Something about Cryptography

1phan

What is cryptography?

- 什么是密码学？
- 需要了解什么？
- 常用的工具？

CONTENTS

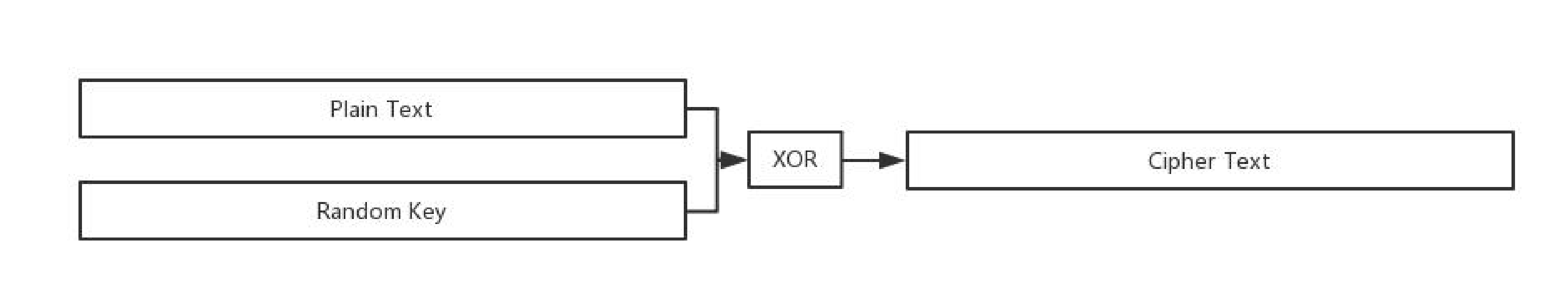
1

流加密

What is？

OTP

- OTP——完善保密性



OTP

- 为什么它是绝对安全的？
- 随机数 XOR 0 == ？
- 随机数 XOR 1 == ？

熵

- 不确定性的量度

- 如英语有26个字母，假如每个字母在文章中出现次数平均的话，每个字母的讯息量为？（4.7）

$$\mathrm{H}(X)=\mathrm{E}[\mathrm{I}(X)]=\mathrm{E}[-\ln(\mathrm{P}(X))]$$

$$\mathrm{H}(X)=\sum_i \mathrm{P}(x_i)\,\mathrm{I}(x_i)=-\sum_i \mathrm{P}(x_i)\log_b \mathrm{P}(x_i).$$

TEST

- 现有一段128bits长的随机数，一段128bits长的明文（这段明文包含了72bits的信息）
- 将他们 XOR 成一段128bits的密文

- 密文中包括了多少bits的信息?

- 信息不会凭空产生！
- 也不会凭空消失！

OTP VS Pseudorandom generator

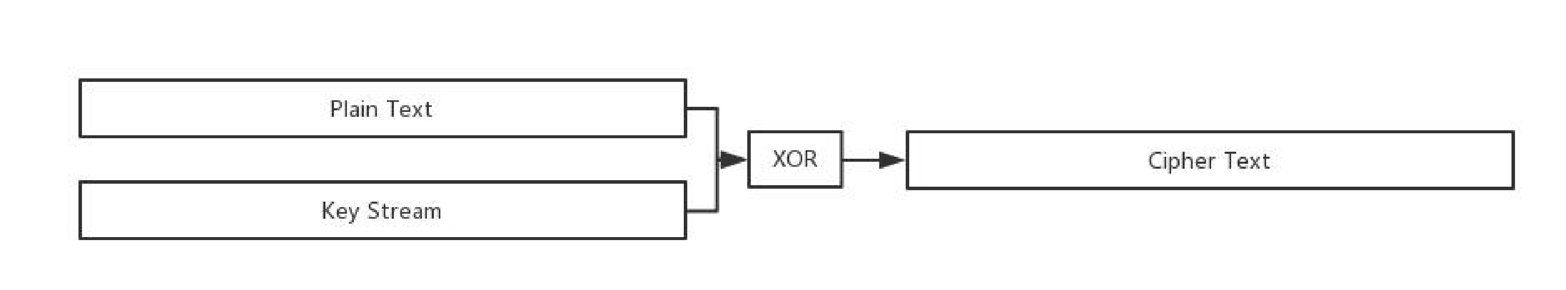
- 安全
- 密钥过长

- 密钥足够短
- 不是信息论安全的

我们需要一个安全性与可用性之间的平衡

流加密

- 一个伪随机数生成器
- 一个简单的运算（XOR）



伪随机数生成器

- glibc rand ？

```
seeding_stage() // (code omitted here, see the description from above link)

for (i=344; i<MAX; i++)
{
    r[i] = r[i-31] + r[i-3];
    val = ((unsigned int) r[i]) >> 1;
}
```

伪随机数生成器

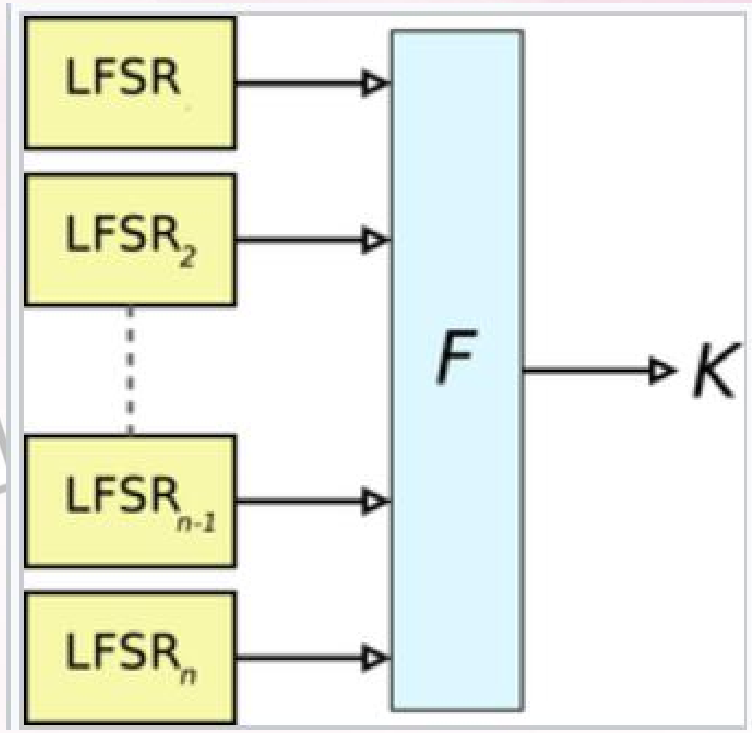
- 线性同余？

$$N_{j+1} \equiv (A \times N_j + B) \pmod{M}$$

统计学伪随机数生成器VS密码学安全伪随机数生成器

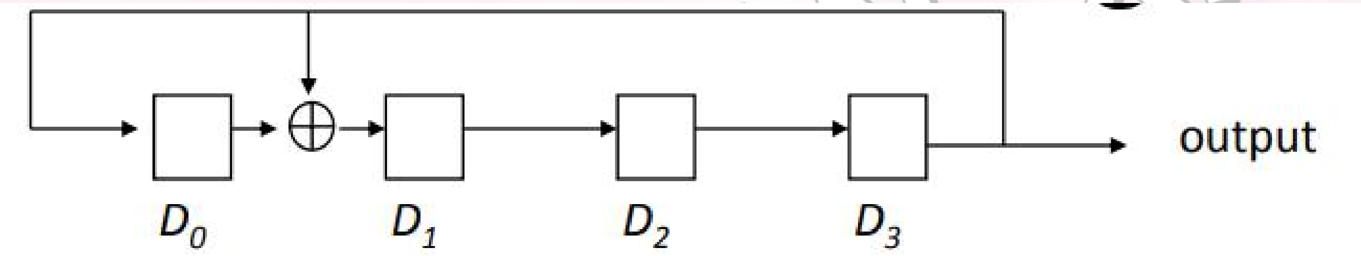
- 伪随机数生成器的内部状态可否轻易地由其输出演算得知

安全的随机数生成器

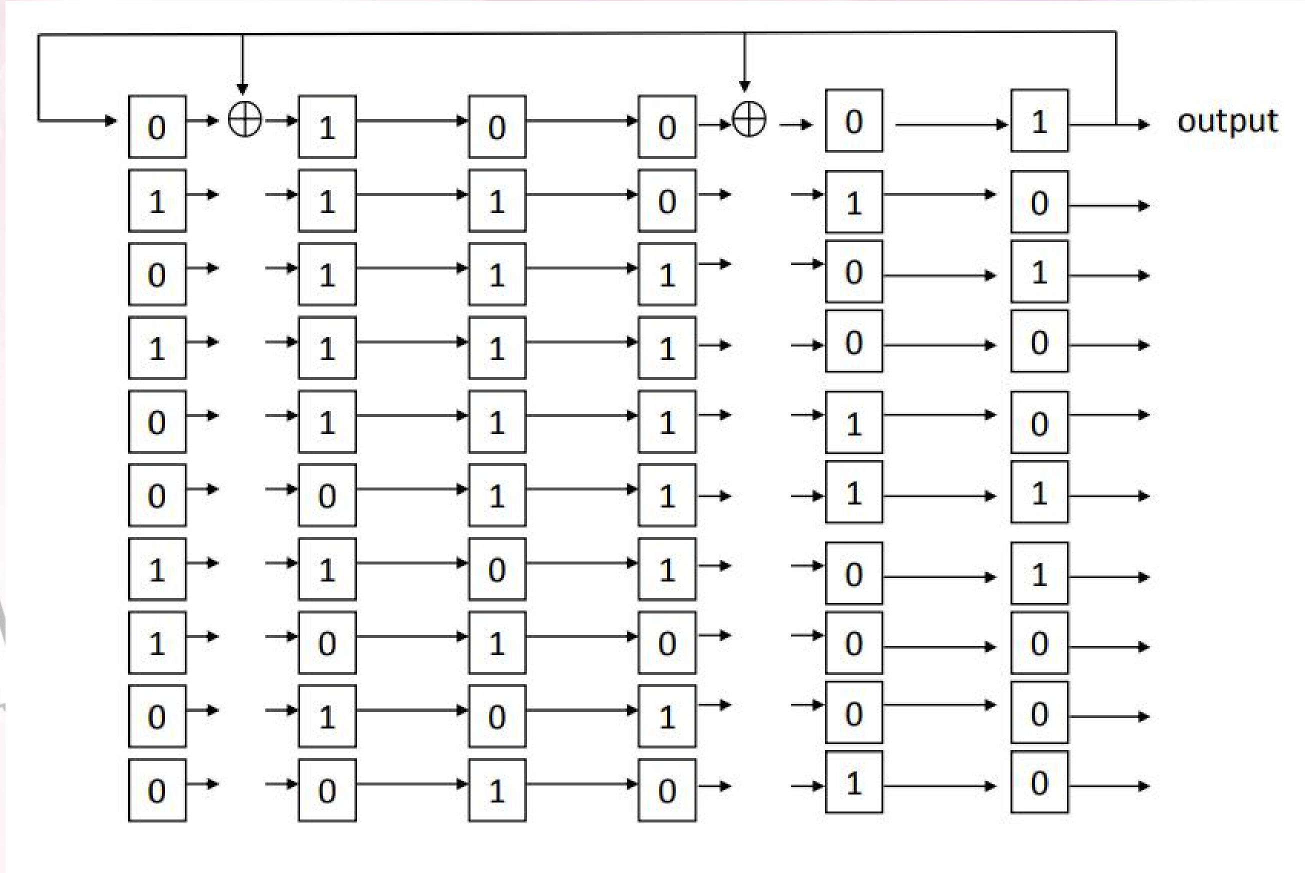


LFSR（线性反馈移位寄存器）

- 因为寄存器具有有限数量的可能状态，所以它最终必须进入重复循环。然而，具有良好选择的反馈功能的LFSR可以产生随机出现且具有非常长周期的比特序列。



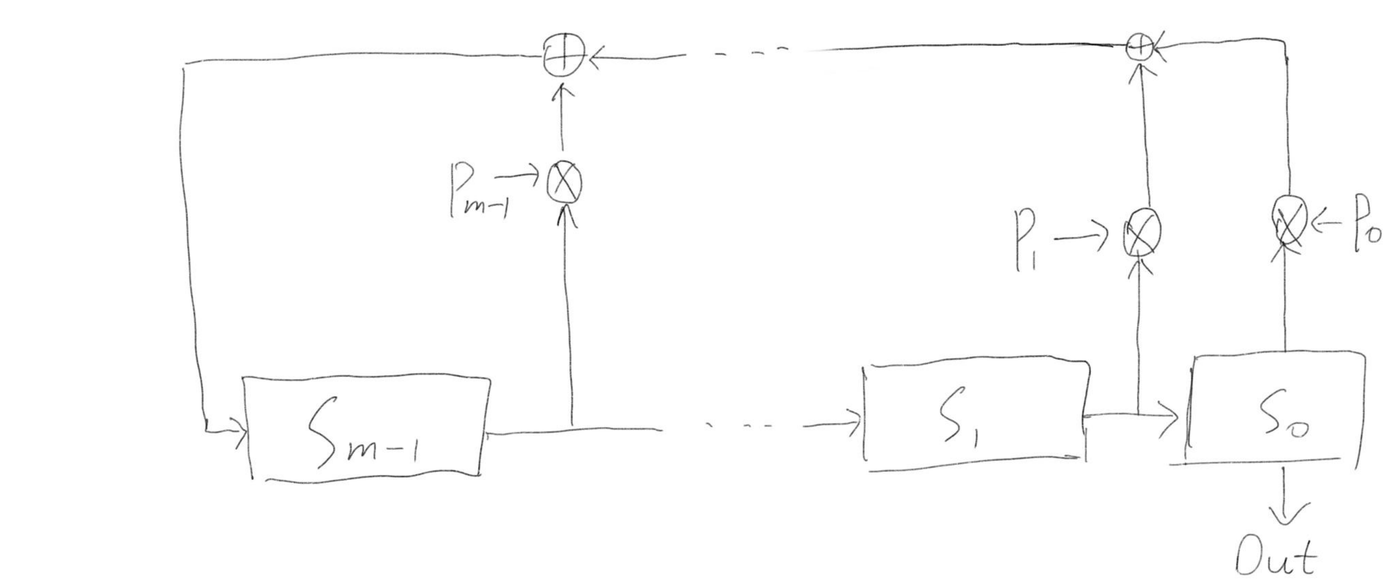
- seed：1111
- 1011-->1
- 1001-->1
- 1000-->1


output

LFSR的数学描述

- 得到最长LFSR <===> P(X)为本原多项式

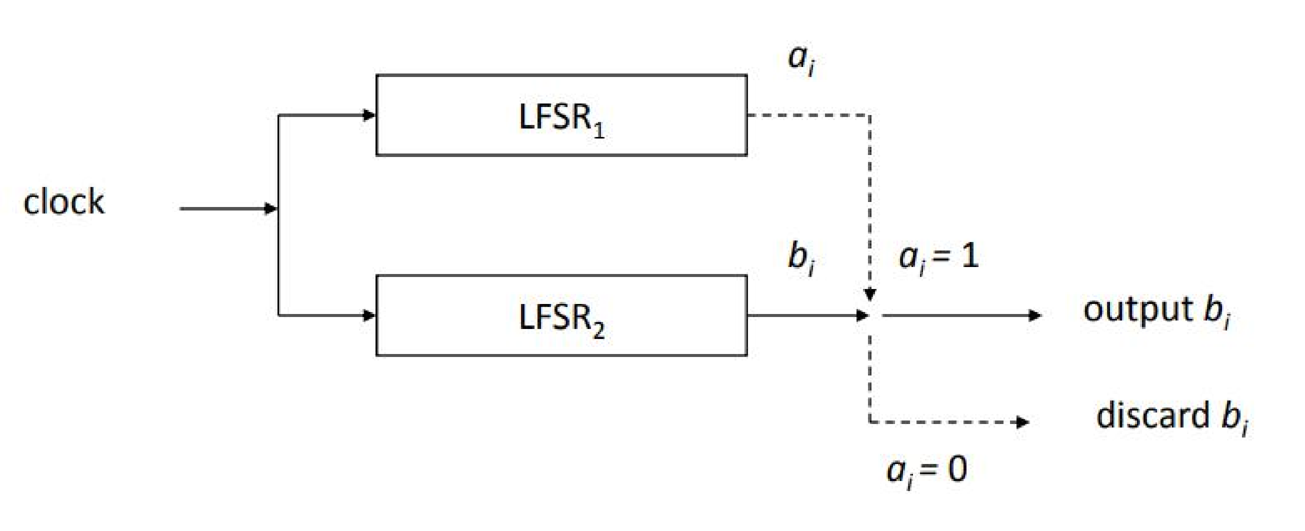
$$P(x) = x^m + p_{m-1}x^{m-1} + \ldots + p_1 x + p_0$$



LFSR不同长度的例子

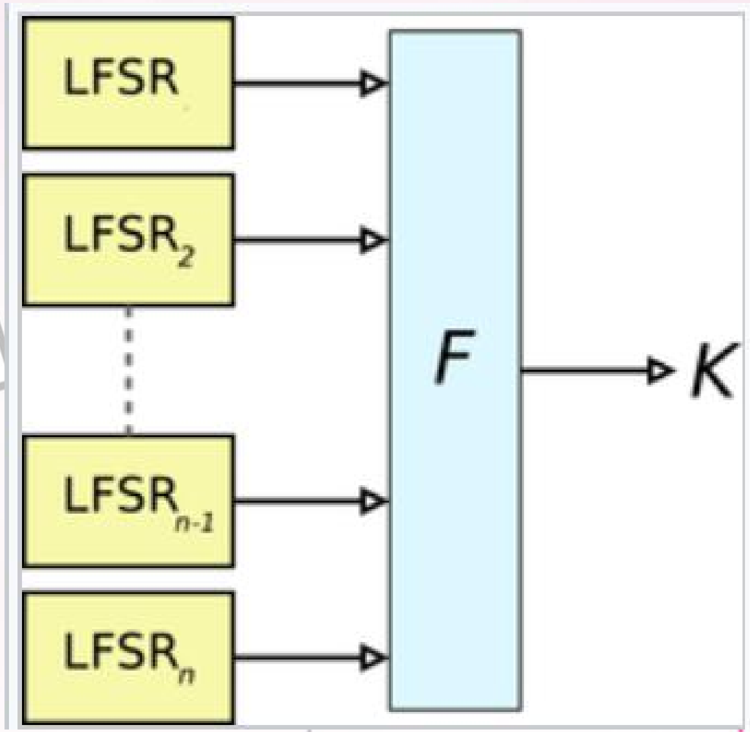
- $X^4 + X + 1$
- $X^4 + X^3 + X^2 + X + 1$

Shrinking Generator

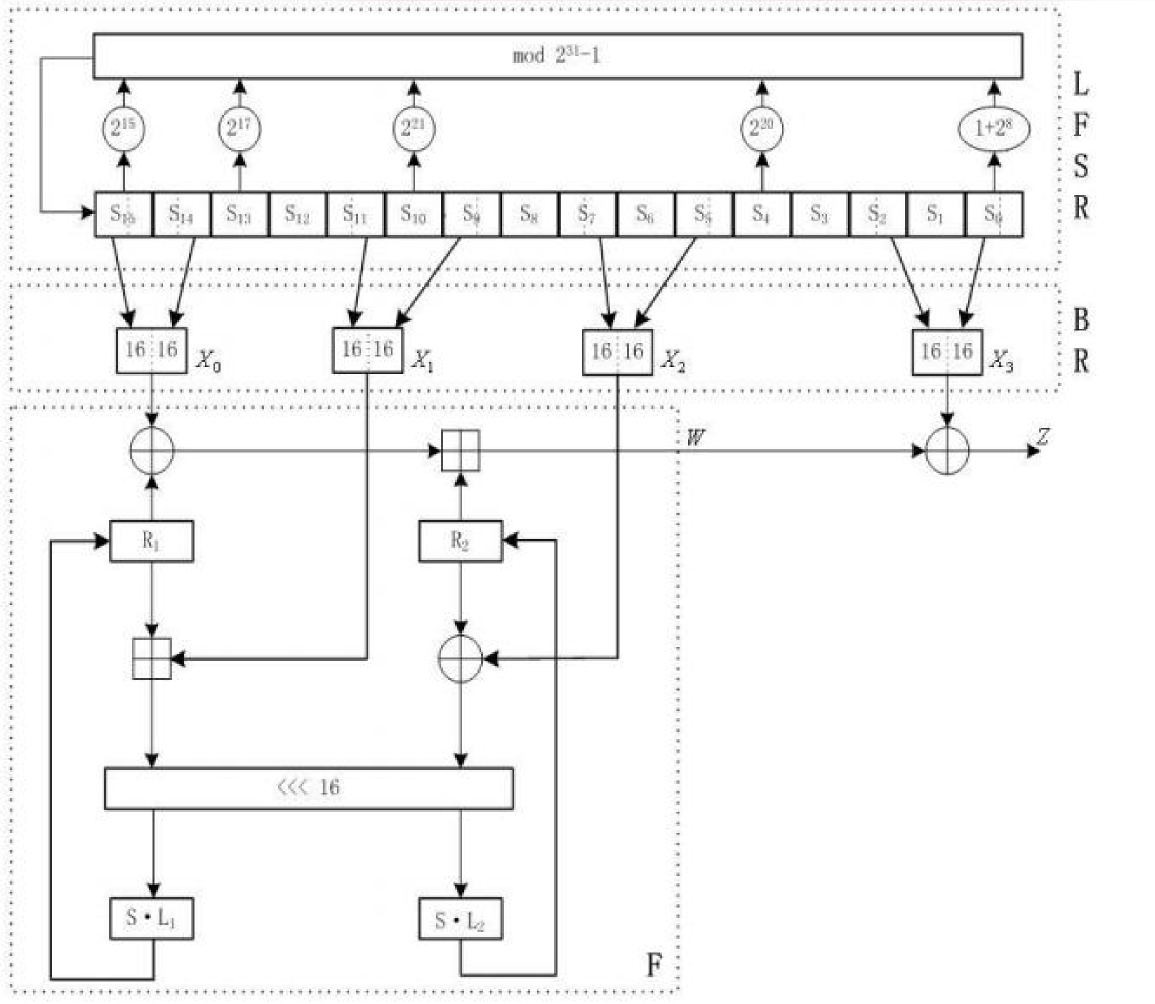


只用LFSR可以么？

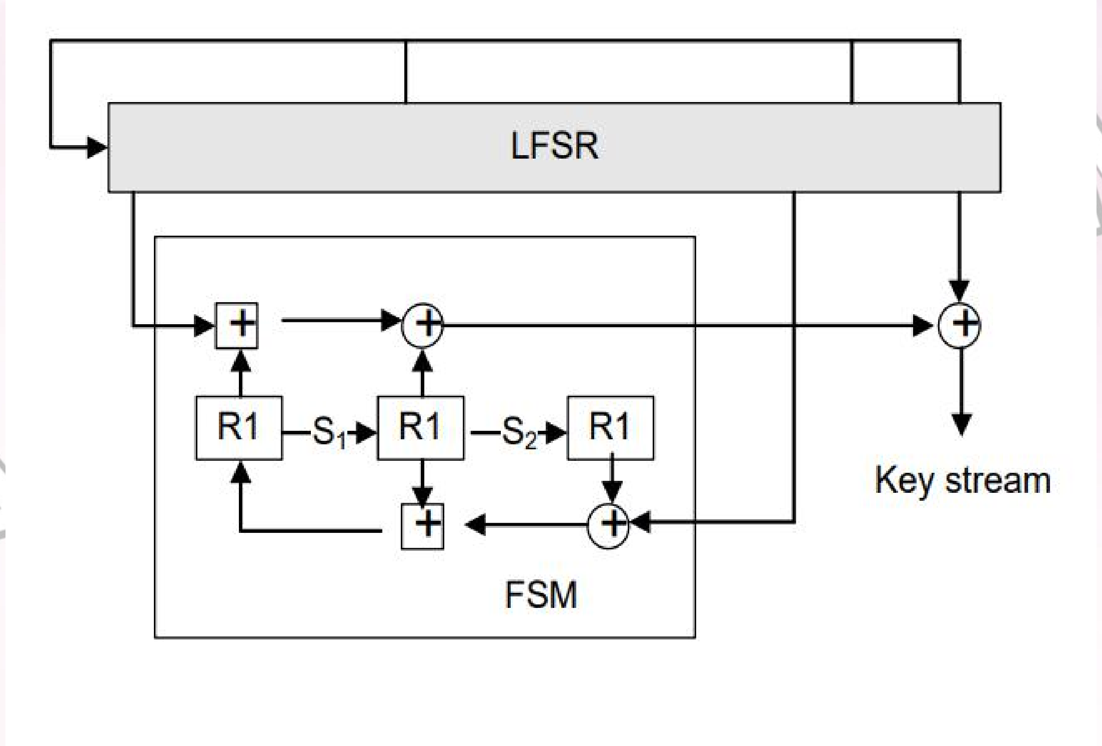
- NO!
- LFSR是可以根据输出反推回去的，我们需要一个非线性函数



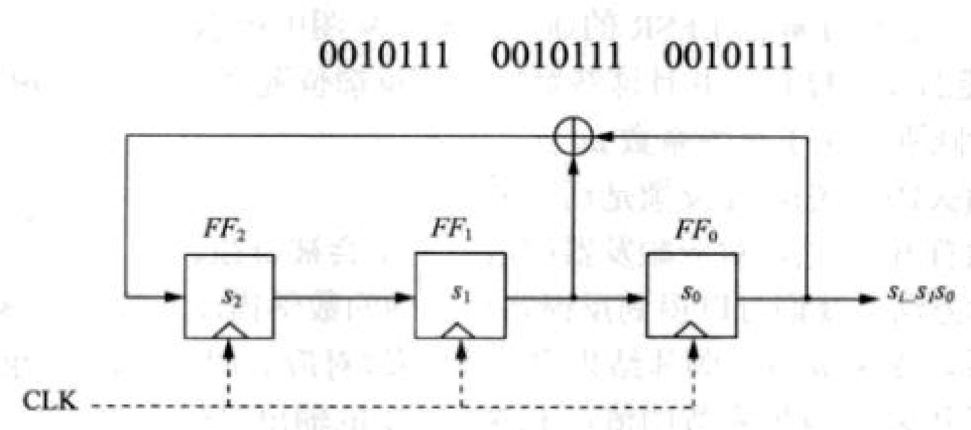
ZUC


mod $2^{31}-1$
2^{15}
2^{17}
2^{21}
2^{20}
$1+2^{8}$
L F S R
S_{15} S_{14} S_{13} S_{12} S_{11} S_{10} S_{9} S_{8} S_{7} S_{6} S_{5} S_{4} S_{3} S_{2} S_{1} S_{0}
B R
16 16 X_0
16 16 X_1
16 16 X_2
16 16 X_3
W
Z
R_1
R_2
<<< 16
$S \cdot L_1$
$S \cdot L_2$
F

SNOW3G



LFSR已知明文攻击



$$s_3 \equiv s_1 + s_0 \bmod 2$$
$$s_4 \equiv s_2 + s_1 \bmod 2$$
$$s_5 \equiv s_3 + s_2 \bmod 2$$
$$\vdots$$

$$s_{i+3} \equiv s_{i+1} + s_i \bmod 2$$

LFSR已知明文攻击

1010 0111 000010
0 1 2 3 4 5 6 7 8 9 10 11

0 0 0 1 0 1

1 1 0 0 0 0

$$\left.\begin{aligned}
&①\ 1 = P_5 + P_2 + P_0\\
&②\ 0 = P_5 + P_4 + P_1\\
&③\ 0 = P_4 + P_3 + P_0\\
&④\ 0 = P_3 + P_2\\
&⑤\ 1 = P_2 + P_1\\
&⑥\ 0 = P_5 + P_1 + P_0
\end{aligned}\right\} \to \bmod 2$$

$⑥+② = P_4 + P_0 = 0 =$

$⑤+②+③ = P_3 = 0$

$P_2 = 0\ ;\ P_1 = 1$

$$\begin{cases} P_5 + P_0 = 1 \\ P_5 + P_4 = 1 \\ P_4 + P_0 = 0 \end{cases} \qquad \begin{aligned} &P_5 = 1 \\ &P_0 = P_4 = 0 \end{aligned}$$

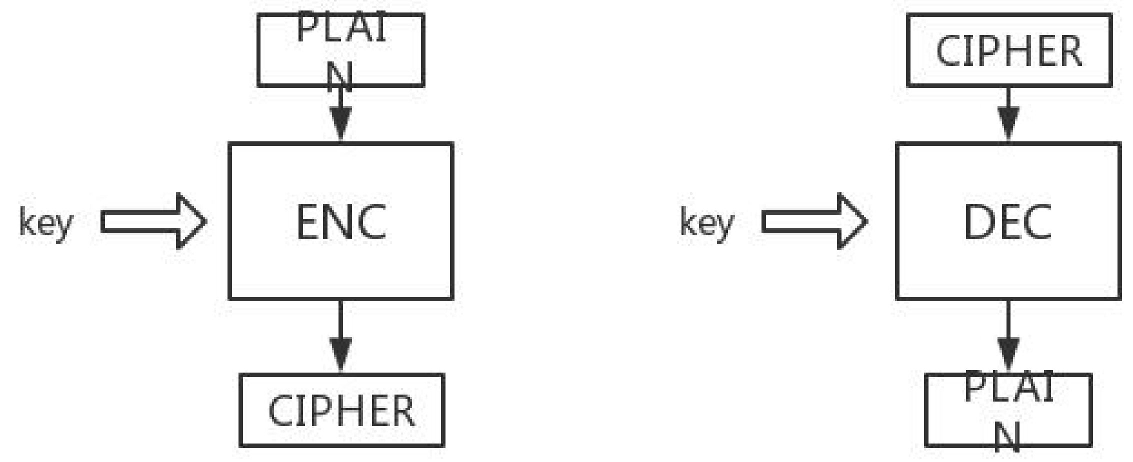
$X_m = X_{m-1} + X_{m-5}$

2

块加密

What is？

块加密



块加密

加密：

$$E_K(P) := E(K,P) : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$$

解密：

$$E_K^{-1}(C) := D_K(C) = D(K,C) : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$$

混淆与扩散

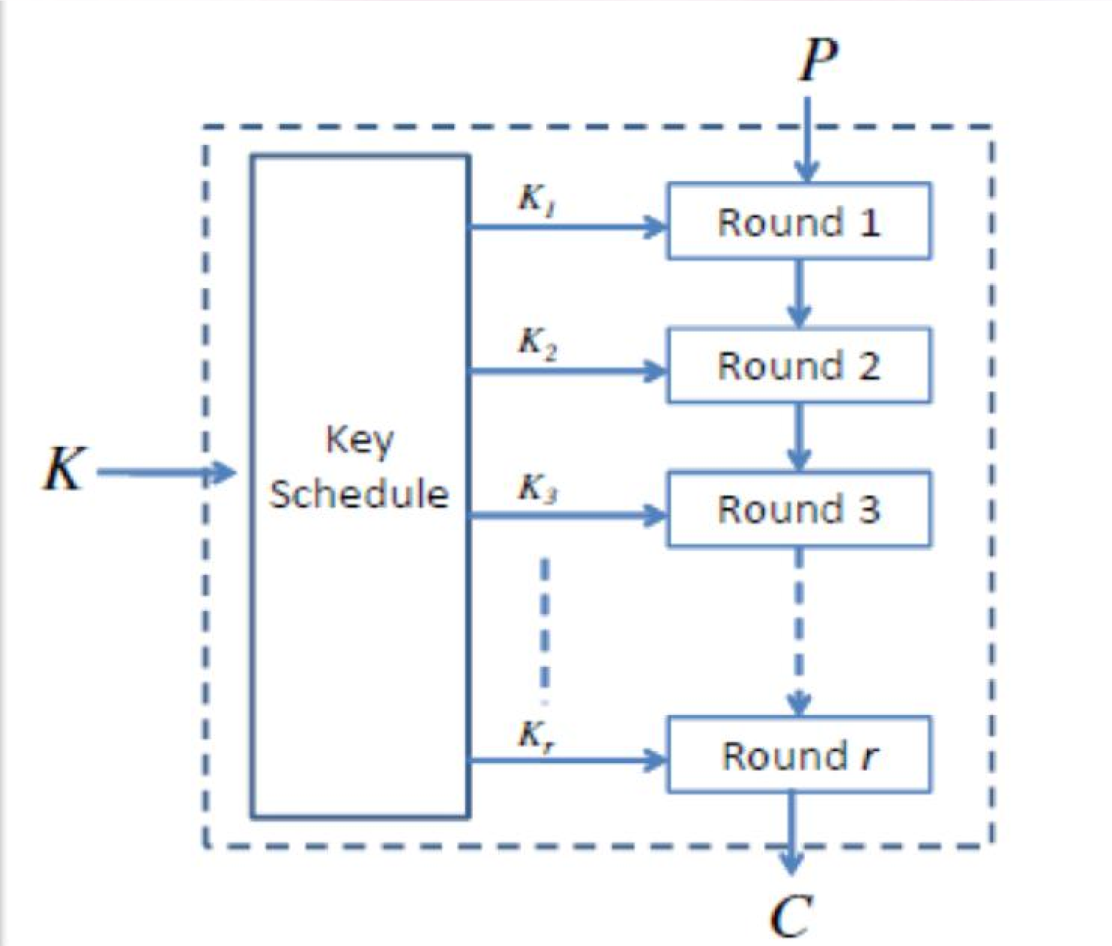
- 混淆，Confusion，将密文与密钥之间的统计关系变得尽可能复杂，使得攻击者即使获取了密文的一些统计特性，也无法推测密钥。
- 扩散，Diffusion，使得明文中的每一位影响密文中的许多位。

块加密

- 大部分分组密码都是迭代分组密码
- what is？

$$
\begin{aligned}
&M_0 = M \oplus K_0 \\
&M_i = R_{K_i}(M_{i-1})\ ;\ i = 1 \ldots r \\
&C = M_r \oplus K_{r+1}
\end{aligned}
$$

迭代分组密码

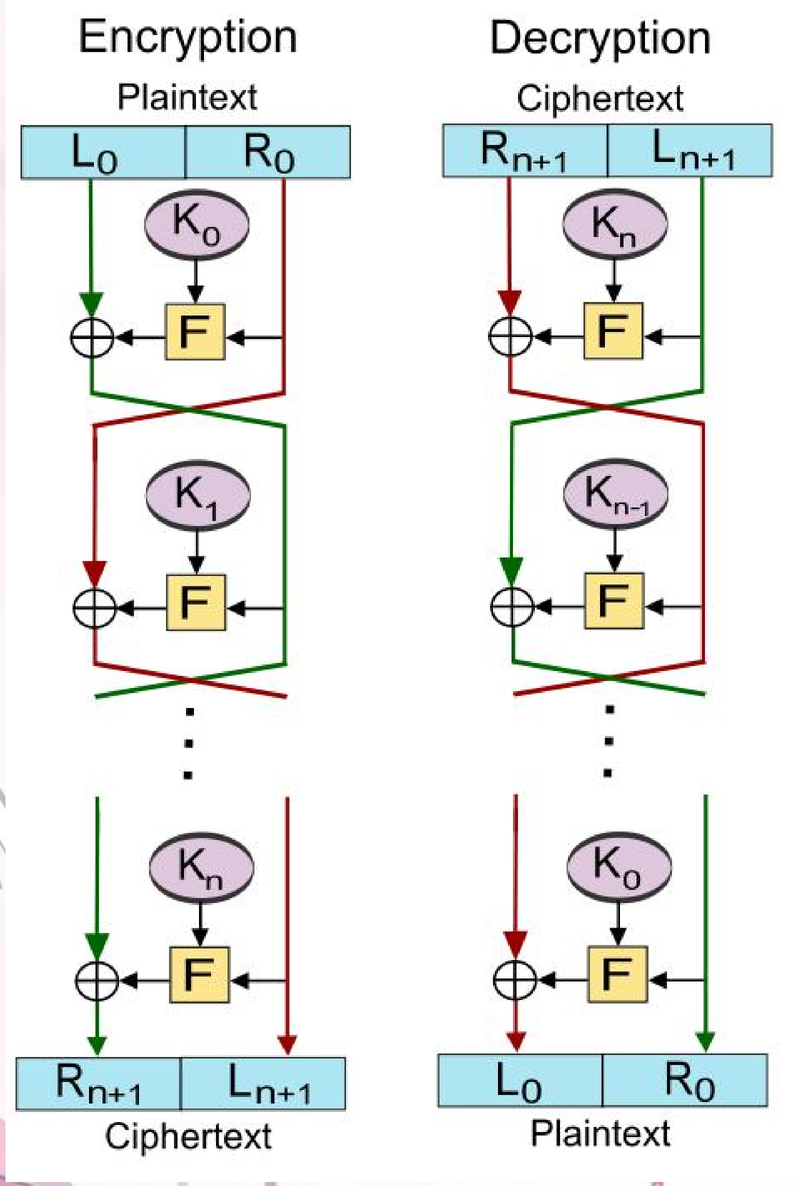


轮函数构造

- Feistel Network
- Substitution-Permutation Network

Feistel Cipher

- 在密码学中，Feistel密码是用于构造分组密码的对称结构，它以在为IBM（美国）工作时做了开创性的研究的德国物理学家和密码学家Horst Feistel的名字命名; 它通常也被称为Feistel网络。 大部分分组密码使用该方案，包括数据加密标准（DES）。
- Feistel结构的优点是加密和解密操作非常相似，在某些情况下甚至相同，只需要反转子密钥的顺序。 因此，实现这种密码所需的代码或电路的大小几乎减半。



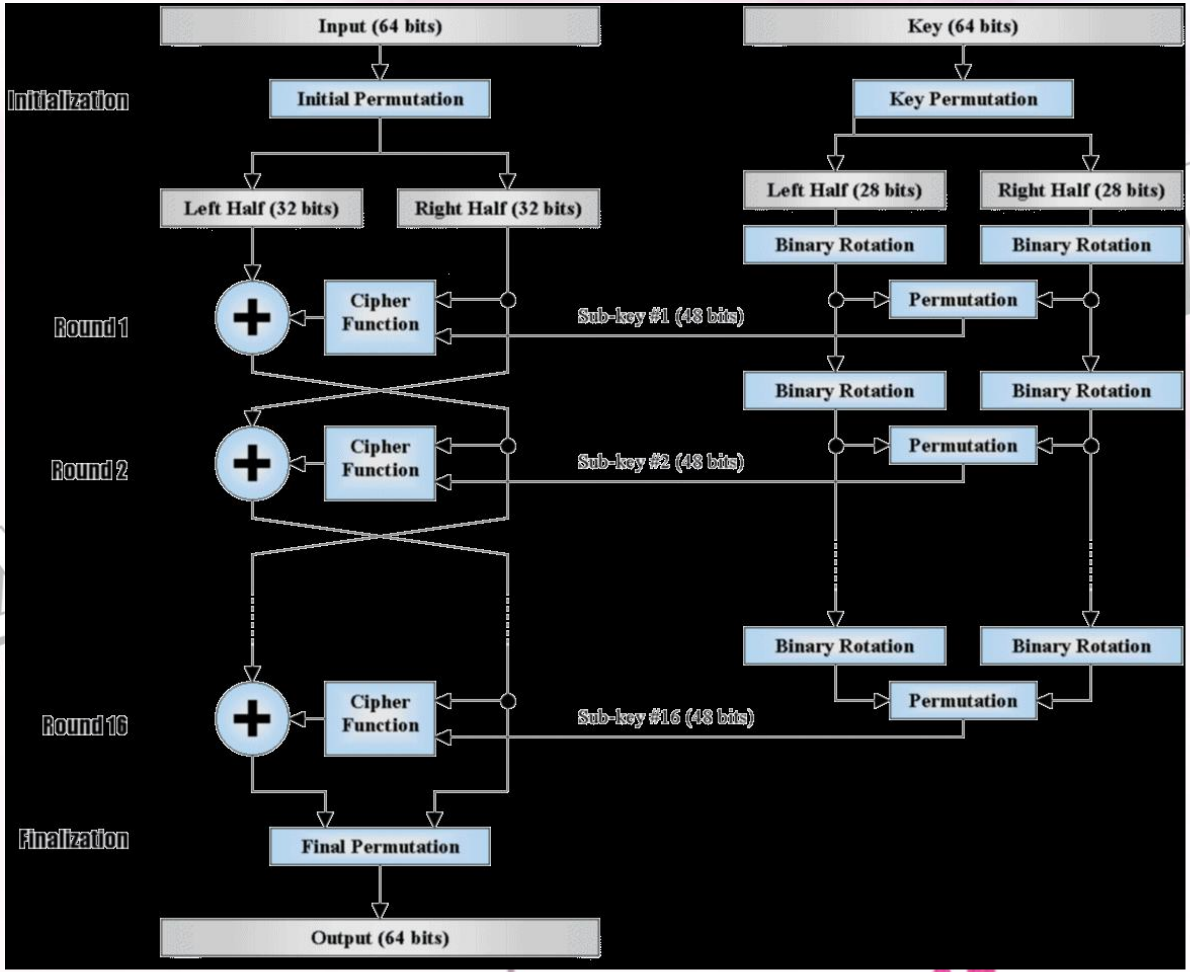
$i = 0, 1, \ldots, n$

$$
\begin{aligned}
L_{i+1} &= R_i \\
R_{i+1} &= L_i \oplus \mathrm{F}(R_i, K_i)
\end{aligned}
$$

$i = n, n-1, \ldots, 0$

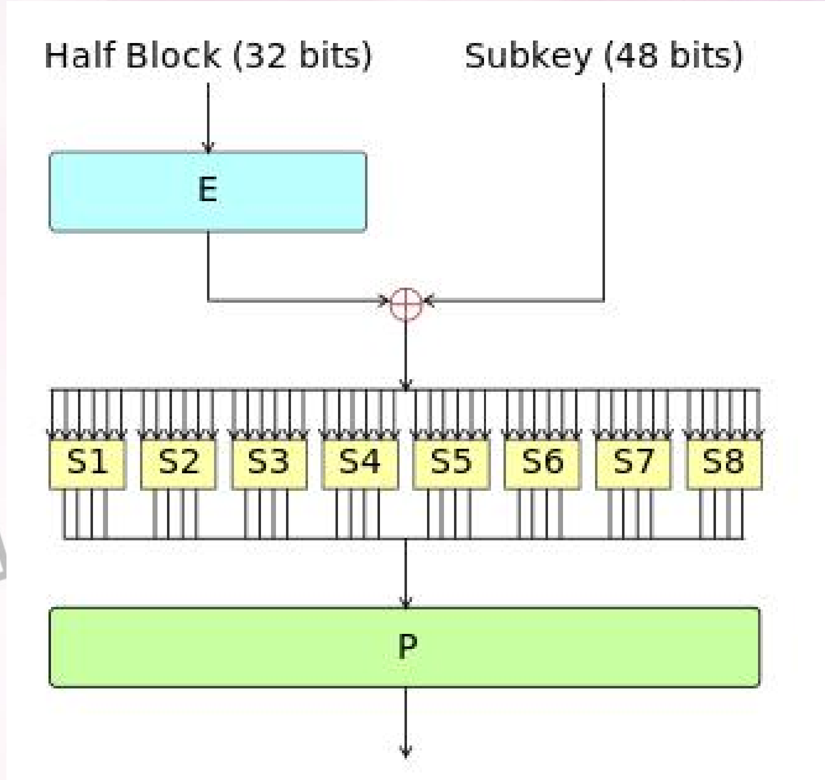
$$
\begin{aligned}
R_i &= L_{i+1} \\
L_i &= R_{i+1} \oplus \mathrm{F}(L_{i+1}, K_i).
\end{aligned}
$$

DES



Input (64 bits)
Key (64 bits)
Initialization
Initial Permutation
Key Permutation
Left Half (32 bits)
Right Half (32 bits)
Left Half (28 bits)
Right Half (28 bits)
Binary Rotation
Binary Rotation
Round 1
Cipher Function
Permutation
Sub-key #1 (48 bits)
Binary Rotation
Binary Rotation
Round 2
Cipher Function
Permutation
Sub-key #2 (48 bits)
Binary Rotation
Binary Rotation
Round 16
Cipher Function
Permutation
Sub-key #16 (48 bits)
Finalization
Final Permutation
Output (64 bits)

F函数



IP

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

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

Initial Key Permutation

Initial Key Permutation

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

Key Shifting

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

Subkey Permutation

Subkey Permutation

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

48 bits key V.S. 32 bits block?

- Cipher expansion!

Input		1	2	3	4			5	6	7	8			...			29	30	31	32	
		↓	↓	↓	↓	↘	↙	↓	↓	↓	↓	↘	↙		↘	↙	↓	↓	↓	↓	
Output	32	1	2	3	4	5	4	5	6	7	8	9	8	...	29	28	29	30	31	32	1

轮函数&弱密钥

- 针对子密钥生成器
- DES的4个弱密钥 0&F

Feistel Challenge

- 给Feistel网络写解密函数

3DES

$$C = E_{k3}(D_{k2}(E_{k1}(P)))$$

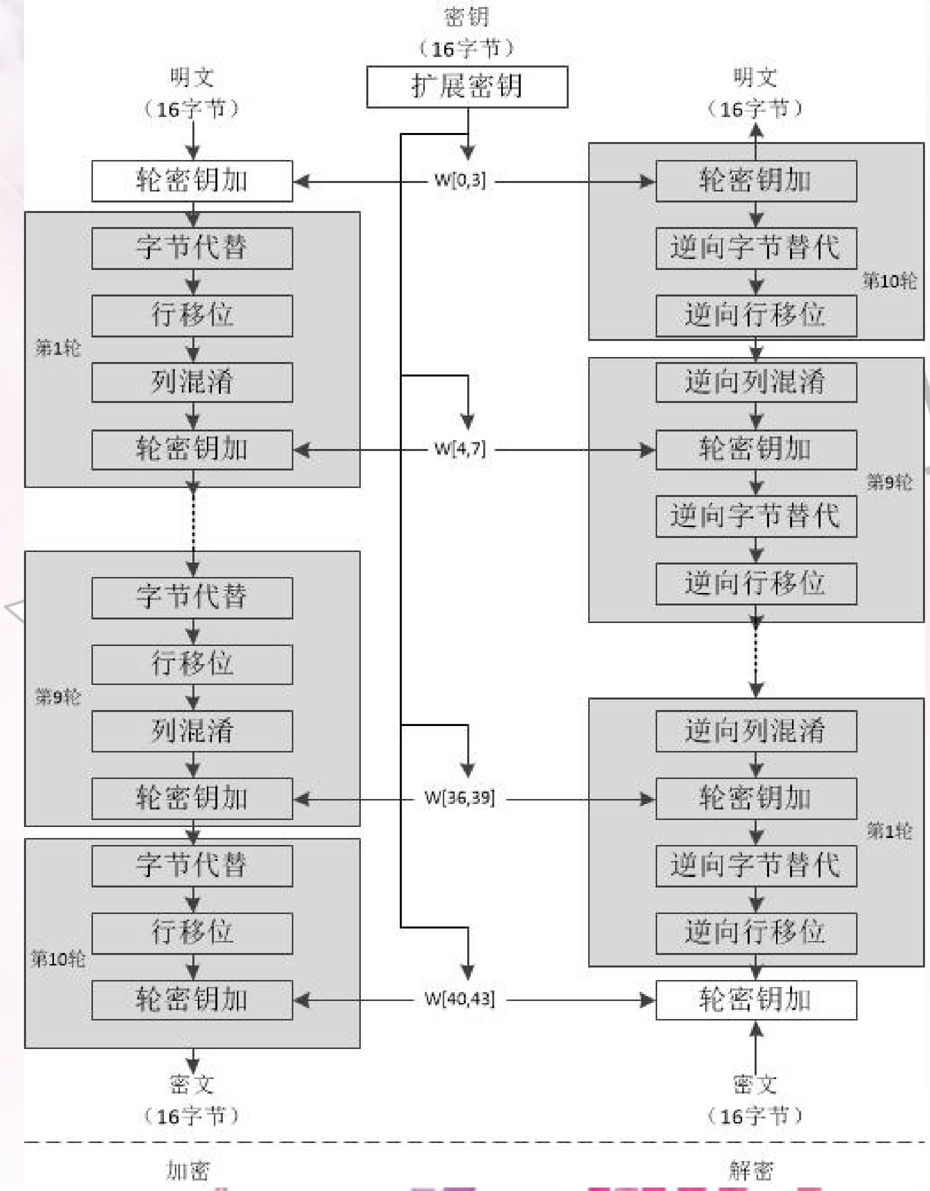
$$P = D_{k1}(E_{k2}(D_{k3}(C)))$$

生日悖论<--->2DES

- 中间相遇攻击
- P --k1--> M --k2--> C

$$C = E_{k2}(E_{k1}(P))$$

AES


密钥
（16字节）
扩展密钥
明文
（16字节）
轮密钥加
字节代替
行移位
列混淆
轮密钥加
第1轮
W[0,3]
W[4,7]
W[36,39]
W[40,43]
字节代替
行移位
列混淆
轮密钥加
第9轮
字节代替
行移位
轮密钥加
第10轮
密文
（16字节）
明文
（16字节）
轮密钥加
逆向字节替代
逆向行移位
第10轮
逆向列混淆
轮密钥加
逆向字节替代
逆向行移位
第9轮
逆向列混淆
轮密钥加
逆向字节替代
逆向行移位
第1轮
轮密钥加
密文
（16字节）
加密
解密

AES

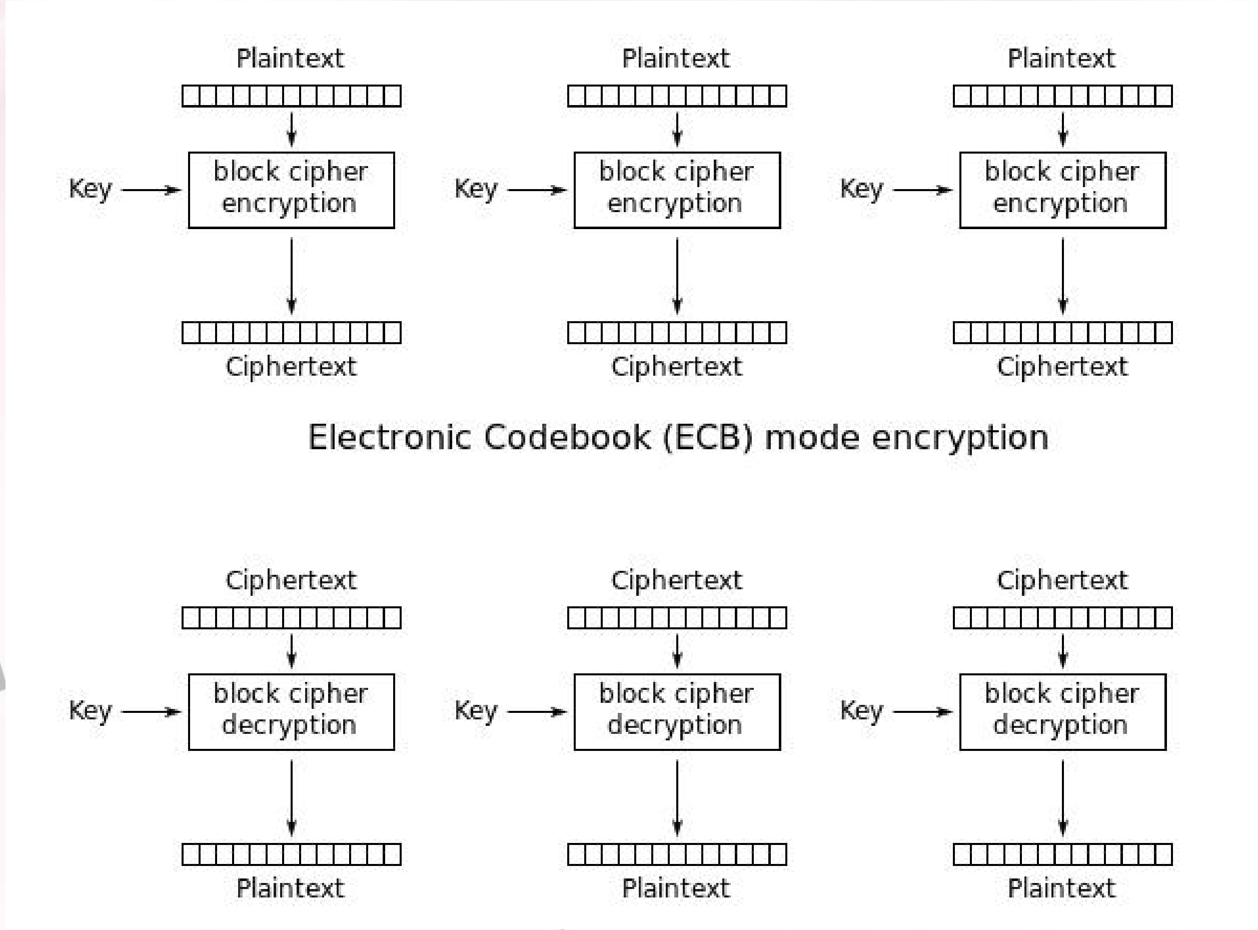
- 动画&手写解释列混淆

3

分组密码加密模式

What is？

ECB



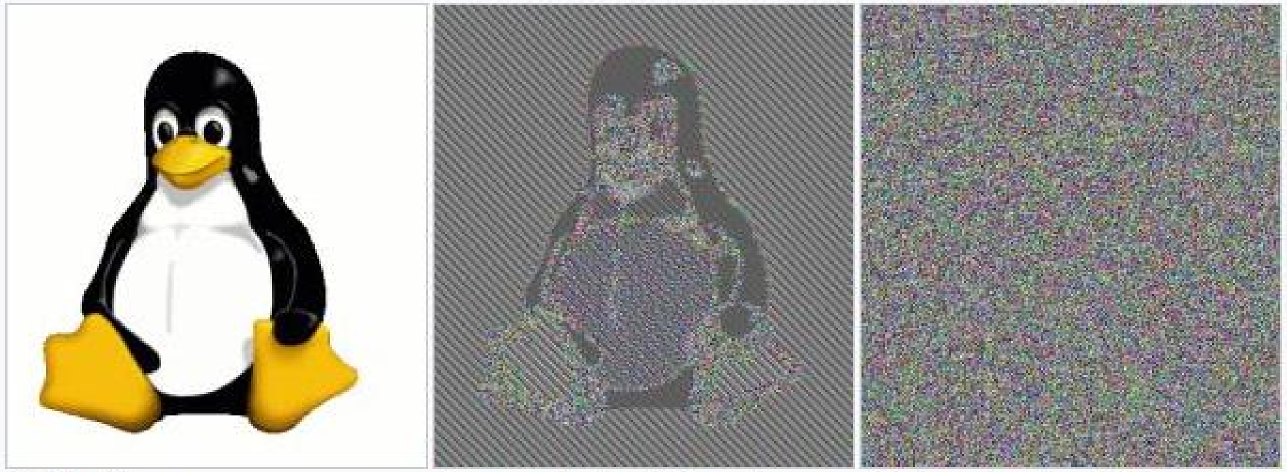
Plaintext
Plaintext
Plaintext
Key
block cipher encryption
Key
block cipher encryption
Key
block cipher encryption
Ciphertext
Ciphertext
Ciphertext
Electronic Codebook (ECB) mode encryption
Ciphertext
Ciphertext
Ciphertext
Key
block cipher decryption
Key
block cipher decryption
Key
block cipher decryption
Plaintext
Plaintext
Plaintext

ECB

- 实现简单。
- 不同明文分组的加密可以并行计算，速度很快。
- 同样的明文块会被加密成相同的密文块（lack of diffusion）

ECB

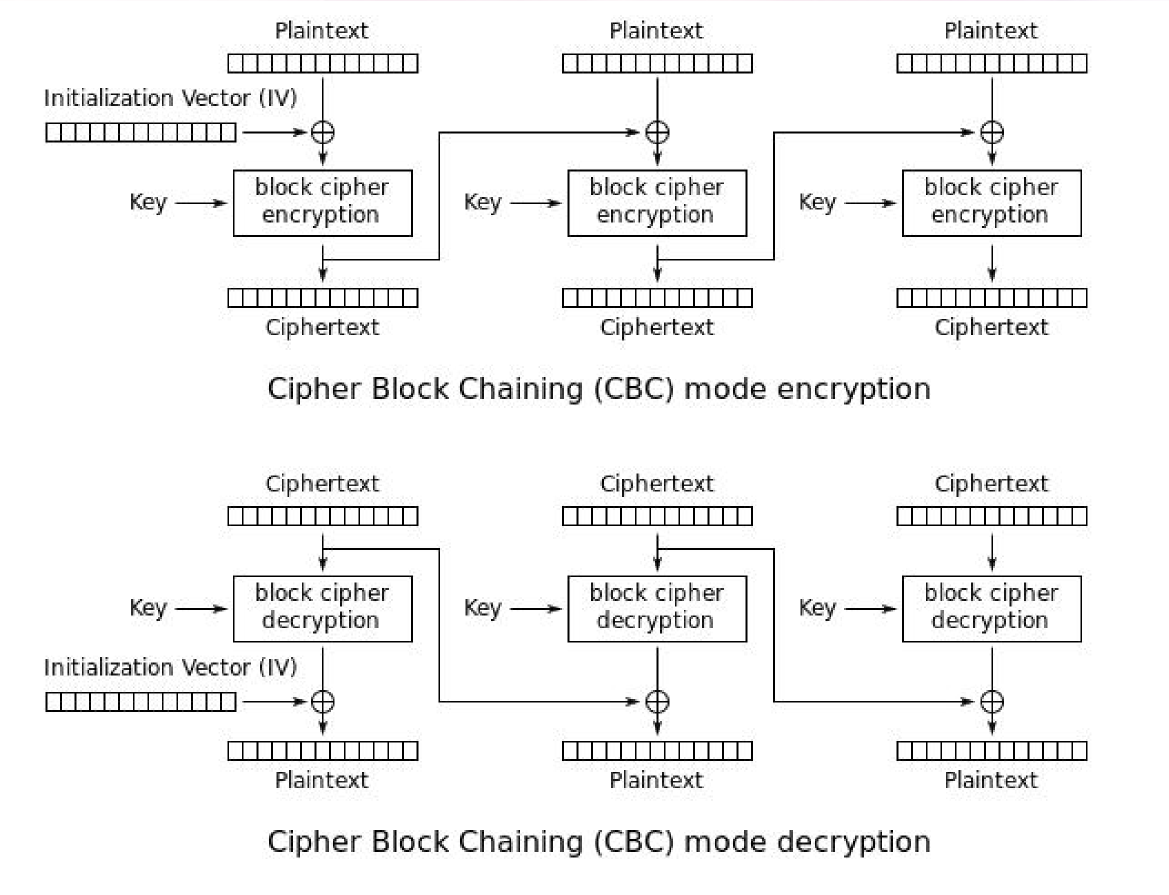
- 加密模式存在的意义



Original image | Encrypted using ECB mode | Modes other than ECB result in pseudo-randomness

The image on the right is how the image might appear encrypted with CBC, CTR or any of the other more secure modes—indistinguishable from random noise. Note that the random appearance of the image on the right does not ensure that the image has been securely encrypted; many kinds of insecure encryption have been developed which would produce output just as "random-looking".

CBC （PCBC）


Plaintext
Initialization Vector (IV)
Key
block cipher encryption
Ciphertext
Cipher Block Chaining (CBC) mode encryption
Ciphertext
Key
block cipher decryption
Initialization Vector (IV)
Plaintext
Cipher Block Chaining (CBC) mode decryption

CBC

- IV 必须是不可预测的，而且要保证完整性
- 两步错误传播特性
- 明文统计特性的隐藏

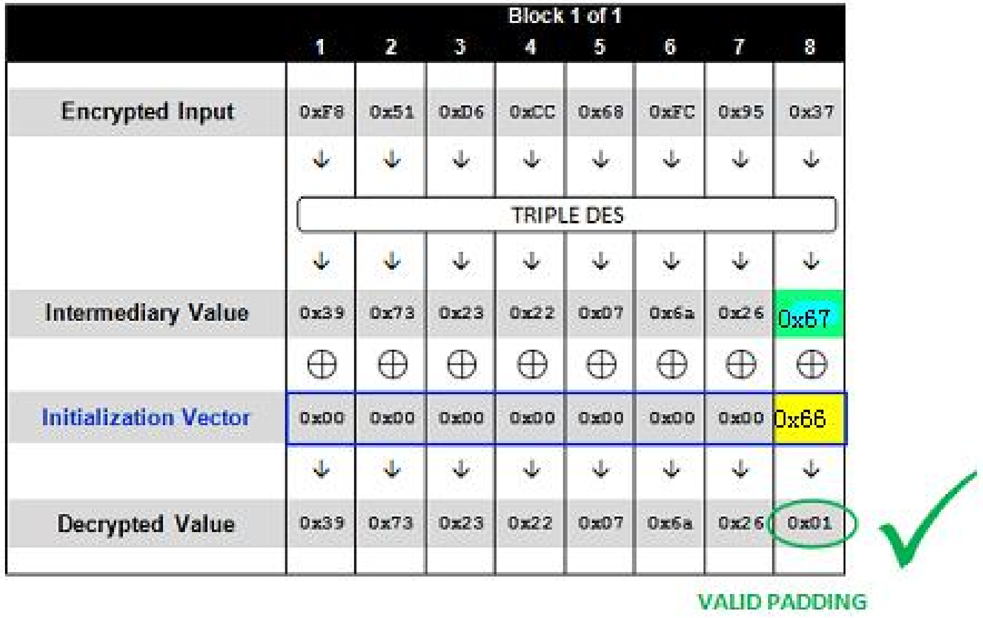
- 并行解密

针对CBC的攻击

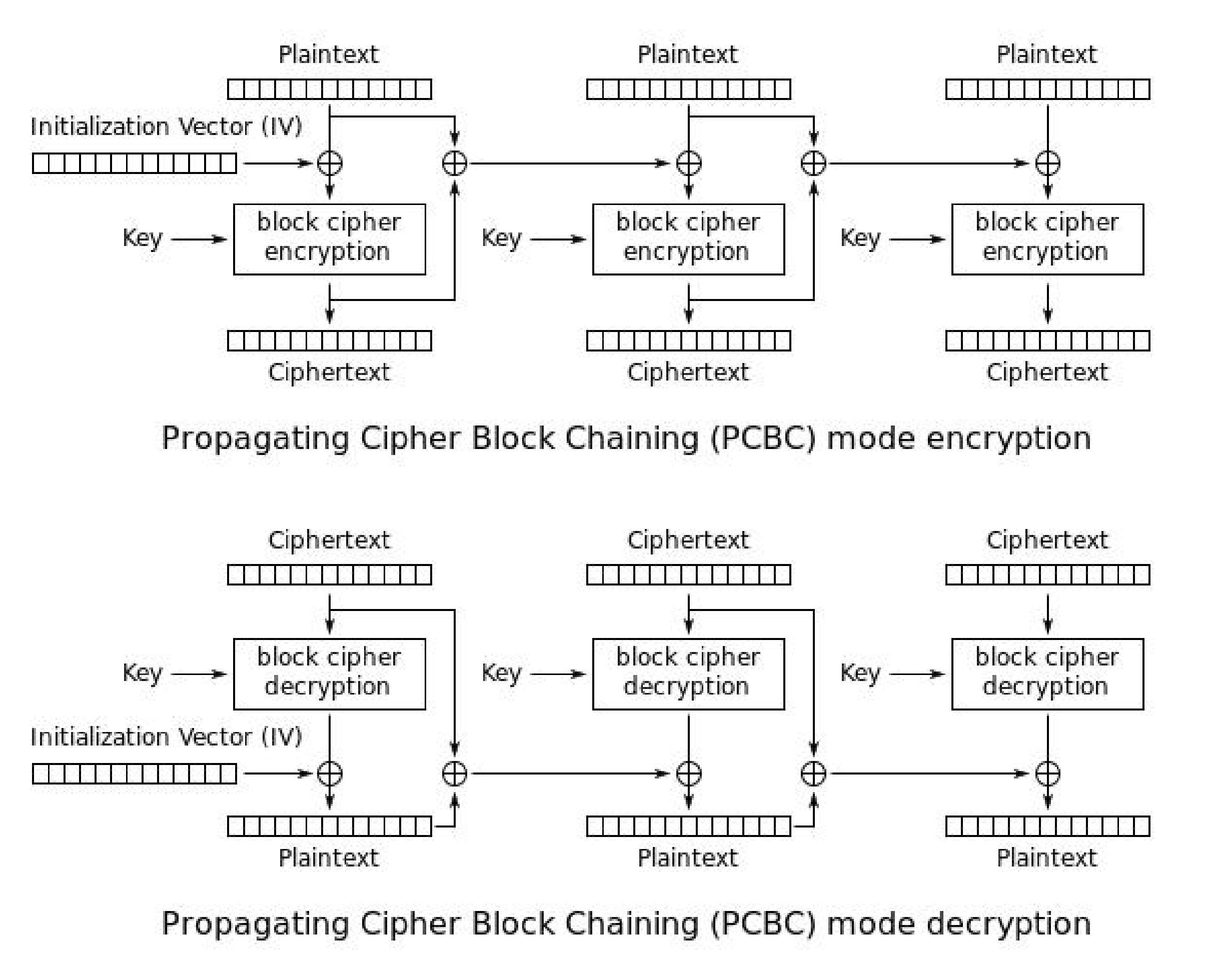
- 字节反转攻击
- Padding Oracle Attack

Padding Oracle Attack

Padding Oracle Attack

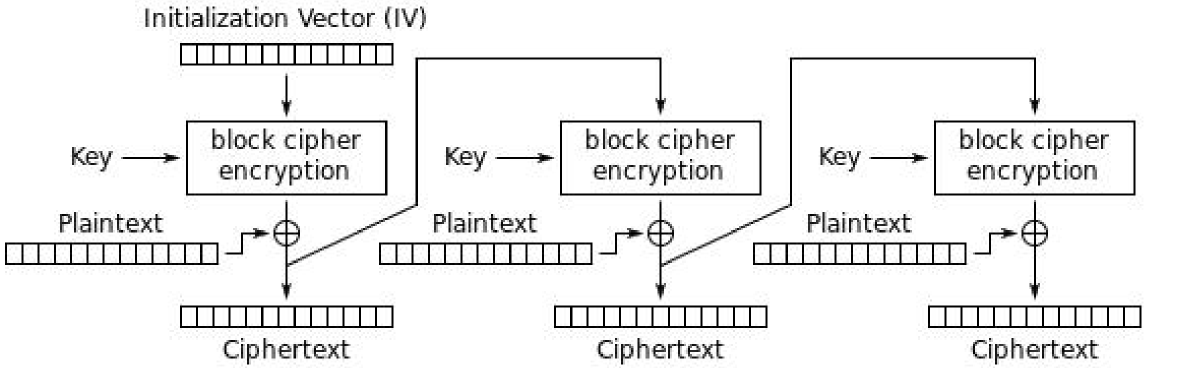


PCBC

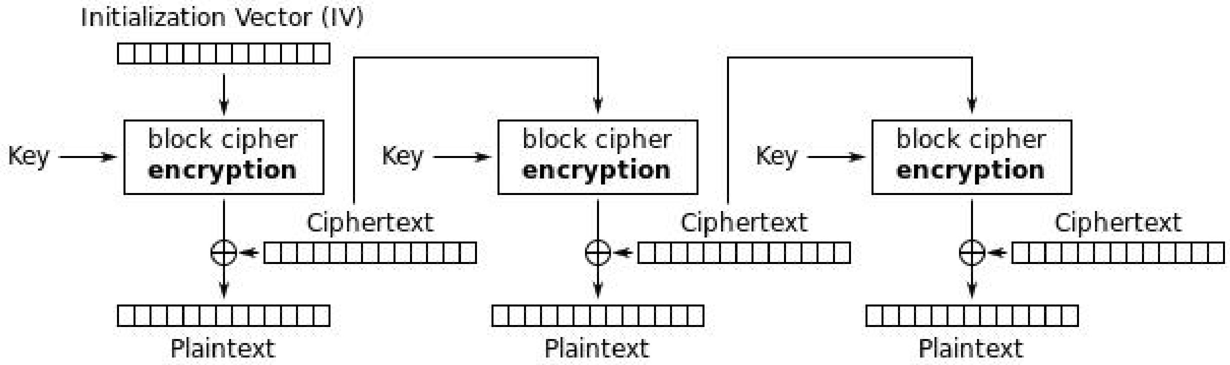


Plaintext
Plaintext
Plaintext
Initialization Vector (IV)
Key
block cipher encryption
Key
block cipher encryption
Key
block cipher encryption
Ciphertext
Ciphertext
Ciphertext
Propagating Cipher Block Chaining (PCBC) mode encryption
Ciphertext
Ciphertext
Ciphertext
Key
block cipher decryption
Key
block cipher decryption
Key
block cipher decryption
Initialization Vector (IV)
Plaintext
Plaintext
Plaintext
Propagating Cipher Block Chaining (PCBC) mode decryption

CFB



Cipher Feedback (CFB) mode encryption

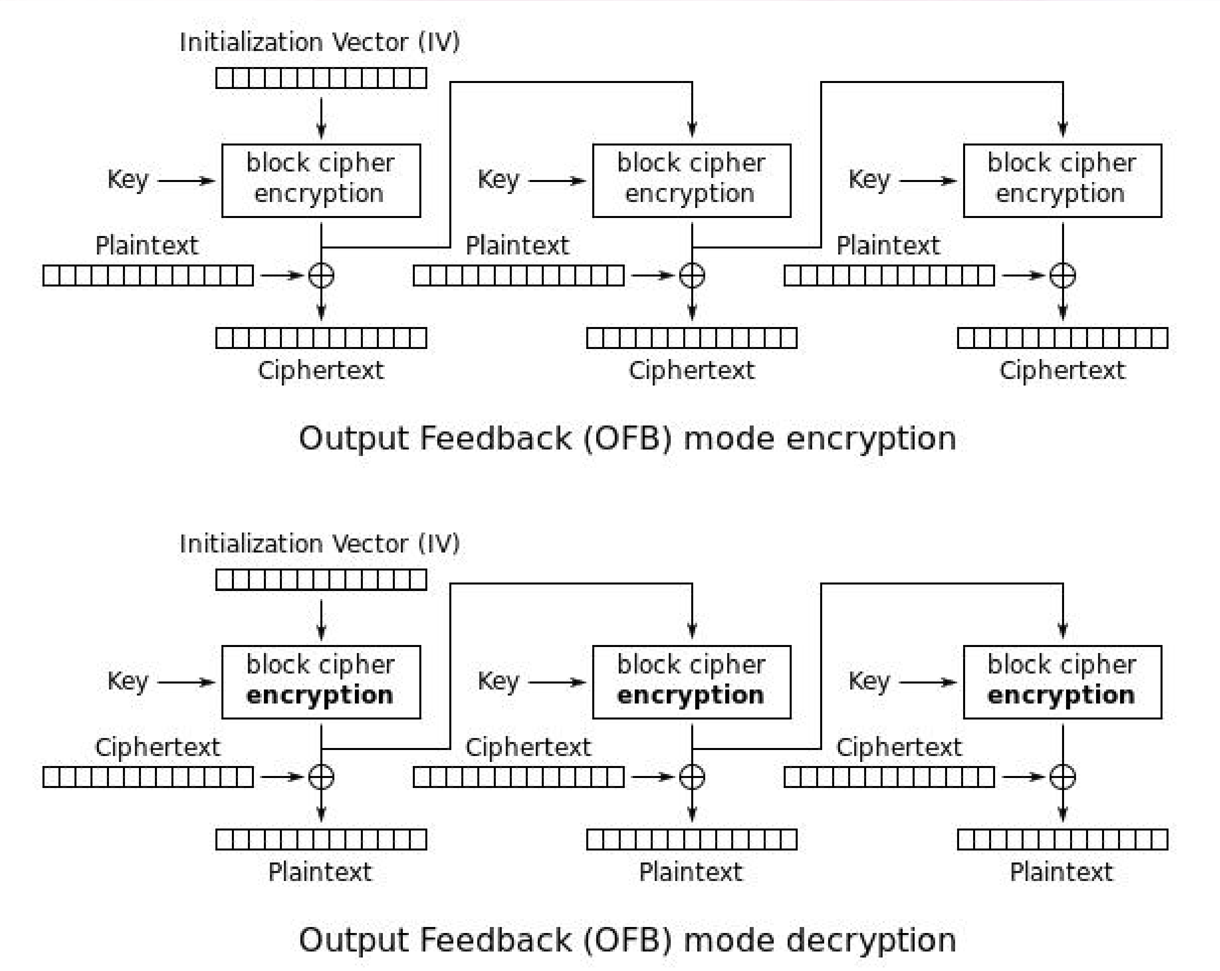


Cipher Feedback (CFB) mode decryption

CFB

- 并行状况与CBC相同
- 不需要padding

OFB

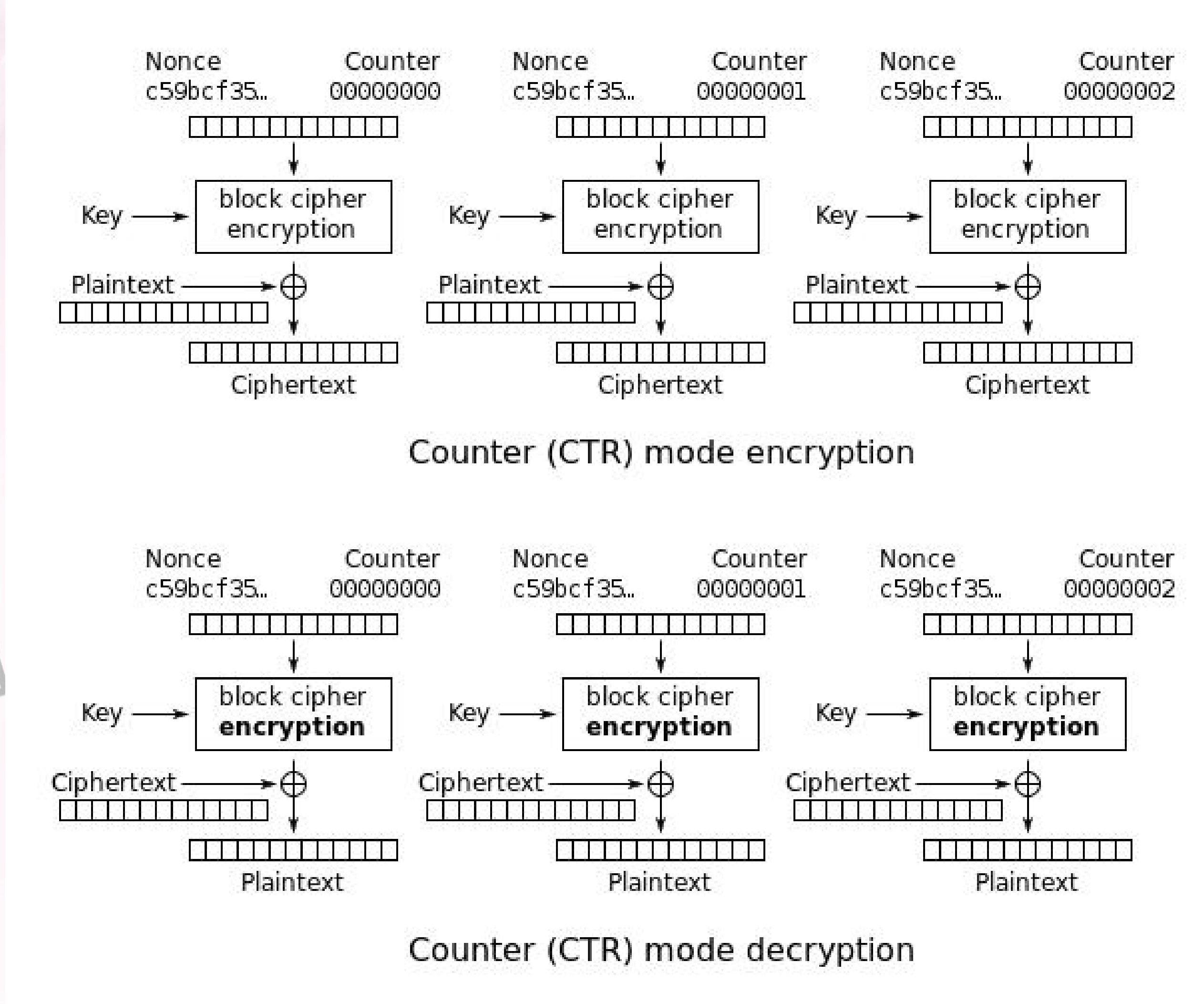

Initialization Vector (IV)
Key
block cipher encryption
Plaintext
Ciphertext
Output Feedback (OFB) mode encryption
Initialization Vector (IV)
Key
block cipher encryption
Ciphertext
Plaintext
Output Feedback (OFB) mode decryption

OFB

- 可以预先计算key stream

- 加解密均不能并行
- 可能存在cycle

CTR


Nonce
c59bcf35...
Counter
00000000
Nonce
c59bcf35...
Counter
00000001
Nonce
c59bcf35...
Counter
00000002
Key
block cipher encryption
Plaintext
Ciphertext
Counter (CTR) mode encryption
Nonce
c59bcf35...
Counter
00000000
Nonce
c59bcf35...
Counter
00000001
Nonce
c59bcf35...
Counter
00000002
Key
block cipher encryption
Ciphertext
Plaintext
Counter (CTR) mode decryption

加密模式间的比较

- 总结一下
- Please use GCM
- Message Authentic Code

3.5

Padding

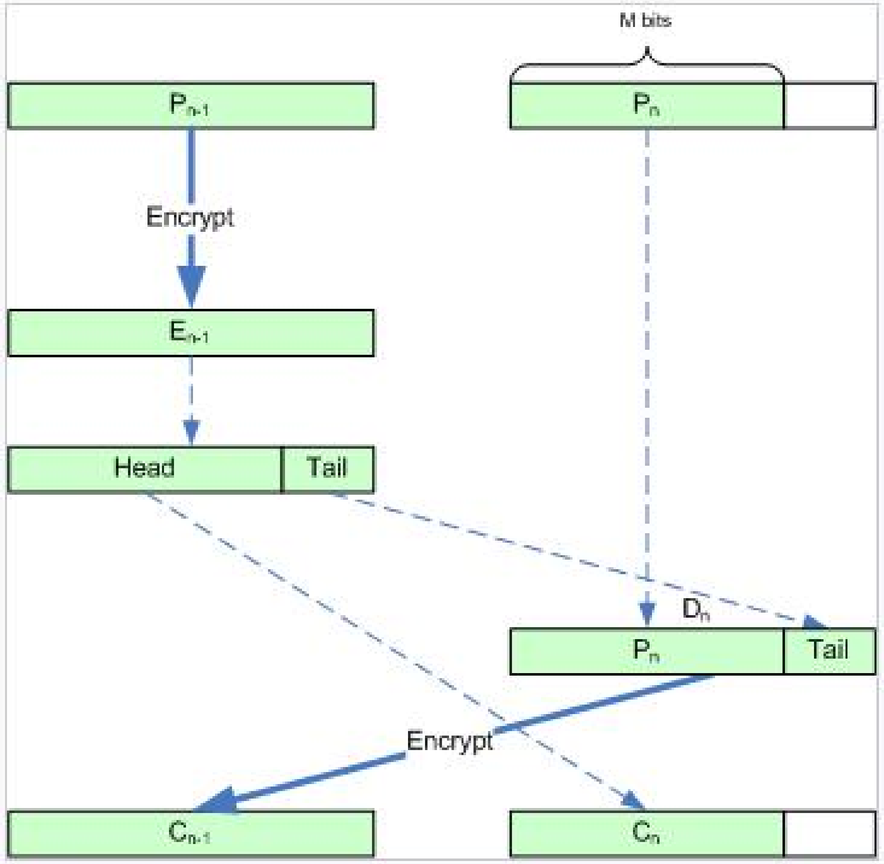
What is？

Padding

- PKCS#5

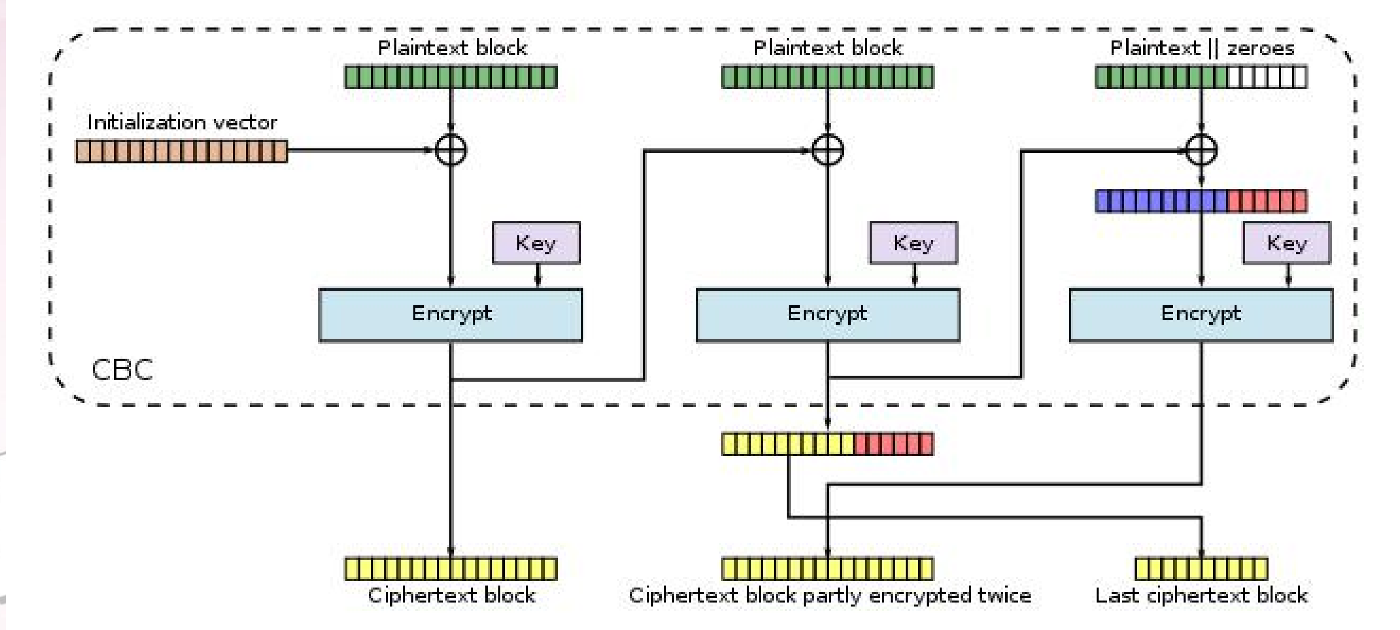
	BLOCK #1								BLOCK #2							
	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
Ex 1	F	I	G													
Ex 1 (Padded)	F	I	G	0x05	0x05	0x05	0x05	0x05								
Ex 2	B	A	N	A	N	A										
Ex 2 (Padded)	B	A	N	A	N	A	0x02	0x02								
Ex 3	A	V	O	C	A	D	O									
Ex 3 (Padded)	A	V	O	C	A	D	O	0x01								
Ex 4	P	L	A	N	T	A	I	N								
Ex 4 (Padded)	P	L	A	N	T	A	I	N	0x08	0x08	0x08	0x08	0x08	0x08	0x08	0x08
Ex 5	P	A	S	S	I	O	N	F	R	U	I	T				
Ex 5 (Padded)	P	A	S	S	I	O	N	F	R	U	I	T	0x04	0x04	0x04	0x04

Ciphertext stealing for ECB

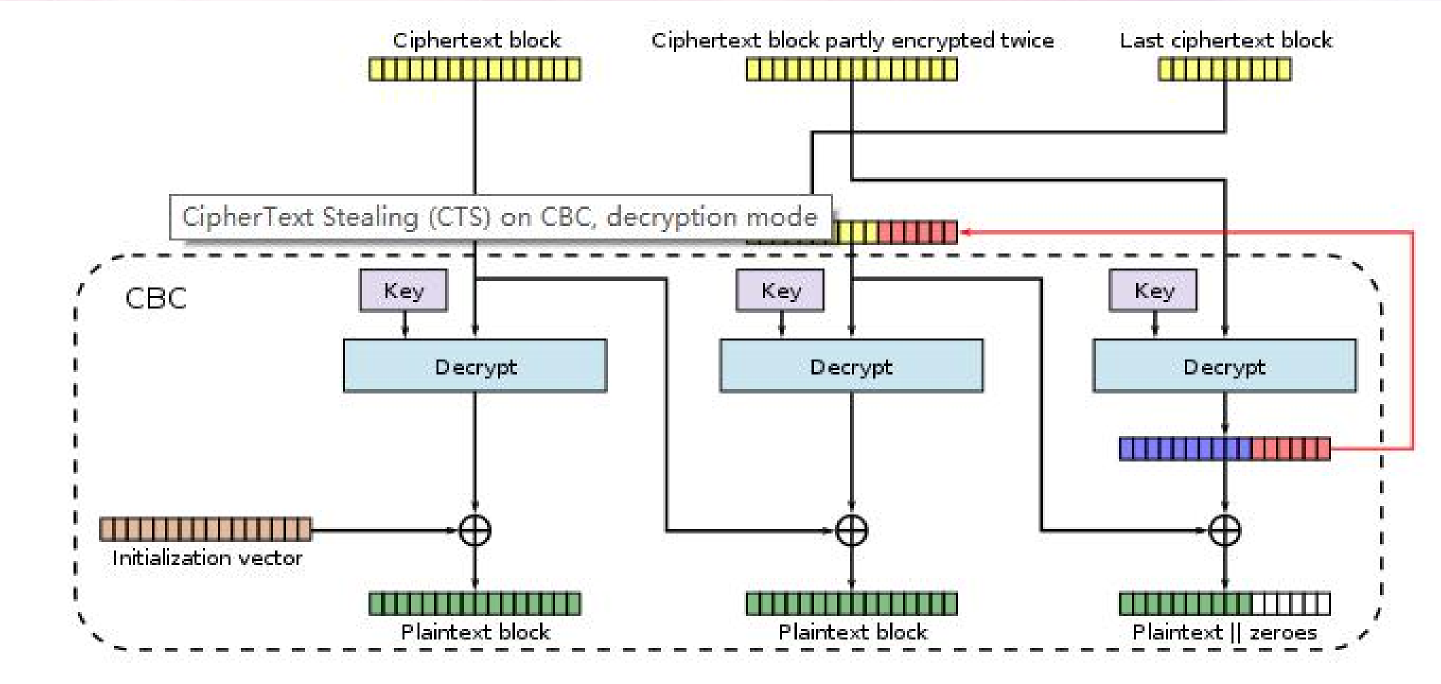


ECB Encryption Steps for CTS

Ciphertext stealing for CBC



Ciphertext stealing for CBC



4

公钥算法

What is？

公钥算法

- 公钥算法存在的意义
  - 公开信道上的通信
  - 更灵活的协议设计

公钥算法的安全性来源

- Discrete log problem (DLP)
  - CDH
  - DDH
  - ...
- Integer factorization
- RSA problem
- ...

- Final Report on Main Computational Assumptions in Cryptography

RSA

- 根据一定条件随机选择两个不同的大质数p和q，计算$N = p * q$
- 根据欧拉函数，$\varphi(N) = \varphi(p)*\varphi(q) = (p-1) * (q-1)$
- 选择一个与$\varphi(N)$互素的数字e，并求得e的逆元d

- Public Key: (N, e)
- Secrect Key (N, d)

ENC & DEC

- $C = pow(M, e, N)$
- $M = pow(C, d, N)$

- $M = pow(C, d, N)$
    $= pow(pow(M, e, N), d, N)$
    $= pow(M, e*d, N)$
    $= pow(M, k*\varphi(N) + 1, N)$

费马小定理

- pow(a, φ(N), N) = 1
- N, a 为正整数；N, a互质

- 证明：

  X1...Xφn == a1...aφn

快速幂算法

```
def fastExp(m, d, N):
    ret = 1
    while d > 0:
        if d&1:
            ret = ret * m % N
        d = d // 2
        m = pow(m, 2, N)
    return ret
```

中国剩余定理加速法

- 已知p, q, e, c

- cp = c mod p
- cq = c mod q

- dp = d mod (p-1)
- dq = d mod (q-1)

- pp = pow(cp, dp, p)
- pq = pow(cq, dq, q)

- p = q * inv(q, p) * pp + p * inv(p, q) * pq mod (p*q)

- [举个例子]

中国剩余定理

$$(S):\quad \begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \\ \quad \vdots \\ x \equiv a_n \pmod{m_n} \end{cases}$$

中国剩余定理

$$M = m_1 \times m_2 \times \cdots \times m_n = \prod_{i=1}^{n} m_i$$

$$M_i = M / m_i, \quad \forall i \in \{1, 2, \cdots, n\}$$

$$t_i M_i \equiv 1 \pmod{m_i}, \quad \forall i \in \{1, 2, \cdots, n\}$$

$$x = a_1 t_1 M_1 + a_2 t_2 M_2 + \cdots + a_n t_n M_n + kM = kM + \sum_{i=1}^{n} a_i t_i M_i$$

$$x = \sum_{i=1}^{n} a_i t_i M_i$$

CTF中的RSA

- pem格式的证书
  - openssl rsautl -encrypt -in FLAG -inkey public.pem -pubin -out flag.enc
  - openssl rsa -pubin -text -modulus -in warmup -in public.pem
- pcap流量包
- nc端口交互 pwntools
- N=******; e=*****...

5

RSA使用中常见的错误

SHOULD NOT DO THIS

- 过小的N
- 过小的d
- 过小的e
- 重复使用p，q
- 不恰当的pq特征
- 广播同一段明文的不同密文
- 不同的e共用n
- 提供 Padding Oracle

过小的N

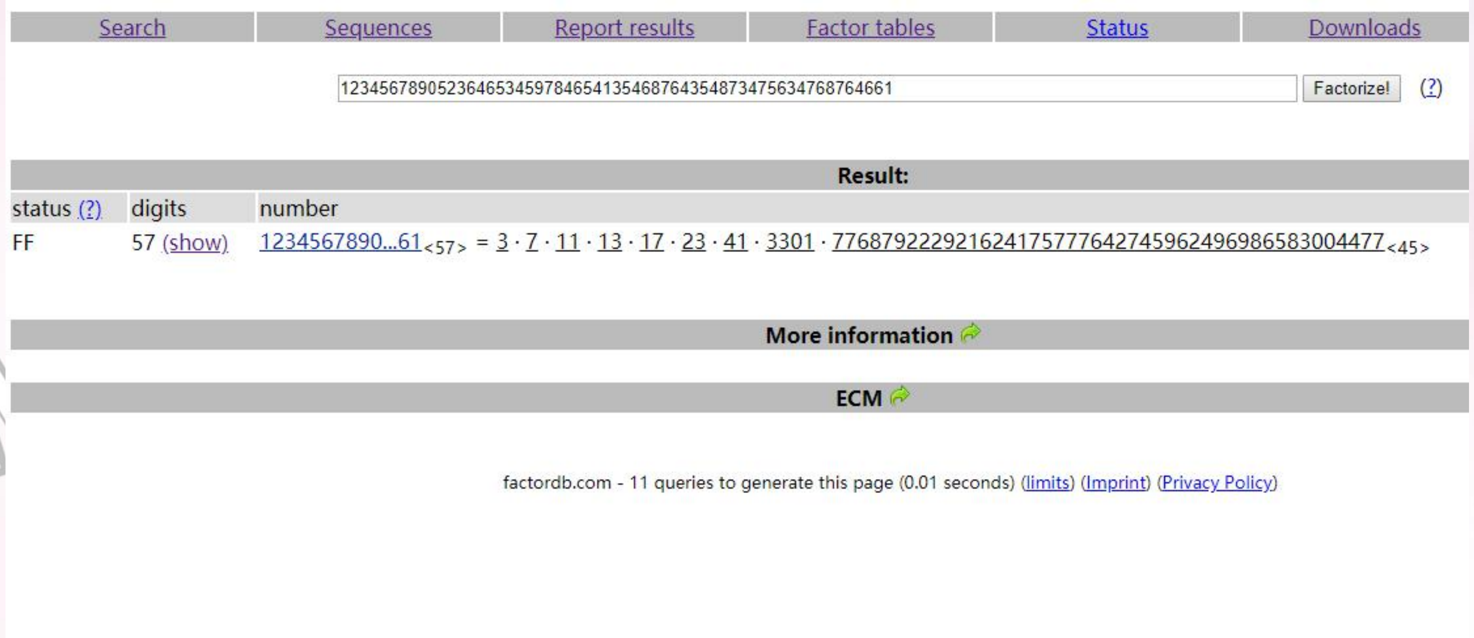
- 素数分解问题是困难的，但是可以通过计算机进行暴力分解。
- 针对RSA最流行的攻击一般是基于大数因数分解。1999年，RSA-155 (512 bits)被成功分解，花了五个月时间（约8000 MIPS年）和224 CPU hours在一台有3.2G中央内存的Cray C916计算机上完成。
- 2009年12月12日，编号为RSA-768（768 bits, 232 digits）数也被成功分解。这一事件威胁了现通行的1024-bit密钥的安全性，普遍认为用户应尽快升级到2048-bit或以上。
- 2010年，又提出了一些针对1024bit的n的分解的途径，但是没有正面分解成功。

过小的N

- 一般认为2048bit以上的n是安全的。现在一般的公钥证书都是4096bit的证书。

过小的N

- online DB：http://factordb.com



过小的N

- 本机：RSATool，yafu

```
$ ./yafu-x64.exe
factor(1234567890523646534597846541354687643548734756347687646610)

fac: factoring 123456789052364653459784654135468764354873475634768764661
fac: using pretesting plan: normal
fac: no tune info: using qs/gnfs crossover of 95 digits
div: primes less than 10000
fmt: 1000000 iterations
Total factoring time = 0.0430 seconds


***factors found***

P1 = 3
P1 = 7
P2 = 11
P2 = 13
P2 = 17
P2 = 23
P2 = 41
P4 = 3301
P45 = 776879222921624175777642745962496986583004477

ans = 1
```

过小的e

- pow(m, e) < N (e == 3)
- Cipher Text = pow(m, e) + k*N (e == 3)

过小的d

- rsa-wiener-attack
- https://github.com/pablocelayes/rsa-wiener-attack

重复使用PQ

- $N1 = p1 * q1$
- $N2 = p1 * q2$

- $gcd(N1, N2) = p1$

不恰当的pq特征

- Fermat方法 & Pollard rho方法

- 在p，q的取值差异过大，或者p，q的取值过于相近的时候，Format方法与Pollard rho方法都可以很快将n分解成功。

- Just Yafu it

广播攻击

- C1 = pow(m, e, N1)
- C2 = pow(m, e, N2)
- C3 = pow(m, e, N3)

- C_ = pow(m, e, N1*N2*N3)

- (中国剩余定理)

共模攻击

- C1 = pow(m, e1, N)
- C2 = pow(m, e2, N)

- if gcd(e1, e2) == 1 --> e1*r1 + e2*r2 == 1 (r1*r2<0)

- m = pow(C1, r1, N)*pow(C2, r2, N)
  = pow(m, e1*r1+e2*r2, N)

Padding Oracle

- SSL with PKCS#1 v1.5
- 0x00 0x02 [some non-zero bytes] 0x00 [here goes M]

0002	09ad829ffb3a98b91a1b ··· 089333f7ca7b4070f272	00	68656c6c6f20776f726c6421

- C' = C*pow(s, e, N) % N
- m' = m*s % N
- Randomly choose s(once every 30000 to 130000 attempts got a padded value)
- After a few million connections in all, the attacker learned enough to pinpoint the exact m

An easier scenario

- 0b 1 [here goes M]

- server will return an error when Plain Text start with 0 at 1023's bit

Padding Oracle's paper & realization

- Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1
- https://github.com/RUB-NDS/TLS-Attacker
- Plz use OAEP!

ATTACK ON RSA

- Coppersmith Attack
- Side channel

Coppersmith’ s Theorem

- Coppersmith method, proposed by Don Coppersmith 是一种主要利用 Lenstra–Lenstra–Lovász lattice basis reduction algorithm (LLL) 找到低系数多项式根的算法

- 在密码学中，Coppersmith方法主要用于攻击RSA

Stereotyped Messages Attack

- Plain Text --> “The PassWord is: *******”
- Attacker knows: M = M0 + x
- Attacker has to solve:
  - F(x0) = (M0 + x0)^e - C = 0 (mod N)
- 如果x0和e足够小的话，LLL算法可以以线性的时间复杂度解决这个问题
  - x0 < X
  - d = e

$$X = N^{\frac{1}{d}-\epsilon} \text{ for } \frac{1}{d} > \epsilon > 0$$

Factoring with high bits known

- p(x, y) == 0
- p(x0, y0) == P*Q - N == 0

$$
\begin{aligned}
P &= P_0 + x_0 \\
Q &= Q_0 + y_0 \\
|x_0| &< X = P_0 / N^{(1/4)+\epsilon} \\
|y_0| &< Y = Q_0 / N^{(1/4)+\epsilon} \\
p(x, y) &= (P_0 + x)(Q_0 + y) - N \\
&= (P_0 Q_0 - N) + Q_0 x + P_0 y + xy
\end{aligned}
$$

LLL的实现

- https://github.com/mimoo/RSA-and-LLL-attacks

side-channel attack

- RSA被能量攻击的点在？
  - 快速幂算法
  - 中国剩余定理加速解密

快速幂算法

```
def fastExp(m, d, N):
    ret = 1
    while d > 0:
        if d&1:
            ret = ret * m % N
        d = d // 2
        m = pow(m, 2, N)
    return ret
```

中国剩余定理加速法

- 已知p, q, e, c

- cp = c mod p
- cq = c mod q

- dp = d mod (p-1)
- dq = d mod (q-1)

- pp = pow(cp, dp, p)
- pq = pow(cq, dq, q)

- p = q * inv(q, p) * pp + p * inv(p, q) * pq mod (p*q)

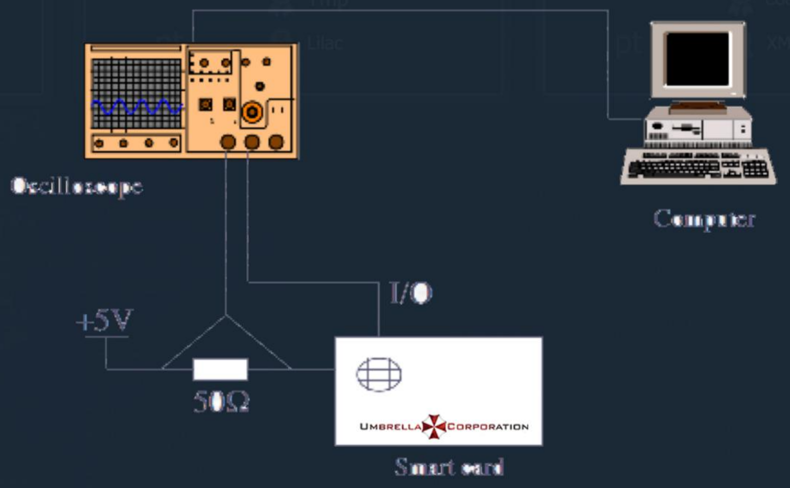
- [举个例子]

HACK IN THE CARD I

Welcome to work for C.I.A. Our agent 47 has successfully penetrated to an evil company and sent this secret smart card to us. Intelligence department said the crypto chip on this card is doing RSA operation and the public key we got is here(attachments -> publickey.pem). Your mission is to extract the private key embedded in this smart card and decrypt the following hex-encoded ciphertext.

014b05e1a09668c83e13fda8be28d148568a2342aed833e0ad646bd45461da2decf9d538c2d3ab245b272873beb112586bb7b17dc4
b30f0c5408d8b03cfbc8388b2bd579fb419a1cac38798da1c3da75dc9a74a90d98c8f986fd8ab8b2dc539768beb339cadc13383c62b5
223a50e050cb9c6b759072962c2b2cf21b4421ca73394d9e12cfbc958fc5f6b596da368923121e55a3c6a7b12fdca127ecc0e8470463f
6e04f27cd4bb3de30555b6c701f524c8c032fa51d719901e7c75cc72764ac00976ac6427a1f483779f61cee455ed319ee9071abefae44
73e7c637760b4b3131f25e5eb9950dd9d37666e129640c82a4b01b8bdc1a78b007f8ec71e7bad48046

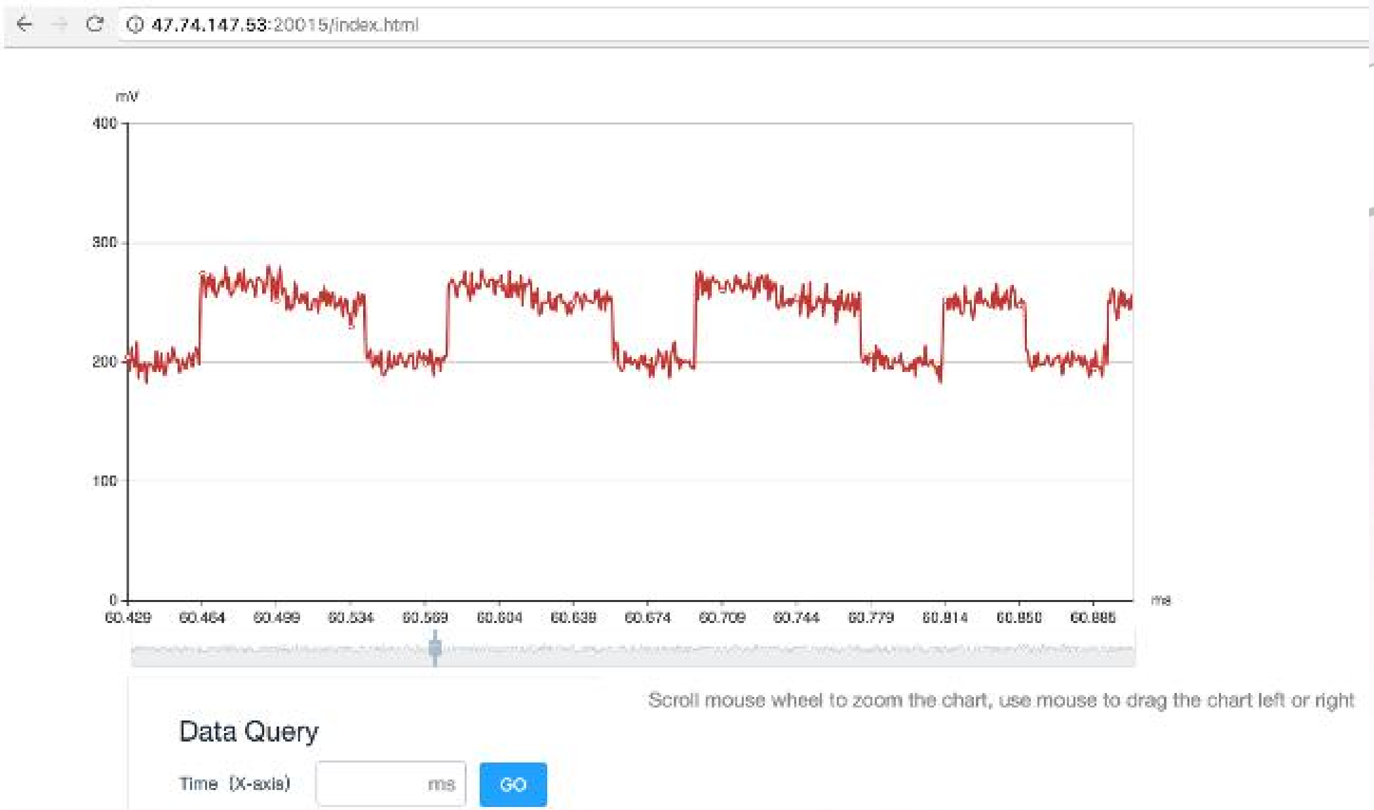
The progress of hacking was going well untill it got stuck, what we did so far is that as you can see the smartcard, an oscilloscope, a computer (act as the card reader) and a resistor are plugged into a circuit board. The circuit diagram is given as follows.



We finally managed to decrypt a crafted message and captured voltage variation(http://47.74.147.53:20015/index.html) of the resistor during the whole process. Now we are counting on you to do the rest...

Download Attachments

side-channel attack



Tools

- Github

THANK YOU

FOR YOUR ATTENTION