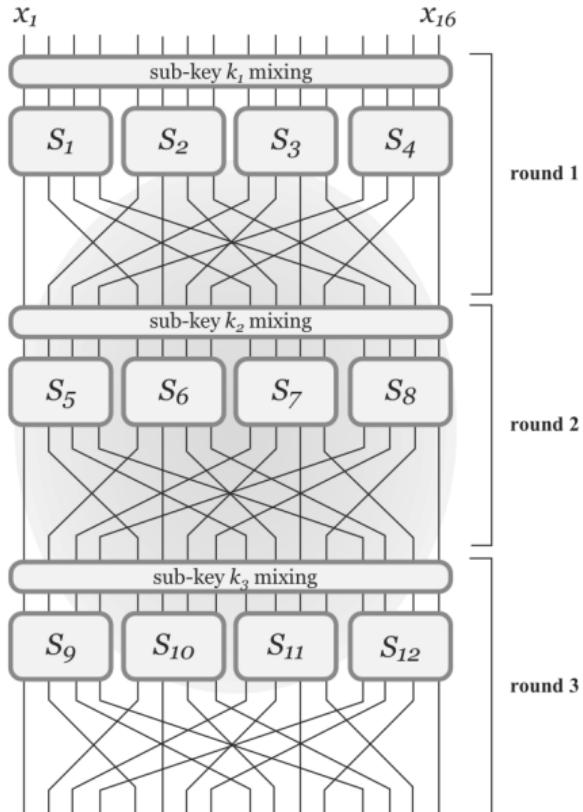


代替置换网络



- 基于一个简单的密钥编排算法由主密钥推导出每一轮的轮密钥。
- 一个包含 r 轮的 SPN 需要使用 $r + 1$ 个轮密钥。
- 给定密钥的情况下，SPN 是可逆的，因为每一轮的运算都可逆。
- SPN 的安全性由 S 盒、每轮的置换操作、密钥编排算法及总轮数决定。

雪崩效应 (Avalanche Effect)

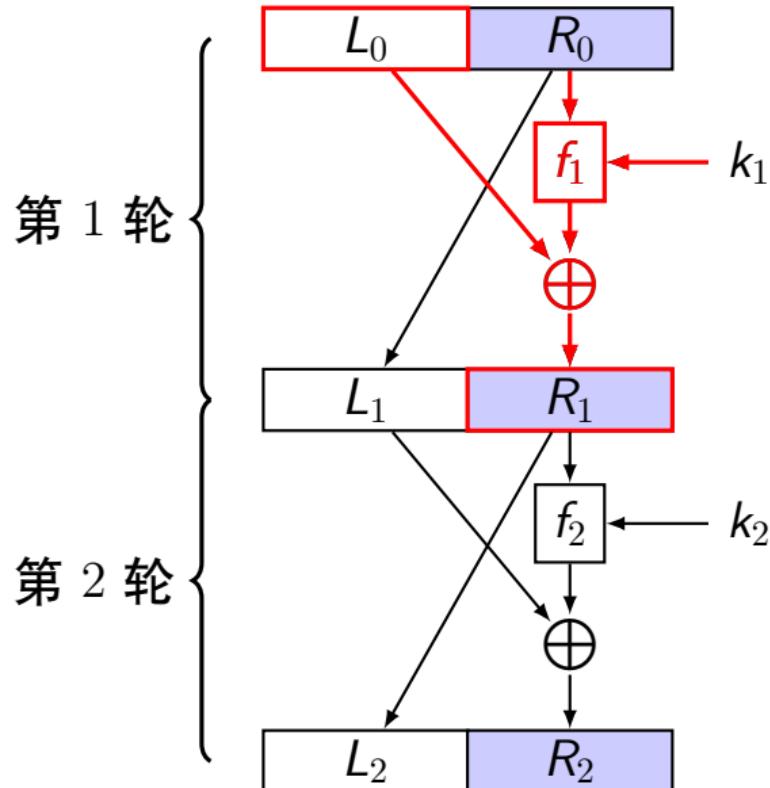


- **雪崩效应**: 分组密码输入的任何改变都能影响输出每一比特的改变。
- 在设计 SPN 时, 需满足以下条件以确保分组密码具有雪崩效应:
 - S 盒: 改变输入的任一比特, 输出至少有两个比特改变;
 - 置换操作: 每个 S 盒的输出能够影响下一轮多个 S 盒的输入。

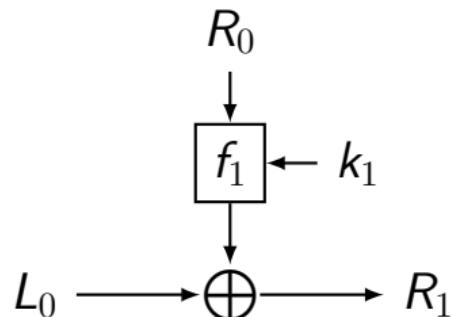
Feistel 网络 (Feistel Networks)

- Feistel 网络由德裔美国人 Horst Feistel (物理学家和密码学家) 在 1973 年提出。
- Feistel 在美国 IBM 工作期间完成此项开拓性研究, 目前大部分分组密码都使用该方案, 包括数据加密标准 (DES)。
- SPN 每一轮运算都要求可逆, 而 Feistel 网络不需要每轮运算都可逆。
- Feistel 网络的优点在于加密和解密操作非常相似, 在某些情况下甚至是相同的, 只需逆转密钥编排, 因此能够使代码或电路规模减半。
- 密码学家已经深入研究了 Feistel 网络的安全性。

Feistel 网络



L_{i-1}, R_{i-1}	输入分组
k_i	轮密钥
f_i	轮函数
L_i, R_i	输出分组



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

$$R_i = L_{i-1} \oplus f_i(R_{i-1})$$

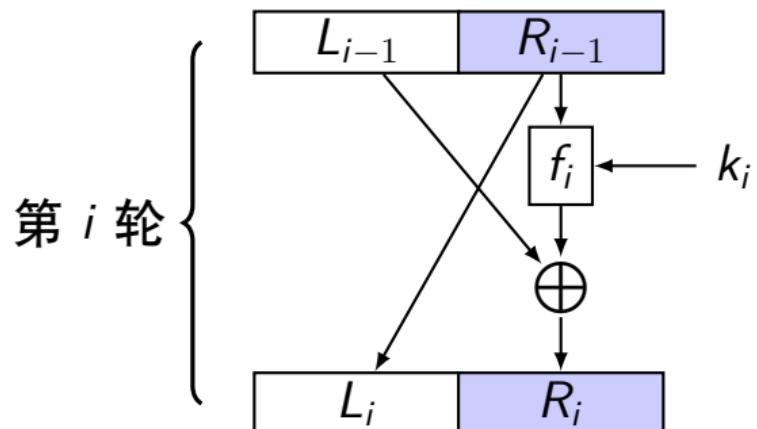
Feistel 网络逆运算

- 给定第 i 轮输出分组 (L_i, R_i) ，可以计算出第 i 轮输入分组 (L_{i-1}, R_{i-1})

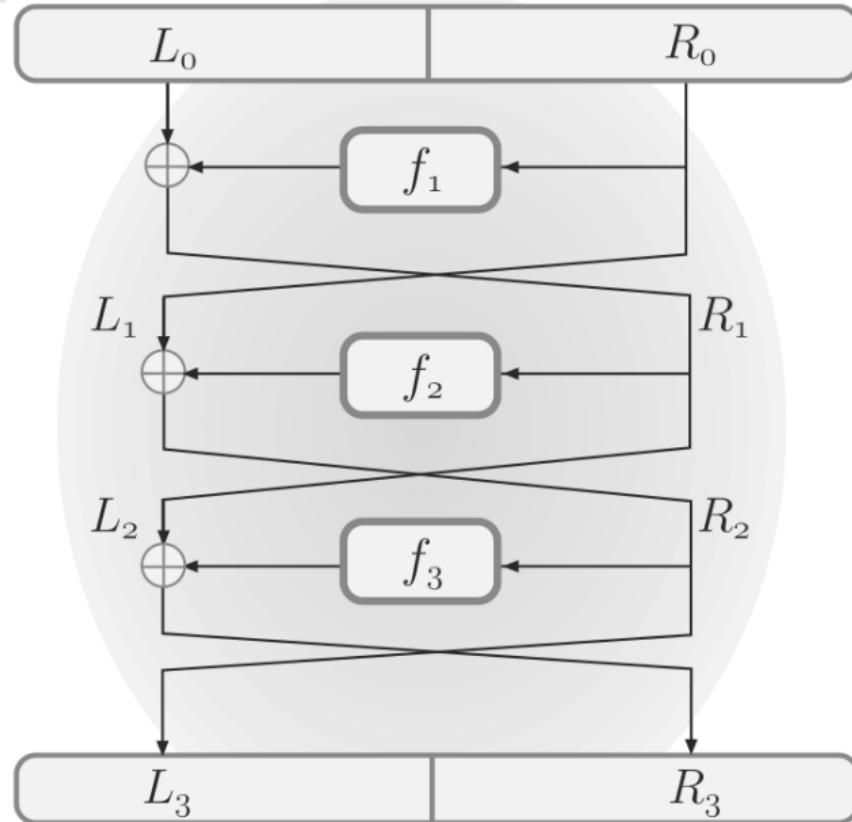
$$R_{i-1} = L_i$$

$$L_{i-1} = R_i \oplus f_i(R_{i-1})$$

- 注意，不需要 f_i 可逆。



Feistel 网络



目录

1 伪随机生成器与流密码

2 伪随机置换与分组密码

- 理想分组密码
- 扩散混淆范式及其实现
- **数据加密标准 DES**
- 高级数据加密标准 AES
- 差分分析和线性分析

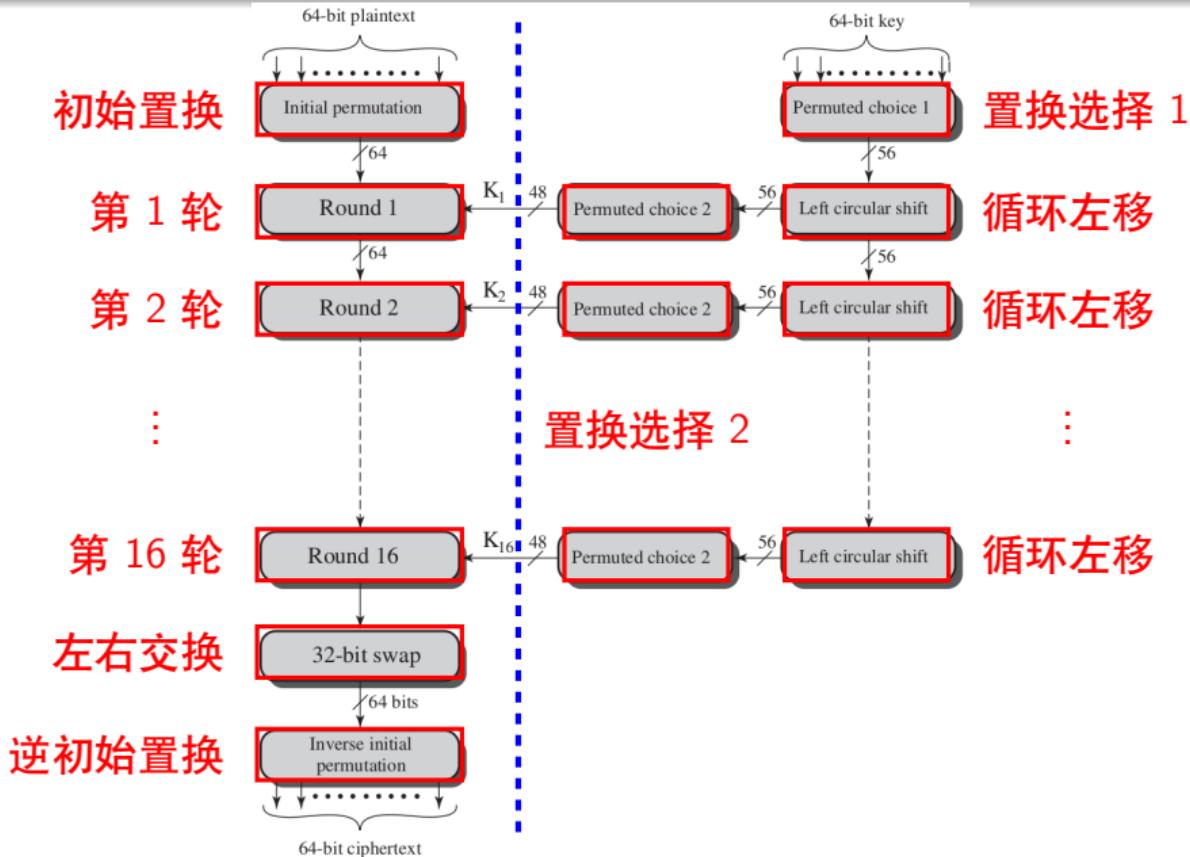
3 多重加密

4 工作模式

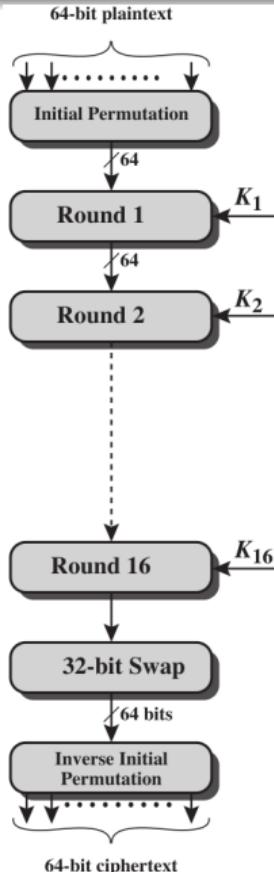
DES 的历史

- IBM 公司在 1971 年由 Horst Feistel 领导开发了 Lucifer Cipher，使用 128 位密钥加密 64 位分组。
- 1974 年，IBM 与 NSA 合作开发了 Lucifer 的一个修订版，易于在芯片上实现，抗密码分析能力更强，密钥缩短为 56 位。
- 1977 年，这个加密方案成为美国国家密码标准，称为 DES。
- DES 在密码学领域被深入分析，是一个非常经典的分组密码，还没有发现比穷举攻击更有效的攻击方法。
- 穷举攻击 DES 的搜索空间为 2^{56} ，现在已经不安全。分组长度 $\ell = 64$ 位也被认为太短，不能满足现代需要。
- 目前 DES 已经被淘汰，替代算法包括 3DES、AES 等。

DES 结构简介



DES 加密过程



- DES 的明文长 64 位, 密钥长 56 位 (虽然输入 64 位密钥, 但内部仅使用了 56 位);
- 明文处理经过三个阶段:
 - 首先 64 位明文经过初始置换 (IP) 而被重新排列;
 - 然后进行 16 轮相同函数的作用, 每轮都进行代替和置换;
 - 最后一轮输出 64 位分组, 左右互换产生预输出, 经过逆初始置换 (IP^{-1}) 产生 64 位密文。
- 除了初始和末尾的置换操作, DES 的结构与 Feistel 网络结构完全相同。

初始置换 IP 和逆初始置换 IP^{-1}

(a) Initial Permutation (IP)

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

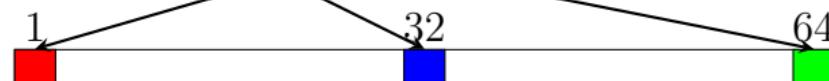
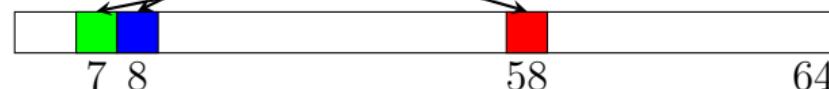
(b) Inverse Initial Permutation (IP^{-1})

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

64 位输入分组



IP 后分组

IP⁻¹ 后分组

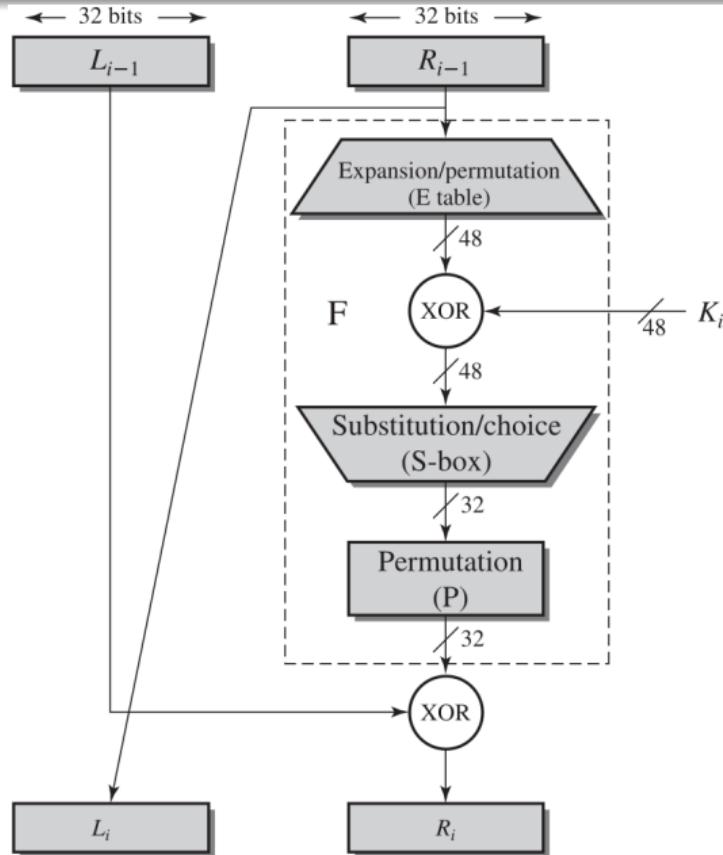
初始置换 IP 和逆初始置换 IP⁻¹

- 初始置换和逆置换并不能增强 DES 的安全性，而是为了方便硬件电路实现¹。
- 当总线宽度为 8 位时，需要配合使用 8 个 8 位移位寄存器，经过 8 个时钟得到 64 位输入分组。
- 如果不使用初始置换，那么从 8 个寄存器取前 32 位和后 32 位时，会在电路连线中产生很多交叉。

8 位 总线	1	57	49	41	33	25	17	9	1	R ₁
	2	58	50	42	34	26	18	10	2	R ₂
	3	59	51	43	35	27	19	11	3	R ₃
	4	60	52	44	36	28	20	12	4	R ₄
	5	61	53	45	37	29	21	13	5	R ₅
	6	62	54	46	38	30	22	14	6	R ₆
	7	63	55	47	39	31	23	15	7	R ₇
	8	64	56	48	40	32	24	16	8	R ₈

¹<https://crypto.stackexchange.com/a/6>

每一轮的运算



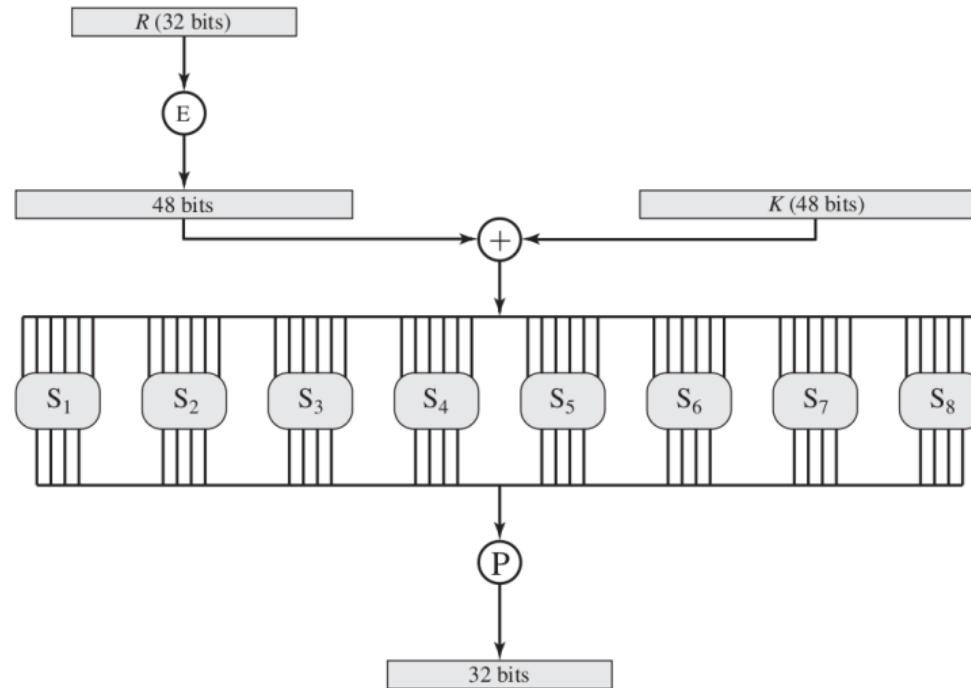
- 第 i 轮的输入分成左右两部分 L_{i-1} 和 R_{i-1} ；
- 做如下运算得到本轮输出的左右两部分 L_i 和 R_i ：

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

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

其中 \oplus 表示异或运算。

轮函数 F

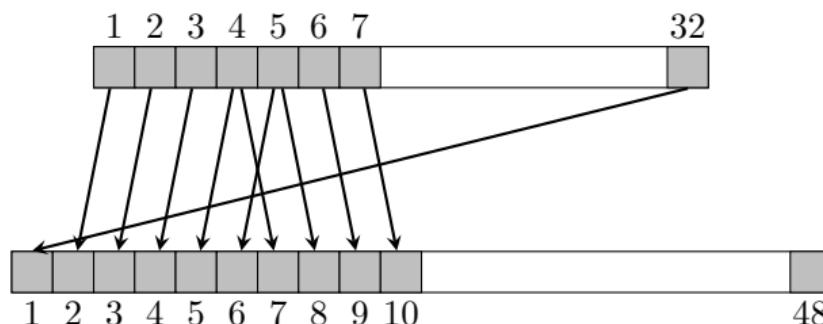


轮函数 F 是 DES 的核心运算函数，包含 4 步运算：扩展置换函数 E 、与子密钥异或、 S 盒替换和置换函数 P 。

扩展置换函数 E

使用置换表 E 将 32 位 R 扩展成 48 位，起扩散作用。

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1



S 盒替换

- 48 位结果送给 8 个替换盒 S_1, \dots, S_8 , 得到 32 位结果;

S_1	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
S_2	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
S_3	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S_4	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S 盒替换

S_5	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S_6	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S_7	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S_8	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

S 盒替换

- S 盒是轮函数 F 的核心，每个 S 盒输入 6 位，输出 4 位。
- 作用是混淆，即通过 S 盒替换使密文和密钥之间的关系尽可能复杂。
- 每个 S 盒输入的第一位和最后一位组成一个 2 位二进制数用来选择 S 盒 4 行中的某一行，中间 4 位用来选择 16 列中的某一列。
- 行列对应的十进制数转换为二进制后可得到输出的 4 位二进制数。

例

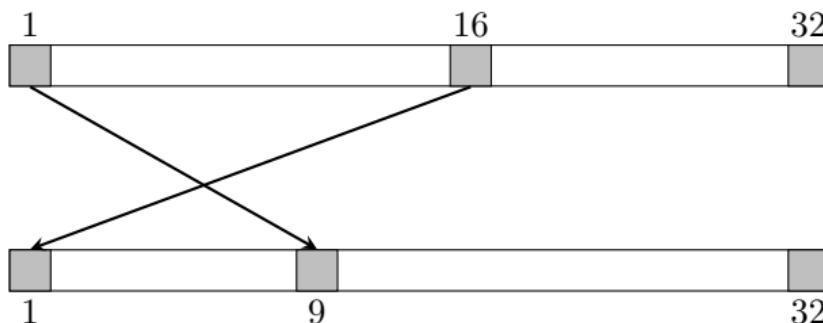
例如，在 S_1 中，若输入为 011001，则行是 1(01)，列是 12(1100)，该处的值为 9，所以输出 1001。

置换函数 P

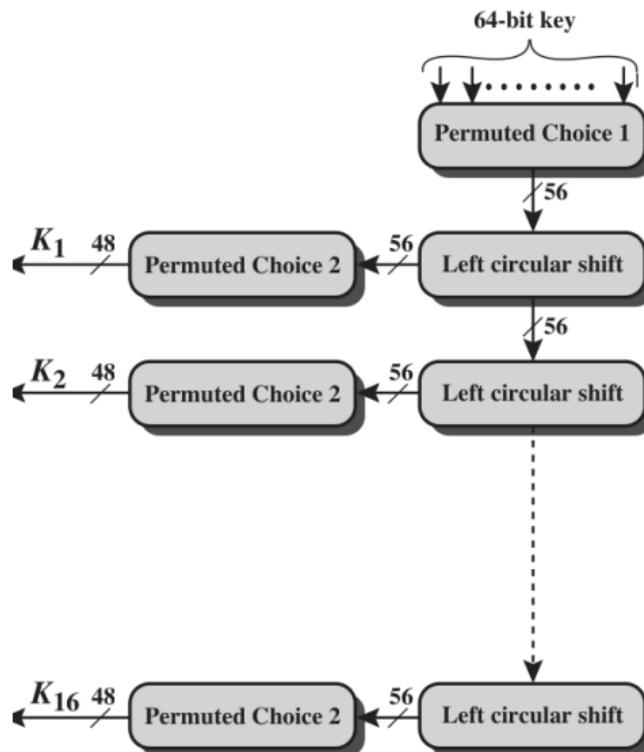
- 最后使用 32 位置换表 P ，把 32 位结果再进行一次置换处理。

(d) Permutation Function (P)

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

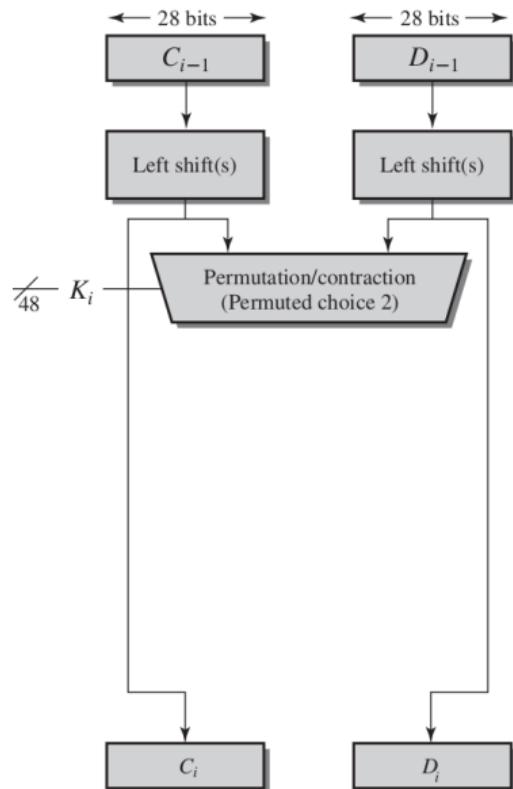


密钥编排



- 密钥经过一个置换，然后循环左移，再经过另一个置换，得到各轮的子密钥 K_i ；
- 每轮的置换函数都一样，由于循环左移，使得各轮子密钥各不相同。

密钥编排



- 每一轮都要依据输入密钥生成一个子密钥以供加密使用；
- 输入密钥为 64 位，DES 只使用其中 56 位，其余位可用于奇偶校验；
- 使用**置换选择 1 (PC-1)**，将 56 位密钥分成两半 C 和 D ，每部分 28 位；
- 根据**循环左移表**将这两半分别循环左移 1 位或 2 位；
- 使用**置换选择 2 (PC-2)**，形成 48 位子密钥，用在轮函数 F 中。

密钥编排：主密钥

(a) Input Key

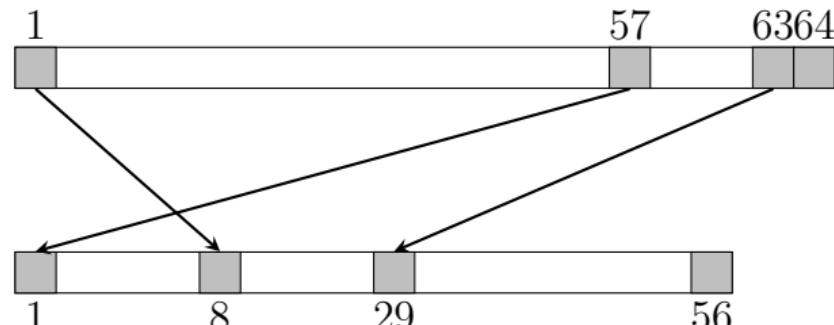
1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

- 为了与输入明文分组长度一致，DES 密钥长度为 64 位，实际只使用 56 位，每个字节的最后一位在 DES 算法中没有使用，可用于校验等其他功能。

密钥编排：置换选择 1

(b) Permuted Choice One (PC-1)

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4



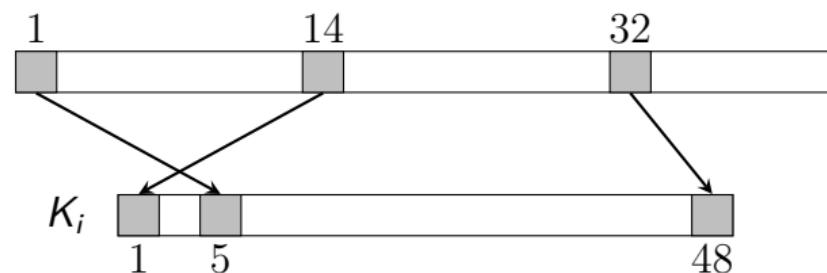
密钥编排：循环左移表与置换选择 2

(d) Schedule of Left Shifts

Round Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bits Rotated	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

(c) Permuted Choice Two (PC-2)

14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32



DES 举例

Plaintext:	02468aceeca86420
Key:	0f1571c947d9e859
Ciphertext:	da02ce3a89ecac3b

Round	K_i	L_i	R_i
IP		5a005a00	3cf03c0f
1	1e030f03080d2930	3cf03c0f	bad22845
2	0a31293432242318	bad22845	99e9b723
3	23072318201d0c1d	99e9b723	0bae3b9e
4	05261d3824311a20	0bae3b9e	42415649
5	3325340136002c25	42415649	18b3fa41
6	123a2d0d04262a1c	18b3fa41	9616fe23
7	021f120b1c130611	9616fe23	67117cf2
8	1c10372a2832002b	67117cf2	c11bfc09
9	04292a380c341f03	c11bfc09	887fbc6c
10	2703212607280403	887fbc6c	600f7e8b
11	2826390c31261504	600f7e8b	f596506e
12	12071c241a0a0f08	f596506e	738538b8
13	300935393c0d100b	738538b8	c6a62c4e
14	311e09231321182a	c6a62c4e	56b0bd75
15	283d3e0227072528	56b0bd75	75e8fd8f
16	2921080b13143025	75e8fd8f	25896490
IP⁻¹		da02ce3a	89ecac3b

Note: DES subkeys are shown as eight 6-bit values in hex format

DES 的雪崩效应：改变明文

使用相同密钥加密两个只差 1 比特的明文：

Round		δ
	02468aceeca86420 12468aceeca86420	1
1	3cf03c0fbad22845 3cf03c0fbad32845	1
2	bad2284599e9b723 bad3284539a9b7a3	5
3	99e9b7230bae3b9e 39a9b7a3171cb8b3	18
4	0bae3b9e42415649 171cb8b3ccaca55e	34
5	4241564918b3fa41 ccaca55ed16c3653	37
6	18b3fa419616fe23 d16c3653cf402c68	33
7	9616fe2367117cf2 cf402c682b2cefbc	32
8	67117cf2c11bfc09 2b2cefbc99f91153	33

Round		δ
9	c11bfc09887fbc6c 99f911532eed7d94	32
10	887fbcc6c600f7e8b 2eed7d94d0f23094	34
11	600f7e8bf596506e d0f23094455da9c4	37
12	f596506e738538b8 455da9c47f6e3cf3	31
13	738538b8c6a62c4e 7f6e3cf34bc1a8d9	29
14	c6a62c4e56b0bd75 4bc1a8d91e07d409	33
15	56b0bd7575e8fd8f 1e07d4091ce2e6dc	31
16	75e8fd8f25896490 1ce2e6dc365e5f59	32
IP⁻¹	da02ce3a89ecac3b 057cde97d7683f2a	32

DES 的雪崩效应：改变密钥

使用相差 1 比特的两个密钥 (0f1571c947d9e859 与 1f1571c947d9e859) 加密相同明文：

Round		δ
	02468aceeca86420 02468aceeca86420	0
1	3cf03c0fbad22845 3cf03c0f9ad628c5	3
2	bad2284599e9b723 9ad628c59939136b	11
3	99e9b7230bae3b9e 9939136b768067b7	25
4	0bae3b9e42415649 768067b75a8807c5	29
5	4241564918b3fa41 5a8807c5488dbe94	26
6	18b3fa419616fe23 488dbe94aba7fe53	26
7	9616fe2367117cf2 aba7fe53177d21e4	27
8	67117cf2c11bfc09 177d21e4548f1de4	32

Round		δ
9	c11bfc09887fbcc6c 548f1de471f64dfd	34
10	887fbcc6c600f7e8b 71f64dfd4279876c	36
11	600f7e8bf596506e 4279876c399fdc0d	32
12	f596506e738538b8 399fdc0d6d208dbb	28
13	738538b8c6a62c4e 6d208dbbb9bdeea	33
14	c6a62c4e56b0bd75 b9bdeeaad2c3a56f	30
15	56b0bd7575e8fd8f d2c3a56f2765c1fb	33
16	75e8fd8f25896490 2765c1fb01263dc4	30
IP⁻¹	da02ce3a89ecac3b ee92b50606b62b0b	30

目录

1 伪随机生成器与流密码

2 伪随机置换与分组密码

- 理想分组密码
- 扩散混淆范式及其实现
- 数据加密标准 DES
- **高级数据加密标准 AES**
- 差分分析和线性分析

3 多重加密

4 工作模式

AES 的历史

- DES 不安全, 建议用 3DES, 密钥 168 位, 抵御密码分析攻击, 但是 3DES 用软件实现速度较慢, 分组仅 64 位。
- 美国国家标准技术协会 NIST 在 1997 年征集新标准, 要求明文分组长度 128 位, 密钥长 128、192 或 256 位。
- 1998 年 6 月, 15 种候选算法通过了第一轮评估; 1999 年 8 月, 仅有 5 个候选算法通过了第二轮评估。
- 2000 年 10 月, NIST 选择 Rijndael 算法作为 AES 算法, Rijndael 的作者是比利时的密码学家 Joan Daemen 博士和 Vincent Rijmen 博士。
- 2001 年 11 月 NIST 发布了 AES 的最终标准 FIPS PUB 197。
- AES 目前被广泛使用, 未发现存在重大安全问题。

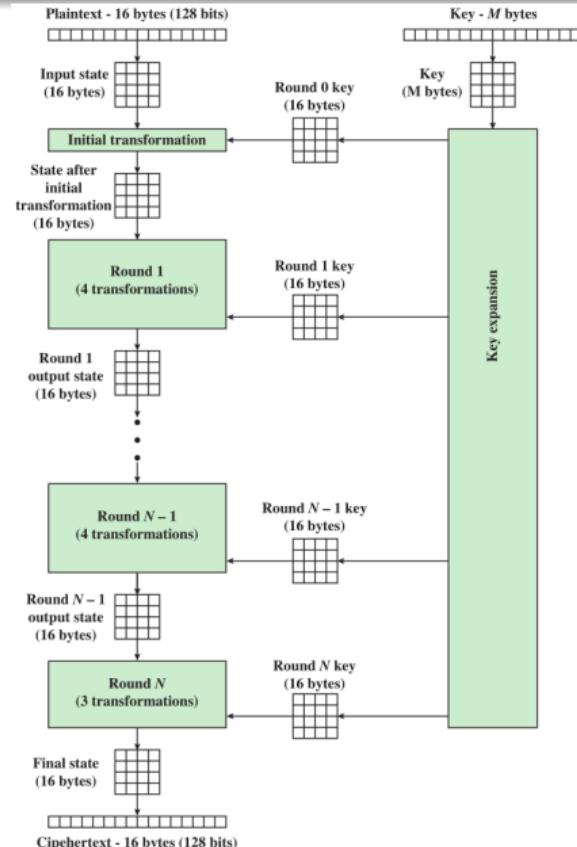
AES 参数选择

- AES 的分组长度为 128 位。
- 密钥长度可以是 128/192/256 的任意一种，根据使用的密钥长度，分为 AES-128，AES-192 或 AES-256。
- 接下来主要以 AES-128 为例进行讲述。

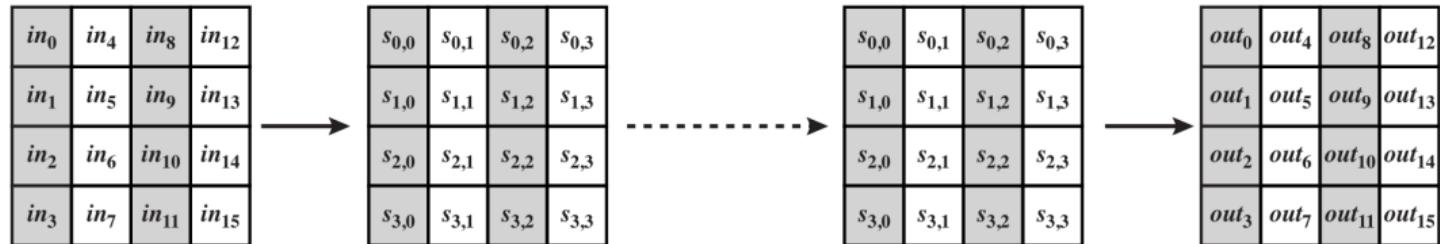
Key Size (words/bytes/bits)	4/16/128	6/24/192	8/32/256
Plaintext Block Size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Number of Rounds	10	12	14
Round Key Size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Expanded Key Size (words/bytes)	44/176	52/208	60/240

AES 加密过程总体结构

- AES 具有典型的 SPN 结构特点。
- 输入 128 位分组，表示为 4×4 字节方阵，称为状态数组。
- 密钥被表示为 4×4 密钥方阵，扩展为密钥字阵列，共包含 $N + 1$ 个密钥方阵。
- 加密由 N 轮构成，前 $N - 1$ 轮由 4 种不同变换组成，最后一轮仅包含 3 种变换。
- 每一轮修改状态数组，最后一轮输出密文。

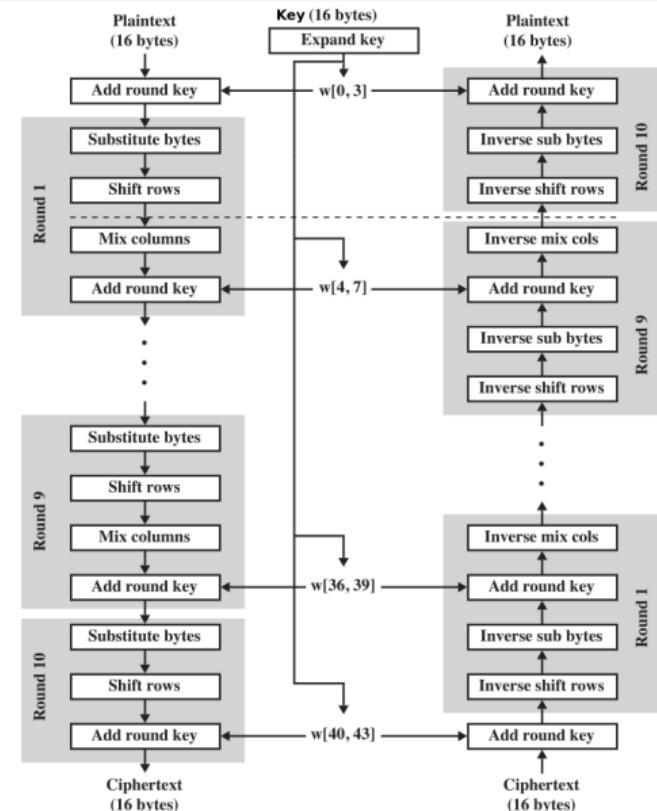


状态数组的修改及密钥编排



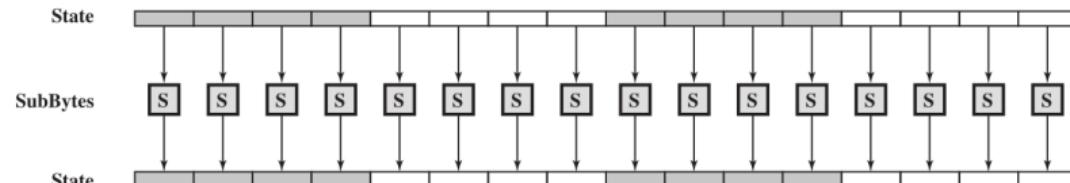
AES 详细结构

- 输入密钥被扩展为 44 字节数组 w 。
- 每轮运算包含 4 种操作：
 - 字节代替**: S 盒完成字节代替
 - 行移位**: 一个简单置换操作
 - 列混淆**: 域 $GF(2^8)$ 上的算术
 - 轮密钥加**: 当前分组与扩展密钥一部分按位异或
- 每种操作均可逆，仅仅在轮密钥加阶段使用密钥。
- AES 的加解密算法不同，加解密的最后一轮均只包含 3 种操作。

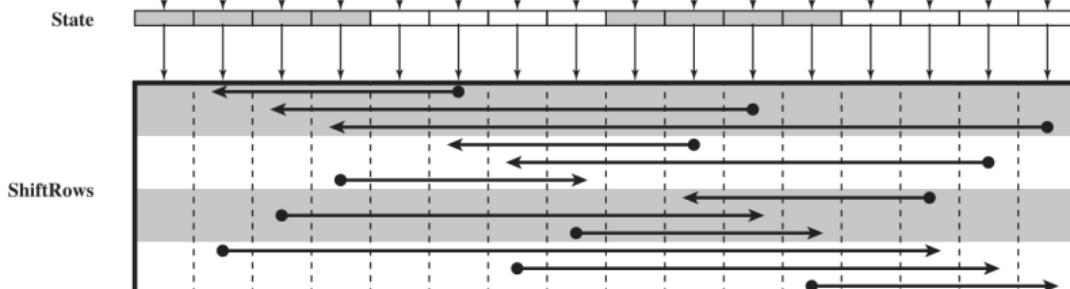


AES 的一轮加密过程

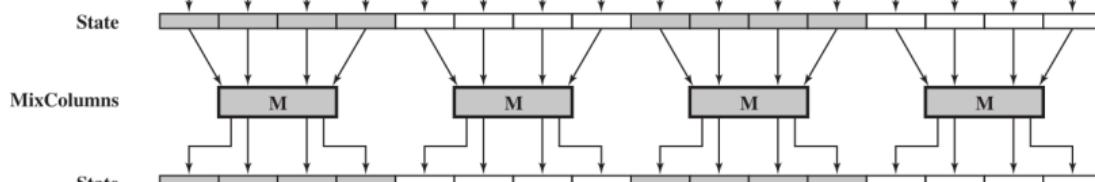
字节代替



行移位

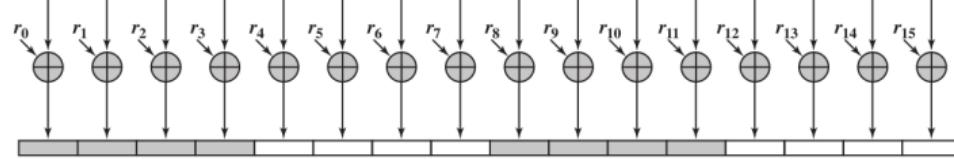


列混淆



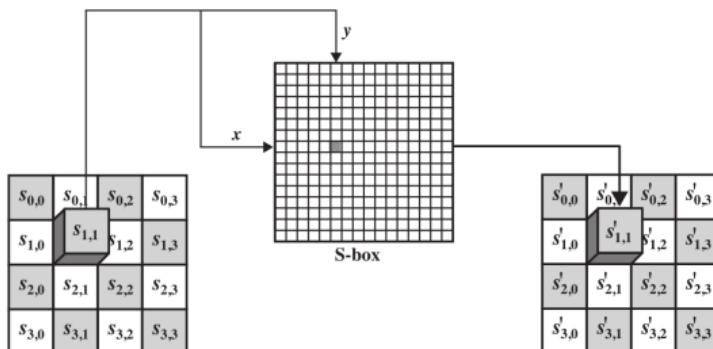
轮密钥加

AddRoundKey



字节代替变换

- 字节代替变换是一个查表操作，实现分组字节到字节的代替。
- S 盒是一个 16×16 字节表，包含了所有 8 位的置换值。
- 每个字节的左 4 位选择行，右 4 位选择列来查表替换。



EA	04	65	85
83	45	5D	96
5C	33	98	B0
F0	2D	AD	C5

→

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

S 盒

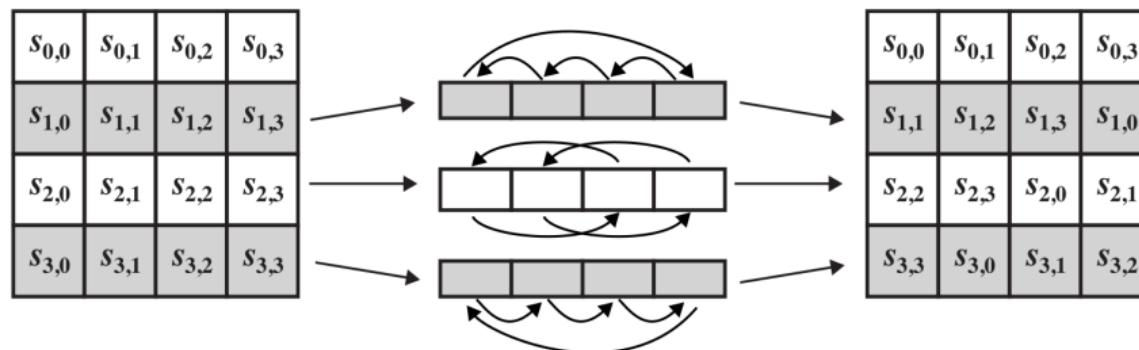
		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0	
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15	
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75	
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	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	

逆 S 盒

		y																	
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
x	0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB		
	1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB		
	2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E		
	3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25		
	4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92		
	5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84		
	6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06		
	7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B		
	8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73		
	9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E		
	A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B		
	B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4		
	C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F		
	D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF		
	E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61		
	F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D		

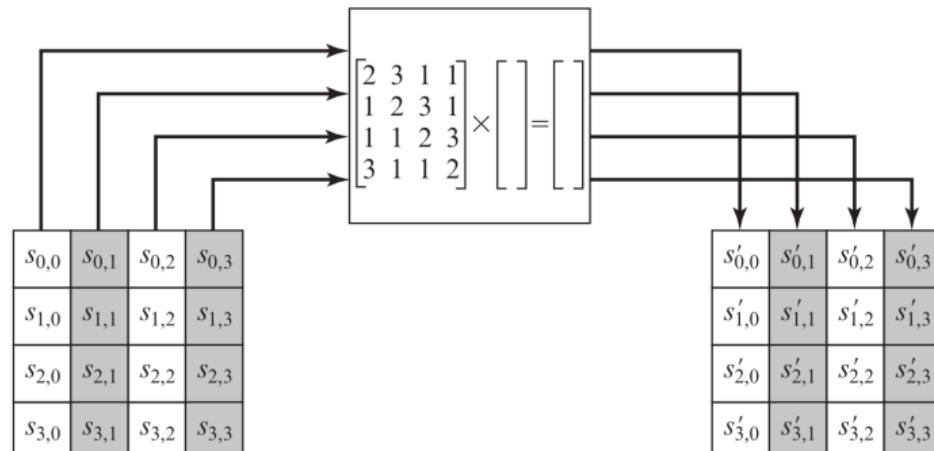
行移位变换

- **正向行移位变换**：第 1 行保持不变。第 2 行循环左移 1 个字节。第 3 行循环左移 2 个字节。第 4 行循环左移 3 个字节。
- **逆向行移位变换**进行相反的移位以实现解密。
- 因为状态矩阵是按列处理的，行移位变换就把字节在列之间进行了置换。



列混淆变换

- 正向列混淆变换对每列独立进行操作。
- 每列中的每个字节被映射为一个新值，由该列中的 4 个字节通过函数变换得到。
- 可以用状态矩阵乘法表示，使用素多项式
 $m(x) = x^8 + x^4 + x^3 + x + 1$ ，加法和乘法都定义在 $GF(2^8)$ 上。



列混淆变换

- 写成矩阵形式为

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} s_{00} & s_{01} & s_{02} & s_{03} \\ s_{10} & s_{11} & s_{12} & s_{13} \\ s_{20} & s_{21} & s_{22} & s_{23} \\ s_{30} & s_{31} & s_{32} & s_{33} \end{bmatrix} = \begin{bmatrix} s'_{00} & s'_{01} & s'_{02} & s'_{03} \\ s'_{10} & s'_{11} & s'_{12} & s'_{13} \\ s'_{20} & s'_{21} & s'_{22} & s'_{23} \\ s'_{30} & s'_{31} & s'_{32} & s'_{33} \end{bmatrix}$$

例如 $s'_{0j} = 2s_{0j} \oplus 3s_{1j} \oplus s_{2j} \oplus s_{3j}$

- 逆向列混淆变换也由矩阵乘法定义：

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \cdot \begin{bmatrix} s_{00} & s_{01} & s_{02} & s_{03} \\ s_{10} & s_{11} & s_{12} & s_{13} \\ s_{20} & s_{21} & s_{22} & s_{23} \\ s_{30} & s_{31} & s_{32} & s_{33} \end{bmatrix} = \begin{bmatrix} s'_{00} & s'_{01} & s'_{02} & s'_{03} \\ s'_{10} & s'_{11} & s'_{12} & s'_{13} \\ s'_{20} & s'_{21} & s'_{22} & s'_{23} \\ s'_{30} & s'_{31} & s'_{32} & s'_{33} \end{bmatrix}$$

轮密钥加变换

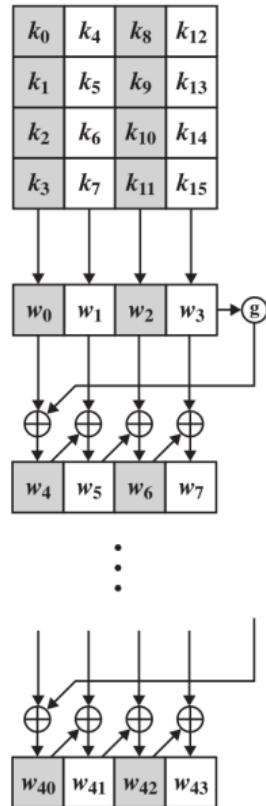
- 正向轮密钥加变换中 128 位的状态矩阵按位与 128 位的密钥异或。
- 都是基于状态矩阵列的操作，即把状态矩阵的一列中的 4 个字节与轮密钥的 1 个字进行异或。
- 逆向轮密钥加变换与正向轮密钥加变换相同，因为异或操作是其本身的逆。
- 轮密钥加变换非常简单，却能影响状态矩阵中的每一位。

密钥编排算法

- AES 密钥编排算法的输入是 4 字，输出 44 字。
- 输入密钥直接被复制到扩展密钥数组的前 4 个字。
- 然后每次用 4 字填充扩展密钥数组余下的部分：

$$w[i] = \begin{cases} w[i-4] \oplus w[i-1] & i \bmod 4 \neq 0 \\ w[i-4] \oplus g(w[i-1]) & i \bmod 4 = 0 \end{cases}$$

其中 g 函数对一个字进行复杂变换，输出另外一个字。

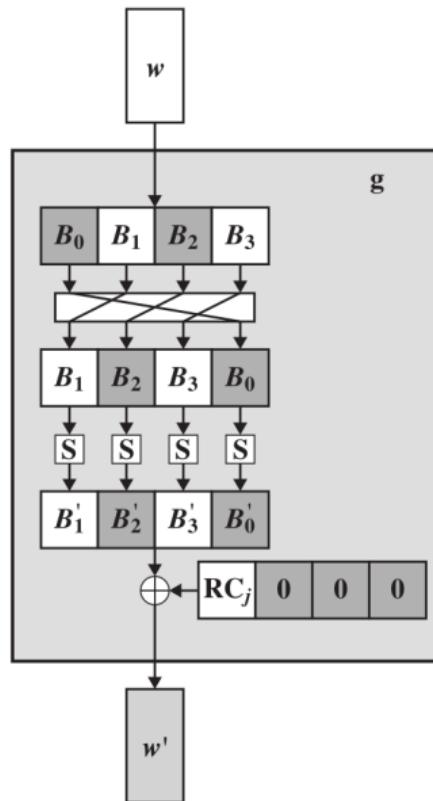


密钥编排算法: g 函数

- 首先, **字循环**使一个字中的 4 个字节循环左移一个字节, 即将输入字 $[B_0, B_1, B_2, B_3]$ 变为 $[B_1, B_2, B_3, B_0]$ 。
- 然后, **字代替**利用 S 盒对输入字中的每个字节进行字节代替。
- 最后, 将前两步的结果与轮常量 $Rcon_j = [RC_j, 0, 0, 0]$ 相**异或**。
 - 轮常量是一个字, 最右 3 字节恒为 0;
 - 轮常量第一个字节定义为

$$RC_1 = 1, \quad RC_j = 2 \cdot RC_{j-1}$$

j	1	2	3	4	5	6	7	8	9	10
RC_j	01	02	04	08	10	20	40	80	1B	36



密钥编排算法

```
KeyExpansion(byte key[16], word w[44]){
    word tmp;
    for(i=0; i<4; ++i)
        w[i] = (key[4*i], key[4*i+1], key[4*i+2],
                 key[4*i+3]);
    for(i=4; i<44; ++i){
        tmp = w[i-1];
        if(i mod 4 ==0)
            tmp = SubWord(RotWord(tmp))^Rcon[i/4];
        w[i] = w[i-4]^tmp;
    }
}
```

AES 举例

Start of Round	After SubBytes	After ShiftRows	After MixColumns	Round Key
01 89 fe 76 23 ab dc 54 45 cd ba 32 67 ef 98 10				0f 47 0c af 15 d9 b7 7f 71 e8 ad 67 c9 59 d6 98
0e ce f2 d9 36 72 6b 2b 34 25 17 55 ae b6 4e 88	ab 8b 89 35 05 40 7f f1 18 3f f0 fc e4 4e 2f c4	ab 8b 89 35 40 7f f1 05 f0 fc 18 3f c4 e4 4e 2f	b9 94 57 75 e4 8e 16 51 47 20 9a 3f c5 d6 f5 3b	dc 9b 97 38 90 49 fe 81 37 df 72 15 b0 e9 3f a7
65 0f c0 4d 74 c7 e8 d0 70 ff e8 2a 75 3f ca 9c	4d 76 ba e3 92 c6 9b 70 51 16 9b e5 9d 75 74 de	4d 76 ba e3 c6 9b 70 92 9b e5 51 16 de 9d 75 74	8e 22 db 12 b2 f2 dc 92 df 80 f7 c1 2d c5 1e 52	d2 49 de e6 c9 80 7e ff 6b b4 c6 d3 b7 5e 61 c6
5c 6b 05 f4 7b 72 a2 6d b4 34 31 12 9a 9b 7f 94	4a 7f 6b bf 21 40 3a 3c 8d 18 c7 c9 b8 14 d2 22	4a 7f 6b bf 40 3a 3c 21 c7 c9 8d 18 22 b8 14 d2	b1 c1 0b cc ba f3 8b 07 f9 1f 6a c3 1d 19 24 5c	c0 89 57 b1 af 2f 51 ae df 6b ad 7e 39 67 06 c0
71 48 5c 7d 15 dc da a9 26 74 c7 bd 24 7e 22 9c	a3 52 4a ff 59 86 57 d3 f7 92 c6 7a 36 f3 93 de	a3 52 4a ff 86 57 d3 59 c6 7a f7 92 de 36 f3 93	d4 11 fe 0f 3b 44 06 73 cb ab 62 37 19 b7 07 ec	2c a5 f2 43 5c 73 22 8c 65 0e a3 dd f1 96 90 50
f8 b4 0c 4c 67 37 24 ff ae a5 c1 ea e8 21 97 bc	41 8d fe 29 85 9a 36 16 e4 06 78 87 9b fd 88 65	41 8d fe 29 9a 36 16 85 78 87 e4 06 65 9b fd 88	2a 47 c4 48 83 e8 18 ba 84 18 27 23 eb 10 0a f3	58 fd 0f 4c 9d ee cc 40 36 38 9b 46 eb 7d ed bd

AES 举例

72 ba cb 04	40 f4 1f f2	40 f4 1f f2	7b 05 42 4a	71 8c 83 cf
1e 06 d4 fa	72 6f 48 2d	6f 48 2d 72	1e d0 20 40	c7 29 e5 a5
b2 20 bc 65	37 b7 65 4d	65 4d 37 b7	94 83 18 52	4c 74 ef a9
00 6d e7 4e	63 3c 94 2f	2f 63 3c 94	94 c4 43 fb	c2 bf 52 ef
0a 89 c1 85	67 a7 78 97	67 a7 78 97	ec 1a c0 80	37 bb 38 f7
d9 f9 c5 e5	35 99 a6 d9	99 a6 d9 35	0c 50 53 c7	14 3d d8 7d
d8 f7 f7 fb	61 68 68 0f	68 0f 61 68	3b d7 00 ef	93 e7 08 a1
56 7b 11 14	b1 21 82 fa	fa b1 21 82	b7 22 72 e0	48 f7 a5 4a
db a1 f8 77	b9 32 41 f5	b9 32 41 f5	b1 1a 44 17	48 f3 cb 3c
18 6d 8b ba	ad 3c 3d f4	3c 3d f4 ad	3d 2f ec b6	26 1b c3 be
a8 30 08 4e	c2 04 30 2f	30 2f c2 04	0a 6b 2f 42	45 a2 aa 0b
ff d5 d7 aa	16 03 0e ac	ac 16 03 0e	9f 68 f3 b1	20 d7 72 38
f9 e9 8f 2b	99 1e 73 f1	99 1e 73 f1	31 30 3a c2	fd 0e c5 f9
1b 34 2f 08	af 18 15 30	18 15 30 af	ac 71 8c c4	0d 16 d5 6b
4f c9 85 49	84 dd 97 3b	97 3b 84 dd	46 65 48 eb	42 e0 4a 41
bf bf 81 89	08 08 0c a7	a7 08 08 0c	6a 1c 31 62	cb 1c 6e 56
cc 3e ff 3b	4b b2 16 e2	4b b2 16 e2		b4 ba 7f 86
a1 67 59 af	32 85 cb 79	85 cb 79 32		8e 98 4d 26
04 85 02 aa	f2 97 77 ac	77 ac f2 97		f3 13 59 18
a1 00 5f 34	32 63 cf 18	18 32 63 cf		52 4e 20 76
ff 08 69 64				
0b 53 34 14				
84 bf ab 8f				
4a 7c 43 b9				

AES 的实现

- 可以在 8 位处理器上非常有效地实现：
 - 字节代替在字节级别上操作，只要求一个 256 字节的表。
 - 行移位是简单的移字节操作。
 - 轮密钥加是按位异或操作。
 - 列混淆变换要求在域 $GF(2^8)$ 上的乘法，所有的操作都是基于字节的，只要简单地查表即可。
- 可以在 32 位处理器上非常有效地实现：
 - 将操作定义在 32 位的字上。
 - 事先准备好 4 张 256 字的表。
 - 每一轮通过查表并加上 4 次异或即可计算每一列的值。
 - 需要 4KB 空间来存储这 4 张表。

AES 在 32 位处理器上的高效实现

3. 列混淆

1. 字节代替

$$b_{i,j} = S[a_{i,j}]$$

$$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix}$$

2. 行移位

$$\begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix} = \begin{bmatrix} b_{0,j} \\ b_{1,j-1} \\ b_{2,j-2} \\ b_{3,j-3} \end{bmatrix}$$

4. 轮密钥加

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

AES 在 32 位处理器上的高效实现

可以写成一个式子：

$$\begin{aligned}
 \begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} &= \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} S[a_{0,j}] \\ S[a_{1,j-1}] \\ S[a_{2,j-2}] \\ S[a_{3,j-3}] \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix} \\
 &= \left(\begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \cdot S[a_{0,j}] \right) \oplus \left(\begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \cdot S[a_{1,j-1}] \right) \\
 &\quad \oplus \left(\begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \cdot S[a_{2,j-2}] \right) \oplus \left(\begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \cdot S[a_{3,j-3}] \right) \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}
 \end{aligned}$$

AES 在 32 位处理器上的高效实现

- 定义 4 个 256 字的表 $T_i, i = 0, 1, 2, 3$ 。
- 每个表输入一个 $0 \sim 255$ 范围内的数 x ，输出一个字 $T_i[x]$ 。
- 四个表可以提前计算好，需要存储空间 4KB。

$$T_0[x] \triangleq \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \cdot S[x] \quad T_1[x] \triangleq \begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \cdot S[x]$$

$$T_2[x] \triangleq \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \cdot S[x] \quad T_3[x] \triangleq \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \cdot S[x]$$

AES 在 32 位处理器上的高效实现

- 对状态数组一列进行一轮运算可以简化为 4 次查表操作和 4 次异或运算：

$$\begin{bmatrix} s'_{0,j} \\ s'_{1,j} \\ s'_{2,j} \\ s'_{3,j} \end{bmatrix} = T_0[s_{0,j}] \oplus T_1[s_{1,j-1}] \oplus T_2[s_{2,j-2}] \oplus T_3[s_{3,j-3}] \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

- 这种紧凑、高效的实现方式是 Rijndael 能被选为 AES 的重要原因之一。

目录

1 伪随机生成器与流密码

2 伪随机置换与分组密码

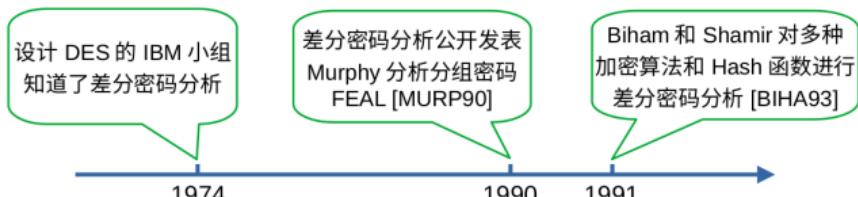
- 理想分组密码
- 扩散混淆范式及其实现
- 数据加密标准 DES
- 高级数据加密标准 AES
- 差分分析和线性分析

3 多重加密

4 工作模式

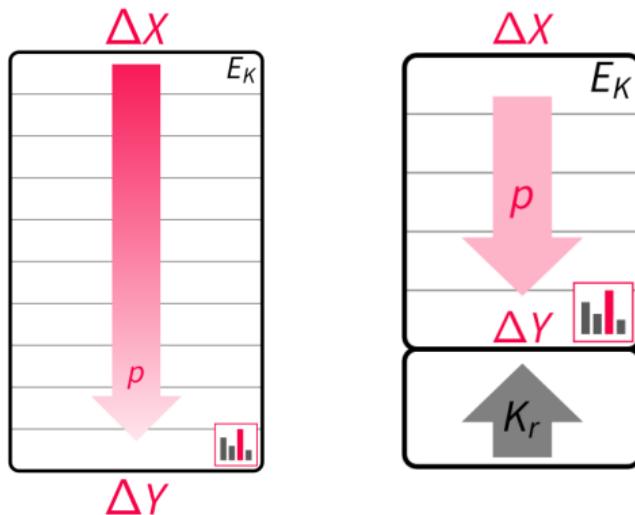
差分密码分析 (Differential Cryptanalysis)

- **差分密码分析**属于选择明文攻击。通过分析明文对的差分（即异或）对结果密文对的差分的影响，确定最有可能的密钥。
- 1990 年，Murphy、Biham 和 Shamir 首次提出用差分密码分析攻击分组密码和散列函数。
- 研究表明，若有 2^{47} 个选择明文，用差分分析就可以在 2^{47} 次加密运算内成功攻击 DES。但是要拥有 2^{47} 个选择明文的条件使得这种方法只具有理论上的意义。
- DES 在设计之初已经考虑了抵抗差分密码分析。在设计 S 盒和置换 P 时已经做了充分考虑。

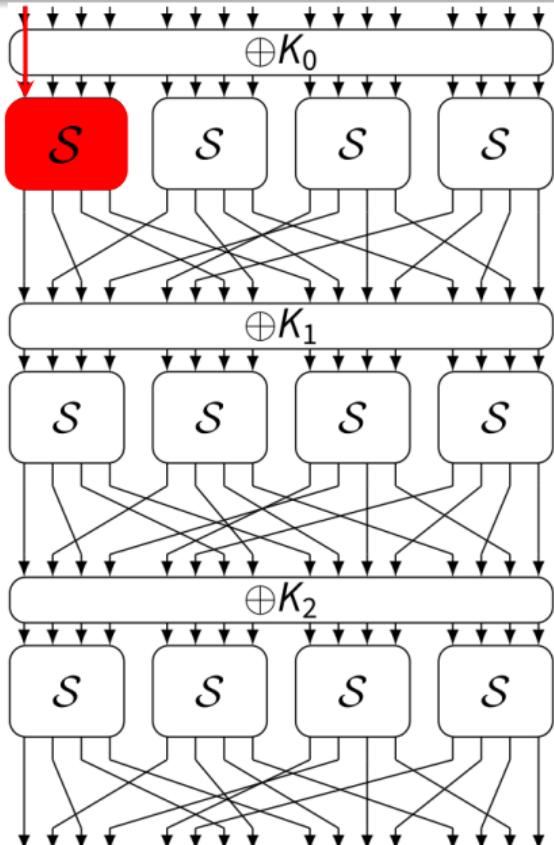


差分密码分析的主要思想

- 考虑输入差分为 ΔX 的一对明文；
- 跟踪这对明文经过每轮处理后的变化；
- 根据输出的差分推测每一轮处理所使用的子密钥。

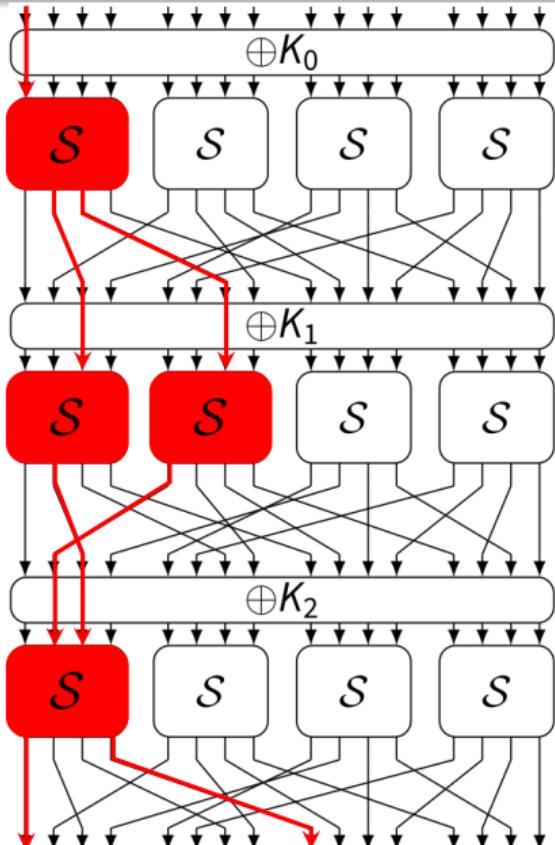


差分密码分析的主要思想



- 将 DES 简化为左图所示的运算，只包含 DES 的关键运算。
- 一次轮函数运算可以看作输入与该轮子密钥异或后再经过 S 盒替换。
- 输入异或不受子密钥影响**：考虑一对输入 x 和 x' ，则
$$(x \oplus k) \oplus (x' \oplus k) = x \oplus x'$$
- 当差分输入到一个 S 盒时，可以利用 S 盒的**差分特性**进行分析。

差分密码分析的主要思想

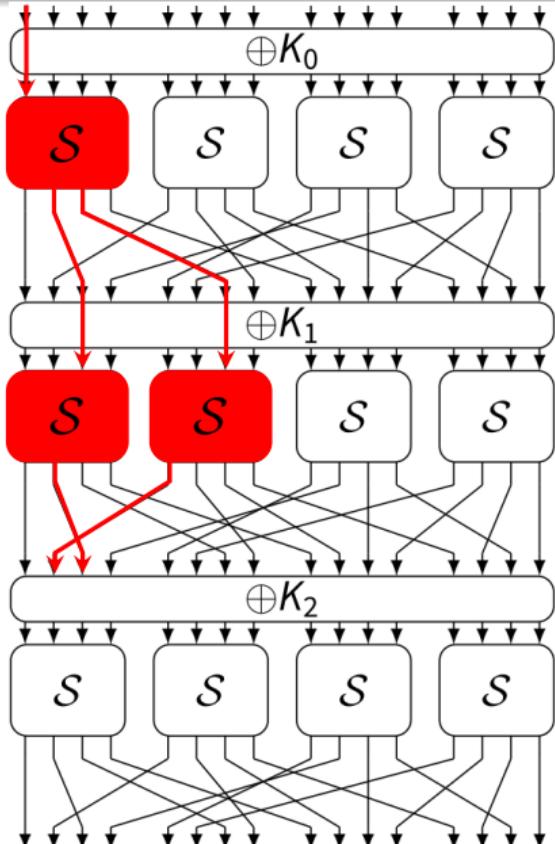


- 考虑具有相同输入差分的所有输入对: $\{(x, x') | x \oplus x' = \delta\}$
 - 统计 S 盒相应的输出对的差分, 会发现**输出差分的分布往往不均匀**。
 - 例如, 当输入差分为 1011 时, S_1 盒的输出差分分布为
- | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f | |
| 0 | 0 | 8 | 0 | 0 | 2 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 2 |
- 从而可以计算出从某个输入差分产生某个输出差分的概率。

S 盒输入输出差分统计特性

		Output Difference Δy															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Input Difference Δx	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	0	0	0	4	0	0	0	2	2	2	2	0	0	4	0
	2	0	0	0	0	0	2	0	2	0	2	2	4	2	2	0	0
	3	0	2	2	4	0	4	0	0	0	0	0	0	0	2	2	0
	4	0	0	0	2	2	2	6	0	2	0	0	0	0	2	0	0
	5	0	2	2	0	0	0	0	0	4	0	0	0	4	2	2	0
	6	0	2	0	2	0	0	0	0	0	2	0	2	0	4	0	4
	7	0	2	0	0	2	4	2	2	0	2	0	0	0	2	0	0
	8	0	0	0	2	0	0	0	2	0	0	0	0	2	2	2	4
	9	0	2	0	2	2	2	0	4	0	2	2	0	0	0	0	0
	A	0	0	4	0	2	0	2	4	2	0	2	0	0	0	0	0
	B	0	0	2	0	0	0	2	0	0	2	0	0	4	2	4	0
	C	0	0	0	0	0	0	0	0	4	4	0	4	0	0	0	4
	D	0	4	2	2	0	0	2	2	0	0	0	0	0	0	0	4
	E	0	2	4	2	4	0	0	0	0	0	0	0	0	2	0	2
	F	0	0	0	0	0	2	2	0	2	0	8	2	0	0	0	0

差分密码分析的攻击过程



- 搜集尽可能多的明密文对 $(x, y), (x', y')$ 且满足 $x \oplus x' = \delta$;
- 对于每对输入明文，计算出倒数第二轮的输出差分及其对应的传递概率；
- 枚举最后一轮的所有可能密钥，对于每对输出密文用密钥解密，得到最后一轮的输入对；
- 若该输入对的差分等于上一步计算出的输出差分，则该密钥计数加一；
- 根据每个密钥的计数值，输出计数值最大的密钥为该轮的子密钥。

线性密码分析

- 由 Matsui 于 1993 年提出的一种统计攻击方法，通过寻找 DES 变换的线性近似来进行攻击，属于已知明文攻击。
- 可以在有 2^{43} 个已知明文的情况下破译 DES 密钥，仍然只具有理论意义。
- 假设密钥为 n 比特，分组长度为 ℓ 比特，令 $I, O \subseteq \{1, \dots, \ell\}$, $K \subseteq \{1, \dots, n\}$ 。
- 对于输入输出分组 x 和 y ，分别用 x_I, y_O 表示由 I, O 指定的比特位置；用 k_K 表示由 K 指定的密钥 k 中的比特位置。
- 线性密码分析的目标是找到 I, O, K 的线性偏置 ϵ :

$$\left| \Pr[x_I \oplus y_O \oplus k_K = 0] - \frac{1}{2} \right| = \epsilon$$

- 如果存在这样的偏置，可由输入输出分组推测密钥。

目录

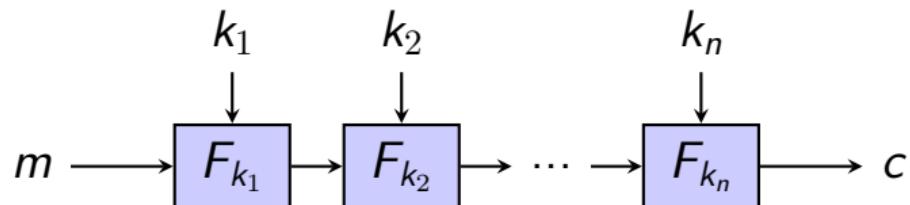
- 1 伪随机生成器与流密码
- 2 伪随机置换与分组密码
- 3 **多重加密**
 - 基本概念
 - 二重加密及 2DES
 - 三重加密及 3DES
- 4 工作模式

目录

- 1 伪随机生成器与流密码
- 2 伪随机置换与分组密码
- 3 **多重加密**
 - **基本概念**
 - 二重加密及 2DES
 - 三重加密及 3DES
- 4 工作模式

多重加密提出的背景

- DES 的密钥长度为 56 位, 已经不安全, 需要寻找更安全的加密方法。
- 例如, DES 密钥的穷举攻击目前仅需要 10 小时。
- 在高级加密标准 AES 出现之前, **多重加密**是一种增强加密算法安全性的解决方案。
- 即多次使用同一个加密算法, 对明文反复加密。
- 多重加密的**优点**: 可以利用现有软硬件资源。

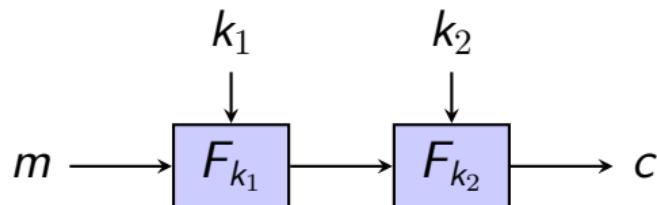


目录

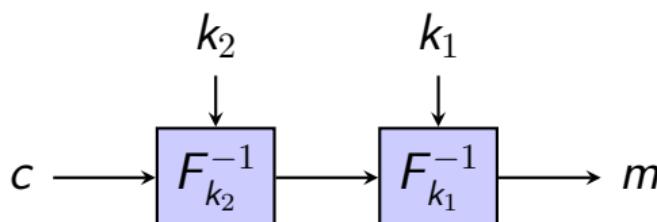
- 1 伪随机生成器与流密码
- 2 伪随机置换与分组密码
- 3 **多重加密**
 - 基本概念
 - **二重加密及 2DES**
 - 三重加密及 3DES
- 4 工作模式

双重加密与 2DES

- 多重加密最简单的形式是双重加密，使用两个密钥加密。
- 加密: $c = F_{k_2}(F_{k_1}(m))$

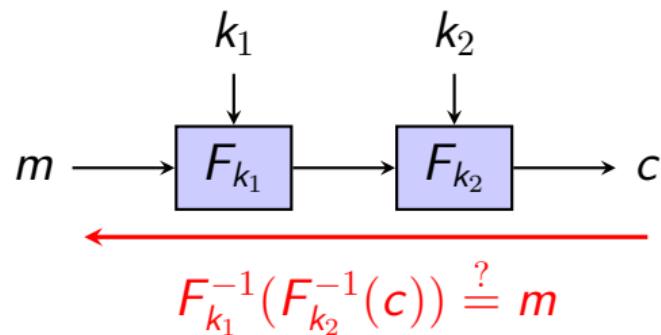


- 解密: $m = F_{k_1}^{-1}(F_{k_2}^{-1}(c))$



- 对于 DES，使用双重加密的 2DES 密钥长度为 112 位。

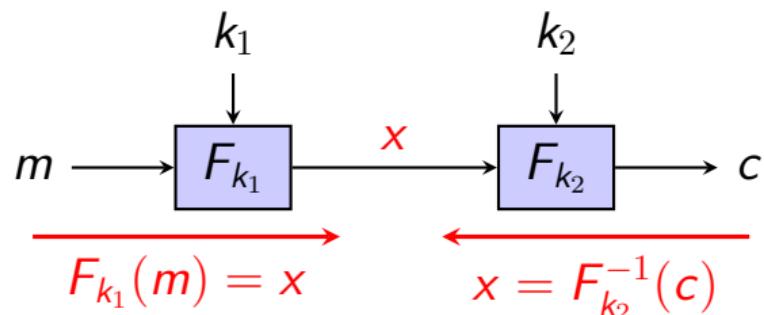
对双重加密的穷举攻击



- 给定密文 c ，依次尝试所有可能的密钥 k_2 和 k_1 ，直到发现明文 m 。
- 需要尝试 $2^{56} \times 2^{56} = 2^{112}$ 次。
- 有没有更有效的攻击手段？

中间相遇攻击 (Meet-in-the-Middle Attack)

- 中间相遇攻击对任何使用双重加密的分组密码都有效。

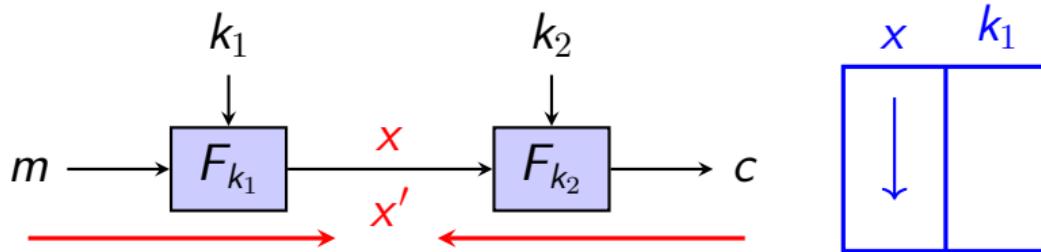


```
for k1 in K
  for k2 in K
    // ...
```

\Rightarrow

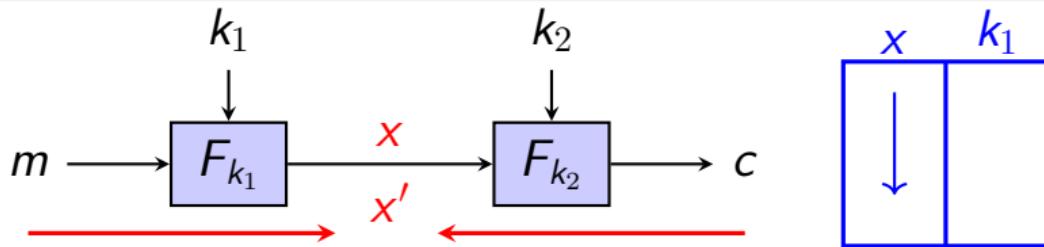
```
for k1 in K
  // ...
  for k2 in K
    // ...
```

中间相遇攻击的攻击步骤



- ① 搜集尽可能多的明密文对 (m, c) ;
- ② 遍历密钥 k_1 对 m 加密, 得到的结果按 x 排序后保存在表中;
- ③ 遍历密钥 k_2 对 c 解密, 每解一次密, 在表中匹配;
- ④ 如产生匹配, 则找到一对可能密钥, 然后用这两个密钥对一个新的明密文对进行验证, 若通过则说明密钥正确。

中间相遇攻击攻击 2DES 的复杂度分析



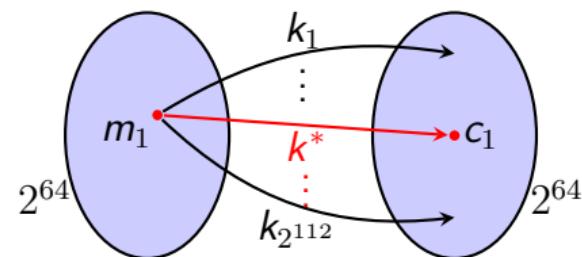
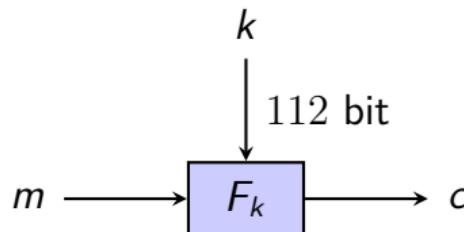
- 第 2 步和第 3 步的时间复杂度: $2^{56} + 2^{56} = 2^{57}$
- 空间复杂度: $(56 + 64) \times 2^{56}$ bit
- 第 4 步分析:
 - 使用一对 (m, c) 找到错误密钥数量: $2^{112} / 2^{64} = 2^{48}$
 - 使用两对 (m, c) 找到错误密钥数量: $2^{48} / 2^{64} = 2^{-16} \approx 0$

2DES 的安全性

❗相较于攻击 DES 的最差时间复杂度 2^{56} ，2DES 的加密强度并没有提高很多！

中间相遇攻击第 4 步需要试多少对 (m, c) ?

- 给定一个明密文对 (m_1, c_1) ，分组长度 64 位，2DES 密钥长度 112 位，所以会存在 2^{112} 种映射将 m_1 映射为密文。
- 由于密文空间大小仅仅为 2^{64} ，所以平均会有 $2^{112}/2^{64} = 2^{48}$ 种映射将 m_1 映射为 c_1 。
- 即每个明密文对平均存在 2^{48} 个映射，其中 $2^{48} - 1 \approx 2^{48}$ 个映射是错误的。
- 使用第二个明密文对 (m_2, c_2) 验证时，会从 2^{48} 个映射中找到正确的映射，找到错误映射的平均个数为 $2^{48}/2^{64} = 2^{-16} \approx 0$ 。



目录

1 伪随机生成器与流密码

2 伪随机置换与分组密码

3 多重加密

- 基本概念

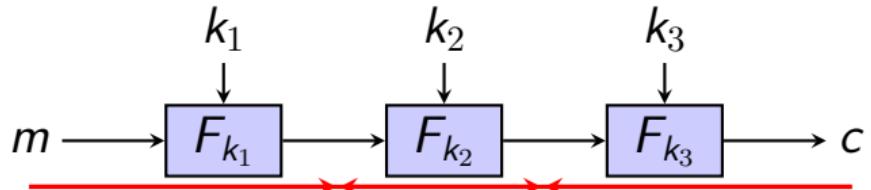
- 二重加密及 2DES

- 三重加密及 3DES**

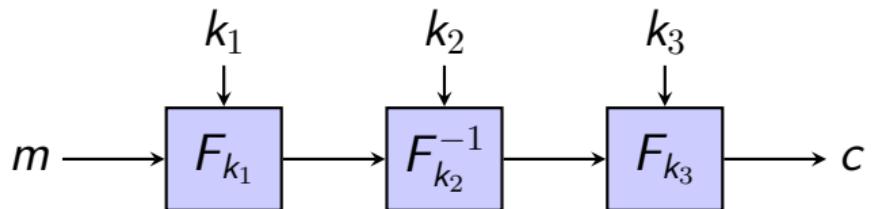
4 工作模式

三重加密与 3DES

- 为了进一步抵抗密码分析，可以使用三重加密：



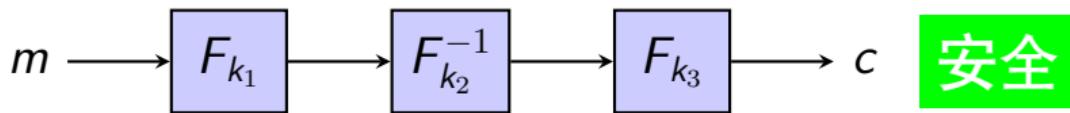
- 中间相遇攻击攻击的时间复杂度变为 $O(2^{112})$ 。
- 实际中会使用如下更灵活的三重加密方案 (RFC 1851)：



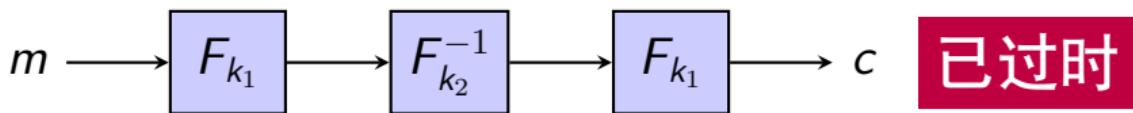
- $k_1 \neq k_2 \neq k_3$ 强三重加密，密钥长度 $3 \times 56 = 168$
- $k_1 = k_3 \neq k_2$ 普通三重加密，密钥长度 $2 \times 56 = 112$
- $k_1 = k_2 = k_3$ 等价于普通分组加密，密钥长度 56

3DES 现状

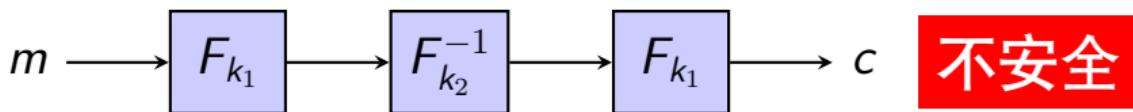
- 使用工作模式 1 的 3DES 目前仍然被广泛应用；



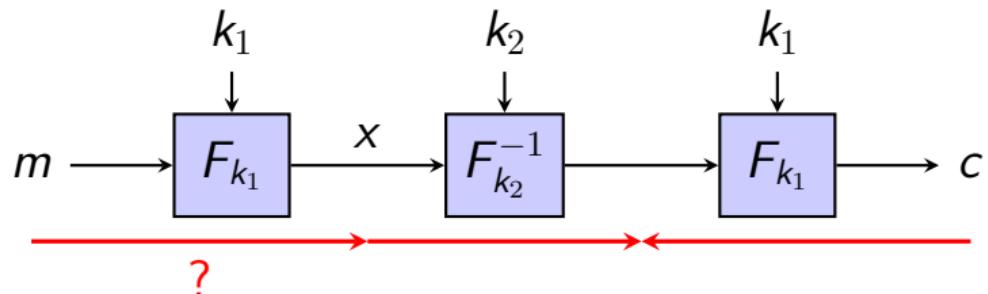
- 使用工作模式 2 的 3DES 于 2017 年被认为已过时；



- 使用工作模式 3 的 3DES 等价于普通 DES，不安全。

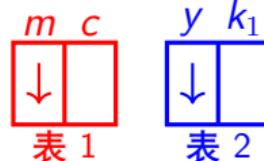
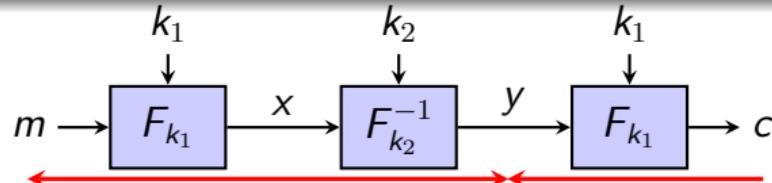


针对 3DES 的已知明文攻击



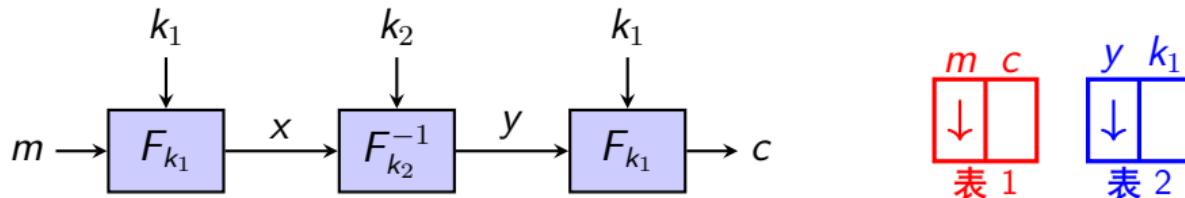
- 如果已知 x 和 c ，那么对三重加密的攻击可以转化为对二重加密的攻击；
- 当然，只要攻击者不知道密钥 k_1 ，即使知道 m 和 c ，还是无法知道 x ；
- 然而，攻击者可以选择 x 的一个可能值，再试着找到一个可产生 x 的 (m, c) 对，从而将对三重加密的攻击转化为对二重加密的攻击。

攻击步骤



- ① 获取尽量多个 (m, c) 对, 存入表 1;
- ② 随意选择值 x , 按以下步骤创建表 2:
 - 对每个可能密钥 k_1 计算可产生 x 的明文 $m = F_{k_1}^{-1}(x)$;
 - 在表 1 中匹配 m , 若匹配成功, 则在表 2 中添加一项 (y, k_1) , 其中 $y = F_{k_1}^{-1}(c)$ 。
- ③ 搜索 k_2 :
 - 对每个可能密钥 k_2 , 计算 $y = F_{k_2}^{-1}(x)$;
 - 在表 2 中匹配 y , 若匹配成功, 则找到一对密钥 (k_1, k_2) 可以产生已知 (m, c) 对。
- ④ 在其他 (m, c) 上验证密钥, 若成功则结束; 否则返回 2。

时间复杂度分析



- 对给定的 (m, c) 对, 选择 x 成功的可能性为 2^{-64} 。
- 给定 n 个 (m, c) 对, 则选择 x 成功的可能性为 $n2^{-64}$ 。
- 所以, 前两步平均需要尝试 $2^{64}/n$ 次才会得到一个正确的 x 。
- 对于每个 x , 第 3 步还需要遍历 k_2 , 因此总的时间复杂度为 $2^{56}2^{64}/n = 2^{120-\log_2 n}$ 。

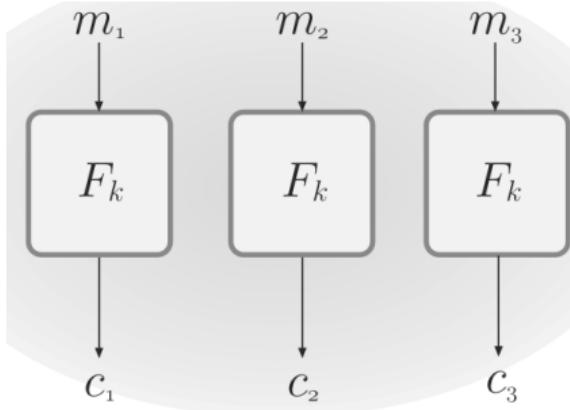
目录

- 1 伪随机生成器与流密码
- 2 伪随机置换与分组密码
- 3 多重加密
- 4 工作模式

分组密码的工作模式 (Block-Cipher Modes of Operation)

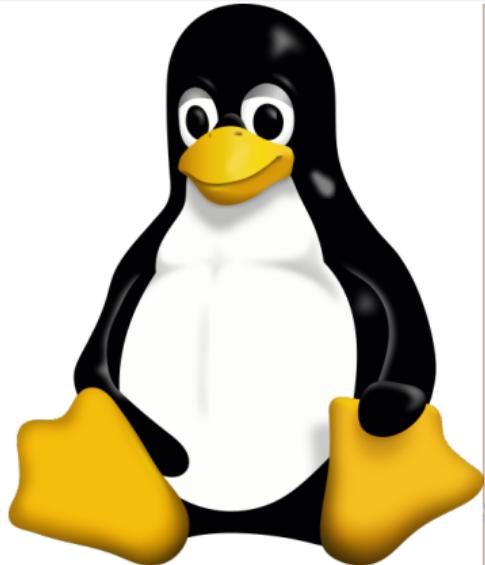
- 分组密码输入 ℓ 位明文分组，输出 ℓ 位密文分组。
- 若明文长度大于 ℓ ，则需要将明文分成 ℓ 位一组的块。
- 每次使用相同的密钥对多个分组加密，会引发安全问题。
- 为了将分组密码应用于各种各样的应用，NIST 定义了分组密码的工作模式。
- 本质上，工作模式是一项增强密码算法或者使算法适应具体应用的技术。
- 工作模式可使用包括 DES 和 AES 在内的任何分组密码。
- 如果最后一个分组不是完整的分组，则需要填充。
- eg., [b1 b2 b3 0 0 0 0 5], i.e., 3 data bytes, then 5 bytes pad+count.

电码本 (Electronic Codebook, ECB)



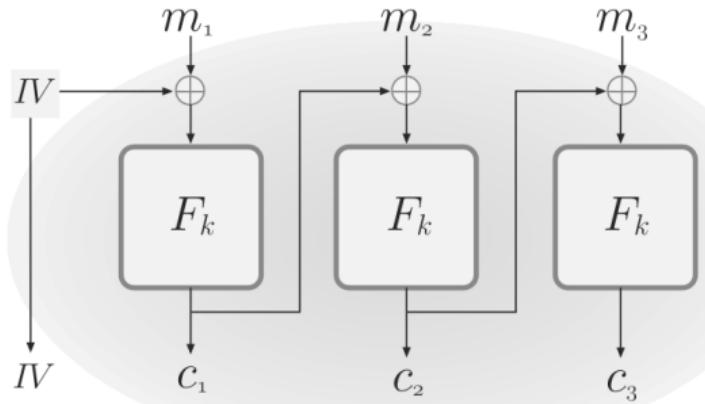
- 最简单的分组密码工作模式，类似于在密码本中查明文对应的密文。
- 解密以类似的方式进行。
- 相同的明文被加密成相同的密文，这是不安全的。

ECB 的局限性



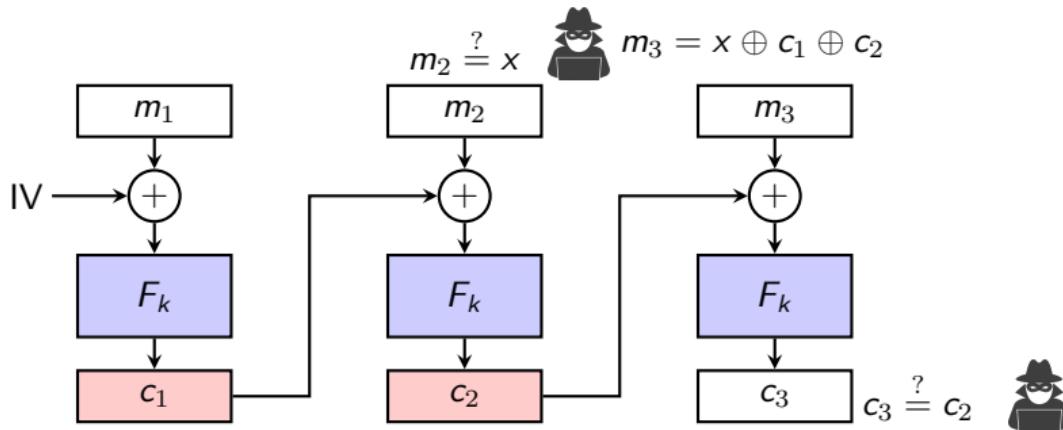
While the color of each individual pixel has supposedly been encrypted, the overall image may still be discerned, as the pattern of identically colored pixels in the original remains visible in the encrypted version.

密文分组链接 (Cipher Block Chaining, CBC)



- 加密算法的输入是当前明文分组与上一个密文分组的异或。
- 初始向量 (Initial Value, IV) 作为密文的一部分，每次随机选择。
- 密文分组链接每次加密是随机的，满足 CPA 安全。

CBC 的一个潜在漏洞

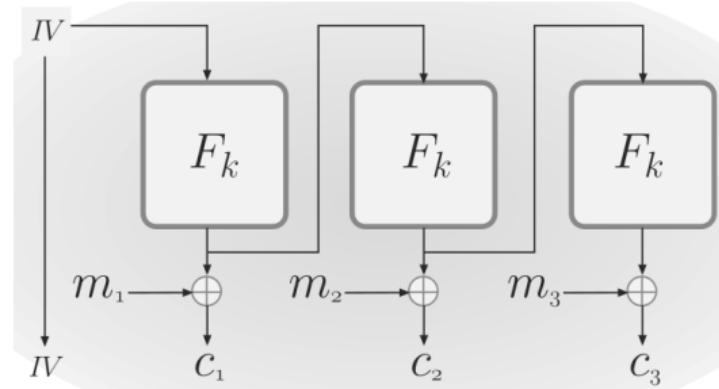


- 如果攻击者能观察连续两个密文分组 c_1 和 c_2 ，那么通过选择下一个明文分组 $m_3 = x \oplus c_1 \oplus c_2$ ，便可以判断第二个明文分组 m_2 是否等于 x 。
- 因为

$$c_3 = F_k(c_2 \oplus m_3) = F_k(c_2 \oplus x \oplus c_1 \oplus c_2) = F_k(x \oplus c_1)$$

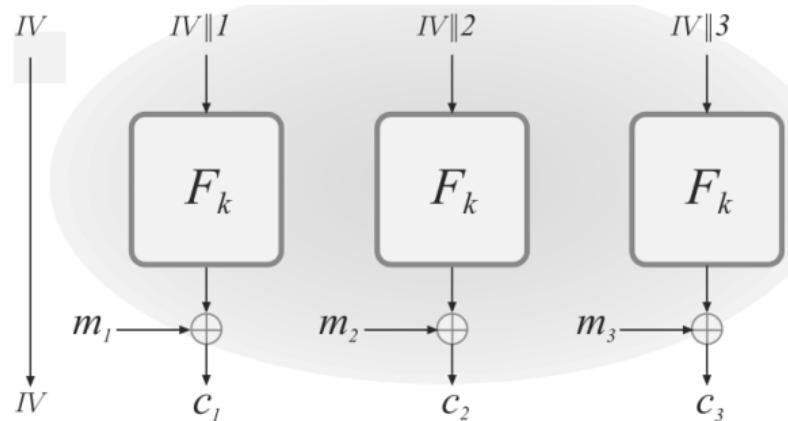
当 $m_2 = x$ 时，有 $c_3 = c_2$ 。

输出反馈 (Output Feedback, OFB)



- 可作为流密码使用（可以对 F_k 的输出进行截断，使其与消息长度等长）。
- 不需要 F_k 可逆，当 F 是伪随机函数时，OFB 满足 CPA 安全。
- 可事先生成伪随机流，加密操作可以并行执行，以加快计算。

计数器 (Counter, CTR)



- 可作为流密码，可高效并行化加解密，用于高速网络加密中。
- 当 F 是伪随机函数时，CTR 满足 CPA 安全。

小结

- 1 伪随机生成器与流密码
- 2 伪随机置换与分组密码
- 3 多重加密
- 4 工作模式