密码学专题一：对称密码

戴勤明

浙江大学

对称密码

在对称加密系统中，加密和解密采用相同的密钥。因为加解密密钥相同，需要通信的双方必须选择和保存他们共同的密钥，各方必须信任对方不会将密钥泄密出去，这样就可以实现数据的机密性和完整性。比较典型的算法有DES（Data Encryption Standard），3DES，AES（ Advanced Encryption Standard）， Blowfish等。

对称密码算法的优点是计算开销小，算法简单，加密速度快，是目前用于信息加密的主要算法。

提纲

- DES/AES加解密流程
- 分组密钥操作模式和填充模式
- 攻击手法
- 密码分析技术

DES

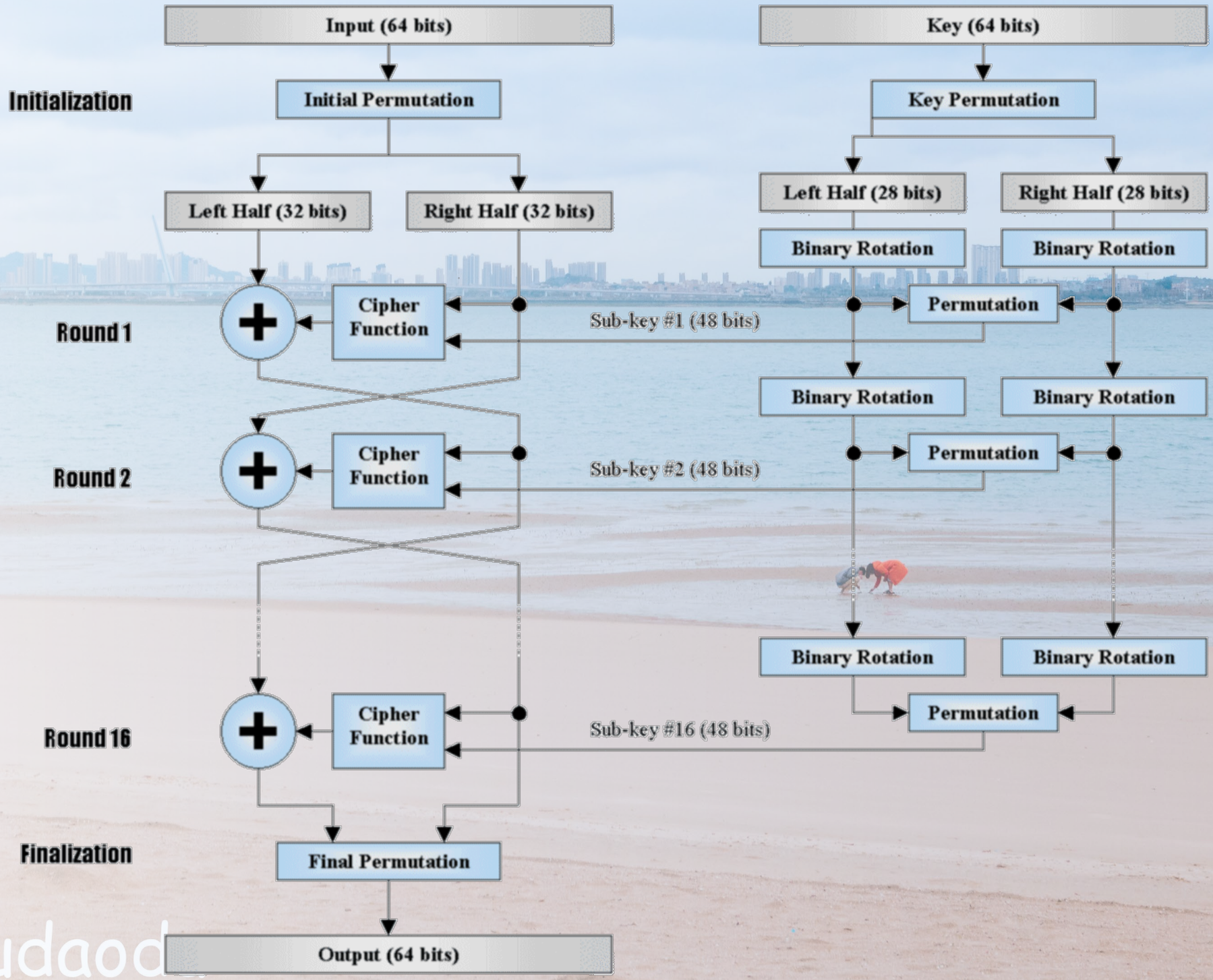
基本信息

- 输入 64 位。
- 输出 64 位。
- 密钥 64 位，使用 64 位密钥中的 56 位，剩余的 8 位要么丢弃，要么作为奇偶校验位。

迭代结构

- 明文经过 16 轮迭代得到密文。
- 密文经过类似的 16 轮迭代得到明文。

DES



DES

迭代结构

加密：

$$L_{i+1} = R_i$$

$$R_{i+1} = L_i \oplus F(R_i, K_i)$$

解密：

$$R_i = L_{i+1}$$

$$L_i = R_{i+1} \oplus F(L_{i+1}, K_i)$$

DES

核心部件

- 子密钥生成
- 初始置换
- F 函数
- E 扩展函数
- S 盒
- P 置换
- 最后置换

DES

（1）子密钥生成

在DES中，加密者输入的明文和密钥都是64 bit，其中只有56 bit是有用的位数（因为有8位为奇偶校验位）。但是DES加密过程有16轮循环函数，其中需要用到16个密钥，所以要将这56 bit密钥扩展生成16个**48 bit** 的子密钥。步骤如下：

（i）密钥初始置换，利用密钥置换表1去掉原64位密钥中的8位，并打乱剩下的56位

(表1)密钥置换表(64bit)													
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

DES

（1）子密钥生成

（ii）将（i）得到的密钥分为left和right两部分，各28bit，然后利用子密钥循环左移表2对left和right进行不断循环左移，每轮left和right进行拼接，得到16个子密钥

(表2)子密钥循环左移位数表(16次)
1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1

（iii）将（ii）得到的16个子密钥分别使用密钥压缩表3进行压缩，共得到16个最终的子密钥，每个子密钥将在每轮加密中被使用到

(表3)密钥压缩置换表(56bit->48bit)											
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

（2）初始置换

利用初始置换IP(Initial Permutation)对明文P进行换位处理，打乱原来的次序，得到一个乱序的64 bit 明文组。

P:　　01010011 01100101 01100011 01110010 01100101 01110100 00100000 01001101

置换后: 10111111 00101001 10110010 10010111 00000000 01111110 10000000 00001101

打乱后的比特将被分为Left，Right两部分，各32bit。

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

初始置换IP盒

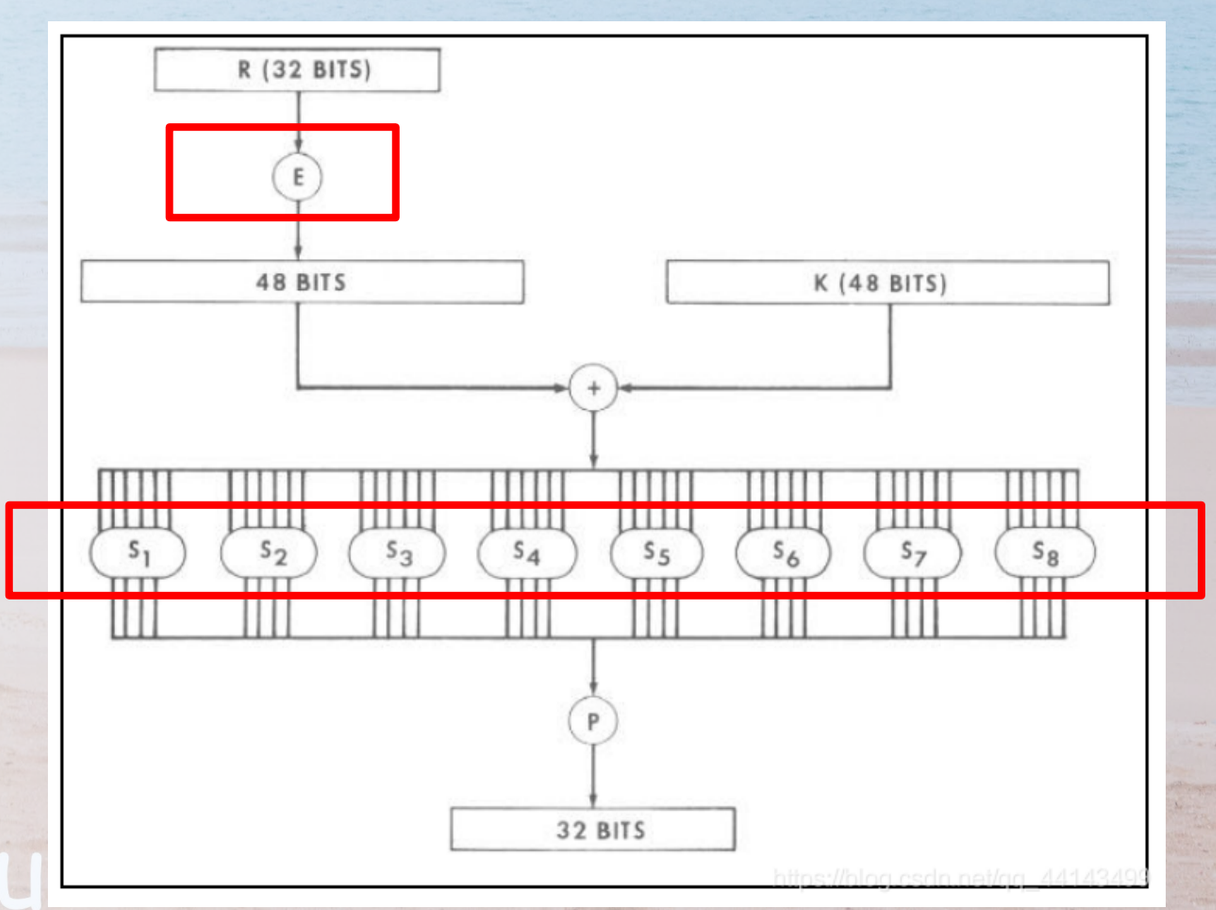
DES

（3）F 函数

函数以长度为32 的比特串 A = Right(32 bit)作为第1个输入，以长度为48的比特串

SubKey(48 bit)作为第2个输入，产生的输出是长度为32的比特串。

F(Ri) = P(S(E(Ri) + Ki))



DES

（4）E 扩展函数

将A = Right(32 bit)扩展到48bit，其中部分比特位将重复出现

E

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

DES

（5）S 盒

将 E(A) 和 K 进行异或操作后，把比特串分为8组，一组 6 bit，分别对每一组进行S盒代换。

经过S盒，每一组由 6 bit 缩减为 4 bit。48bit将再次变为32bit。

S盒的行号从0到3，列号从0到15。

代换的过程如下，例如需要代换的第一组数据输入为011001，则第一位0和最后一位1组合成的01即为行号，中间的1101为列号，第一组数据对应S1

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

DES

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_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

DES

（6）P 置换

P为固定置换，将经过S盒变换得到的32 bit进行一个置换操作。

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

DES

（7）最后置换

在十六轮运算完毕之后，最后再进行一次置换

(表8)末置换表(64bit)															
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

Python DES

```python
from Crypto.Cipher import DES

k = b'01234567'
plain = b'thistest'

des = DES.new(key=k, mode=DES.MODE_ECB)

print(des.encrypt(plain))
```

➜ pip3 install pycryptodome

➜ python3 des_test.py

b'\xe8\x03:[?\x8f\xdc\xd6'

AES

基本信息

输入：128 位

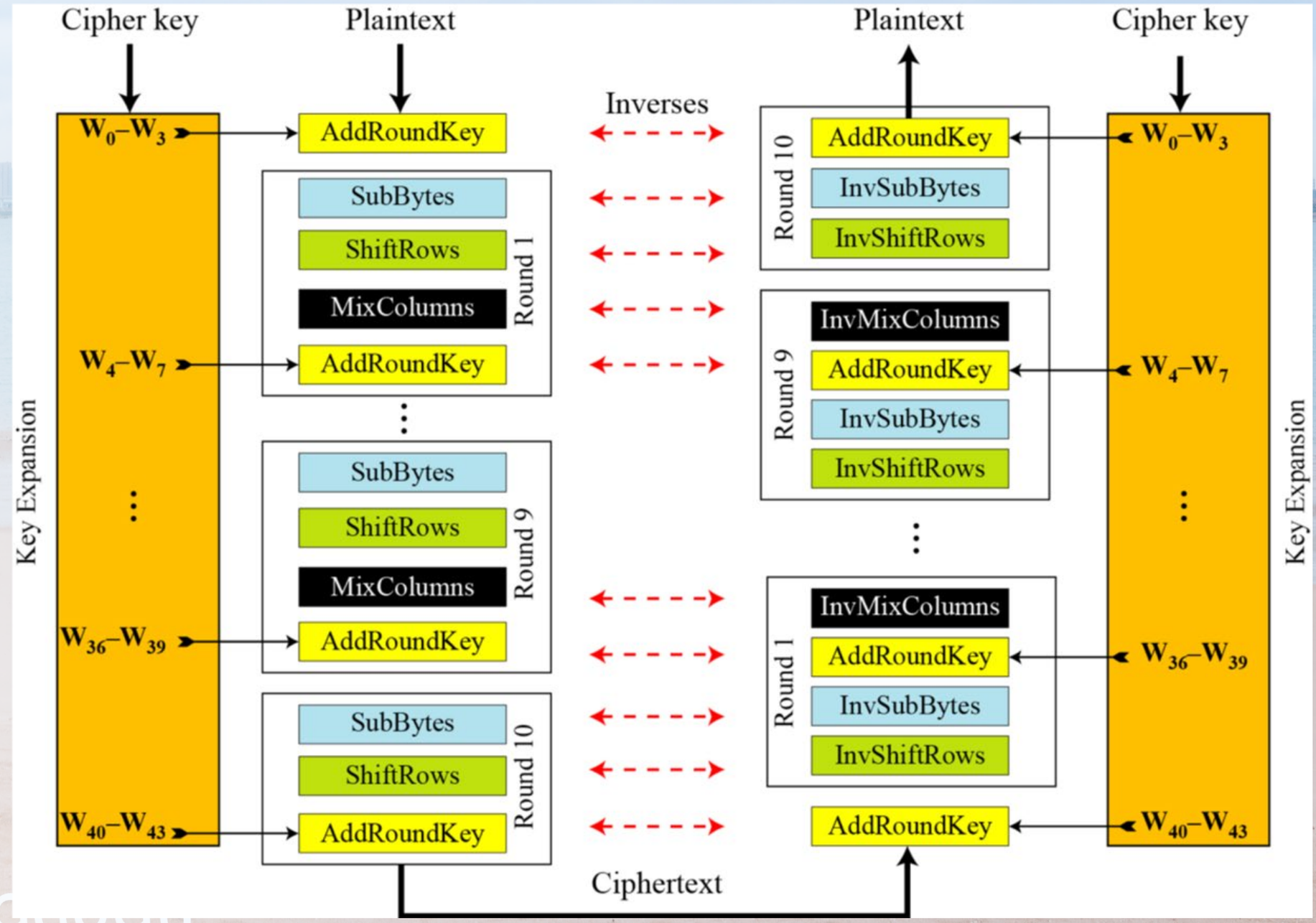
输出：128 位

其迭代轮数与密钥长度有关系：

密钥长度（比特）	迭代轮数
128	10
192	12
256	14

AES

AES加解密结构

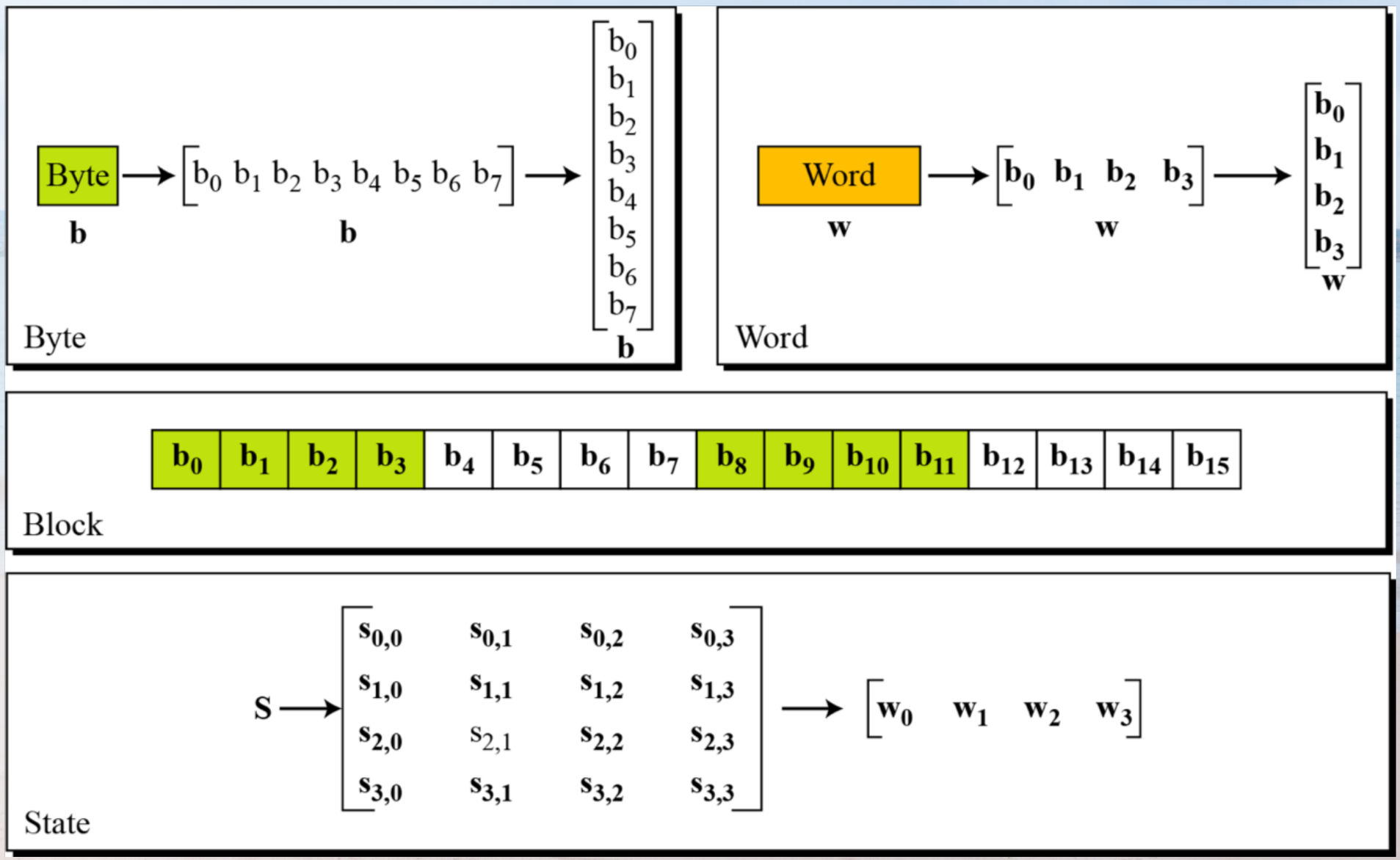


AES

AES加解密关键的部件

- 密钥扩展
- 轮密钥加，AddRoundKey
- 字节替换，SubBytes
- 行移位，ShiftRows
- 列混淆，MixColumns

AES



AES

（1）密钥扩展

密钥扩展的目的是将一个128位的密钥扩展10次变成11个128位的秘钥来用于接下来的轮密钥加操作。

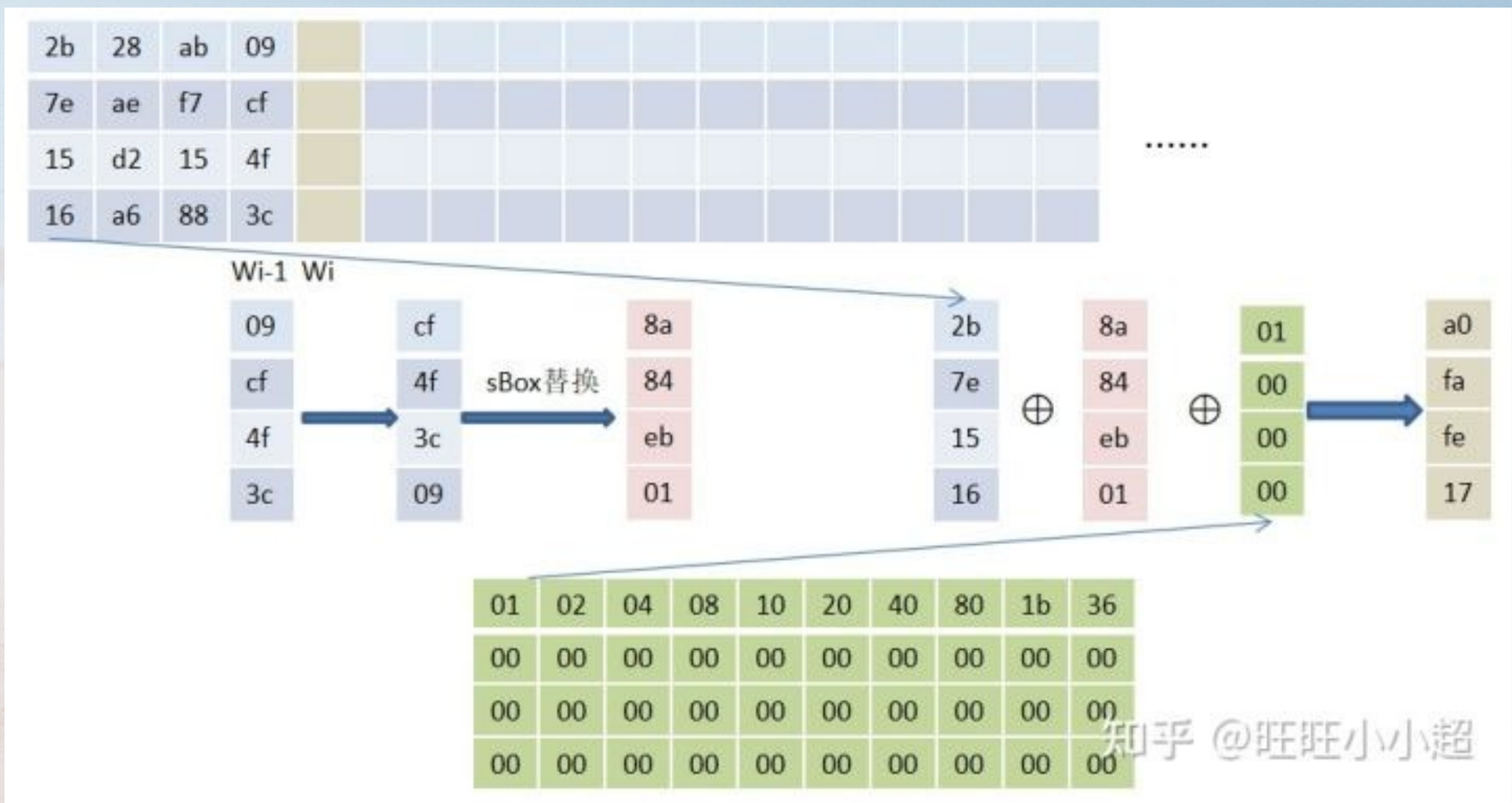
密钥扩展操作每次是对一个word（4个字节）进行操作，扩展40次，加上原先4个word共44个word，每4个word构成一个轮密钥。

AES

（1）密钥扩展

当i%4=0时，$W_i = W_{i-4} + g(W_{i-1})$，函数g：左移1个字节，Sbox替换，轮常数异或

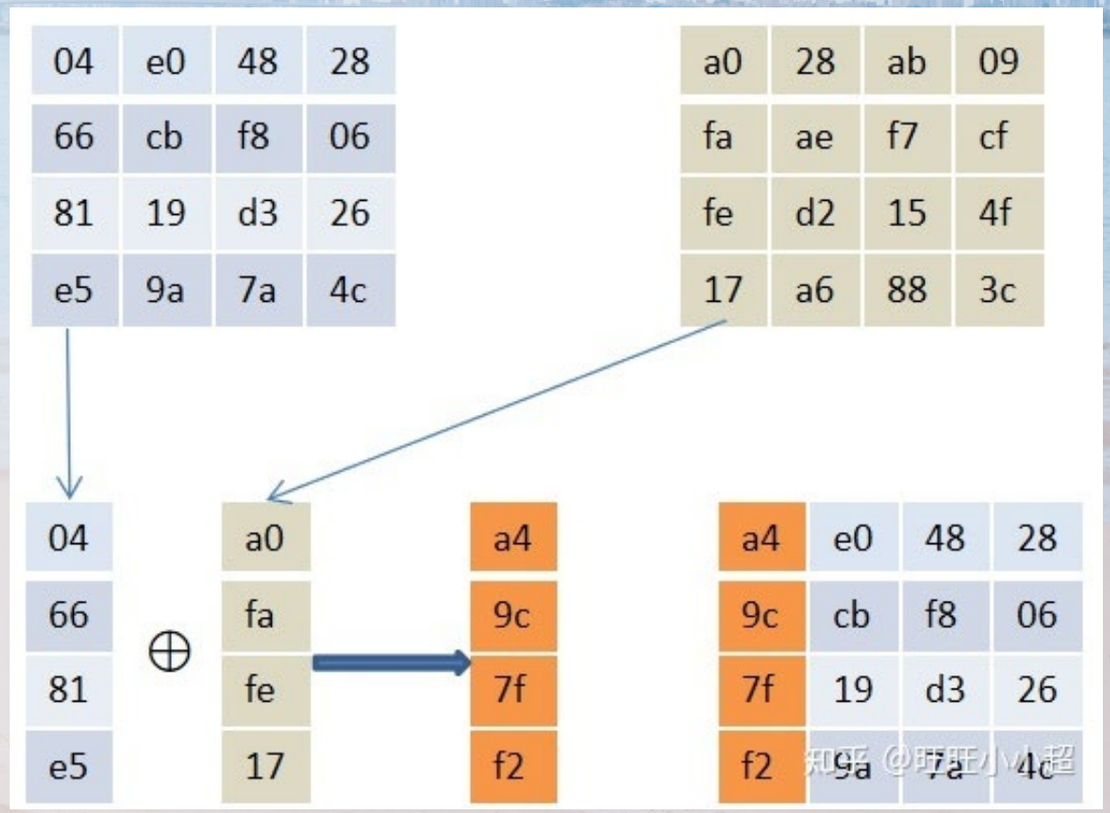
当i%4≠0时，$W_i = W_{i-4} + W_{i-1}$



AES

（2）轮密钥加，AddRoundKey

中间密文和轮密钥进行异或



AES

（3）字节替换，SubBytes

字节代换就是将被替换的字节按照高4位做行坐标，低4位做列坐标在sBox中找到替换到的字节表示。

比如，输入0x4f，即第4行，第15列，得到0x84

		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

(a) S-box

AES

（4）行移位，ShiftRows

第一行保持不变，第二行循环左移1字节，第三行2字节，第四行3字节。

0	1	2	3
4	5	6	7
8	9	a	b
c	d	e	f

0	1	2	3
5	6	7	4
a	b	8	8
f	c	d	e

AES

（5）列混淆， MixColumns

GF(2^8)域下的矩阵操作，简单来说，对state左乘一个矩阵：

02	03	01	01
01	02	03	01
01	01	02	03
03	01	01	02

X

d4	e0	b8	1e
bf	b4	41	27
5d	52	11	98
30	ae	f1	e5

=

04	e0	48	28
66	cb	f8	06
81	19	d3	26
e5	9a	7a	4c

MixColumns中的有限域

给定一个素数p, 元素个数为p的有限域GF(p)定义为整数{0,1,2,...,p-1}的集合Zp,其运算为mod p的算术运算。

最简单的有限域是GF(2), 该有限域的元素的个数为2, 它们分别是0和1, 在GF(2)上的加法运算等价于异或运算, 乘法运算等价于二进制与运算。

更多资料可以参考白老师的http://10.71.45.100/bhh/crypto.doc

MixColumns中的有限域

有限域GF(p^n)

(1)对于F[x]中的每个不可约多项式p(x), 可以构造一个域$F[x]_{p(x)}$;

(2)设p(x)是F[x]中n次不可约多项式, 令$F[x]_{p(x)}$为F[x]中所有次数小于n的多项式集合, 即:

$F[x]_{p(x)} = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1 x + a_0$

其中ai∈F, 即在集合{0,1,2,…,p-1}上取值, 则我们可以定义$F[x]_{p(x)}$上的加法运算及乘法运算如下:

a(x)+b(x) = (a(x)+b(x)) mod p(x)

a(x)*b(x) = (a(x)*b(x)) mod p(x)

更多资料可以参考白老师的http://10.71.45.100/bhh/crypto.doc

MixColumns中的有限域

在GF(2^n)中, F[x]p(x)中所有次数小于n的多项式可以表示为:

f(x) = $a_{n-1}x^{n-1}+a_{n-2}x^{n-2}+ \ldots +a_1x +a_0$

其中系数ai是二进制数, 该多项式可以由它的n个二进制系数唯一表示。因此GF(2^n)中的每个多项式都可以表示成一个n位的二进制数;

AES算法选择用多项式表示有限域GF(2^8)中的元素, 其中不可约多项式p(x)为

x^8+x^4+x^3+x+1, 该多项式对应的二进制数为1 0001 1011, 对应的十六进制数为0x11B。

AES算法的MixColumn步骤中对两个8位数做乘法运算时必须mod 0x11B。

更多资料可以参考白老师的http://10.71.45.100/bhh/crypto.doc

MixColumns中的有限域

计算8位数乘法 mod 0x11B

A = 0b10001000, B=0b00000101

A*B = (x^7 + x^3) * (x^2 + 1) = x^9 + x^7 + x^5 + x^3 =

x^7 + x^4 + x^3 + x^2 + x mod (x^8 + x^4 + x^3 + x + 1)

x^9 + x^7 + x^5 + x^3 = (x^8 + x^4 + x^3 + x + 1) * x + (x^7 - x^4 + x^3 - x^2 - x)

更多资料可以参考白老师的http://10.71.45.100/bhh/crypto.doc

AES解密

- 密钥扩展（相同）
- 逆轮密钥加，Rev_AddRoundKey（相同）
- 逆字节替换，Rev_SubBytes（不同）
- 逆行移位，Rev_ShiftRows（不同）
- 逆列混淆，Rev_MixColumns（不同）

AES解密

逆字节替换，Rev_SubBytes

x \ y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
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

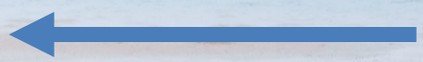
(b) Inverse S-box

AES解密

逆行移位，Rev_ShiftRows

第一行保持不变，第二行循环右移1字节，第三行右移2字节，第四行右移3字节。

0	1	2	3
4	5	6	7
8	9	a	b
c	d	e	f



0	1	2	3
5	6	7	4
a	b	8	8
f	c	d	e

AES解密

逆列混淆，Rev_MixColumns

同样是左乘一个矩阵

0E	0B	0D	09
09	0E	0B	0D
0D	09	0E	0B
0B	0D	09	0E

X

04	e0	48	28
66	cb	f8	06
81	19	d3	26
e5	9a	7a	4c

=

d4	e0	b8	1e
bf	b4	41	27
5d	52	11	98
30	ae	f1	e5

Python AES

```python
from Crypto.Cipher import AES

k = b'0123456789abcdef'
plain = b'[this is a test]'

aes = AES.new(key=k, mode=AES.MODE_ECB)

print(aes.encrypt(plain))
```

➜ python3 aes_test.py

b'&\xe0\xde\xd2;Ib8\xf4?\x87\x91\xcf\xe9\x8c/'

分组密码操作模式

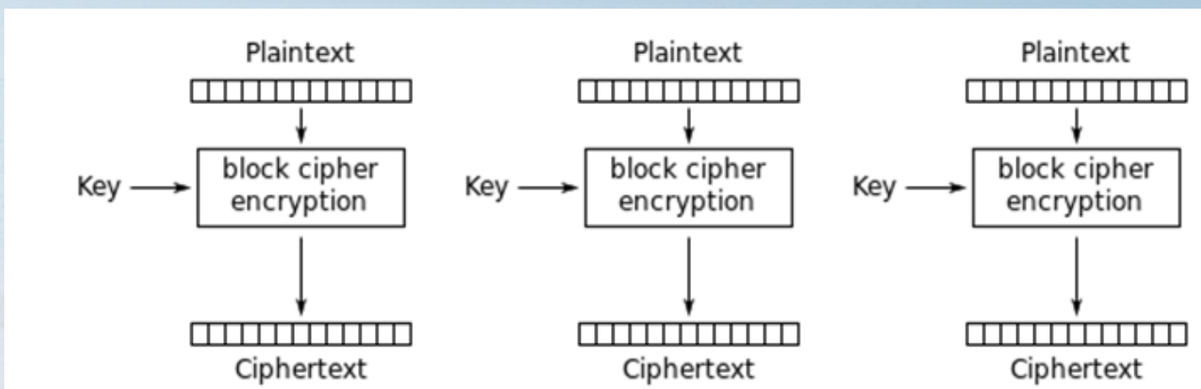
当明文长度超过加密所支持的长度时，需要对明文进行分组。

常见的分组密码操作模式有：电子密码本模式（ECB）、密码分组链接模式（CBC）、密码反馈模式（CFB）、输出反馈模式（OFB）、计数器模式（CTR）。

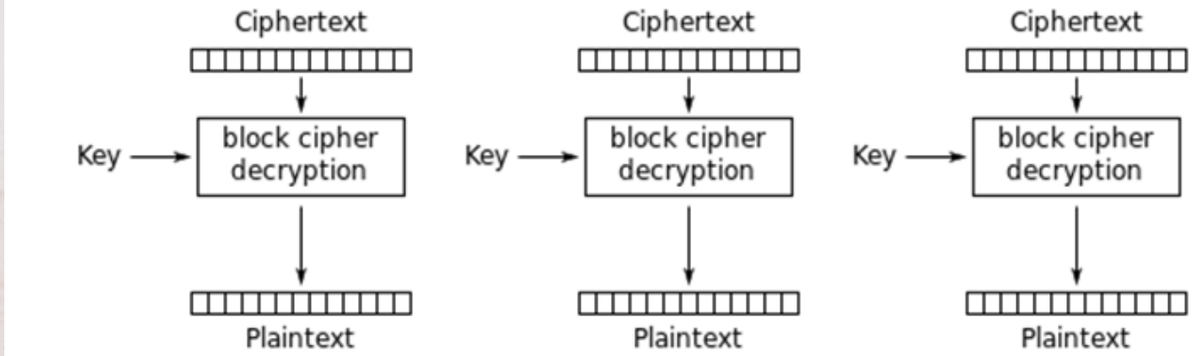
电子密码本模式（ECB）

ECB就是使用相同的密钥对明文组进行加密，一次只加密一组明文。解密时也使用相同的密钥对密文组进行解密，一次解密一组密文。尾部明文不够一组时需要填充。

例如，明文长度为32字节时，则根据DES可以分为4组，AES则为2组。



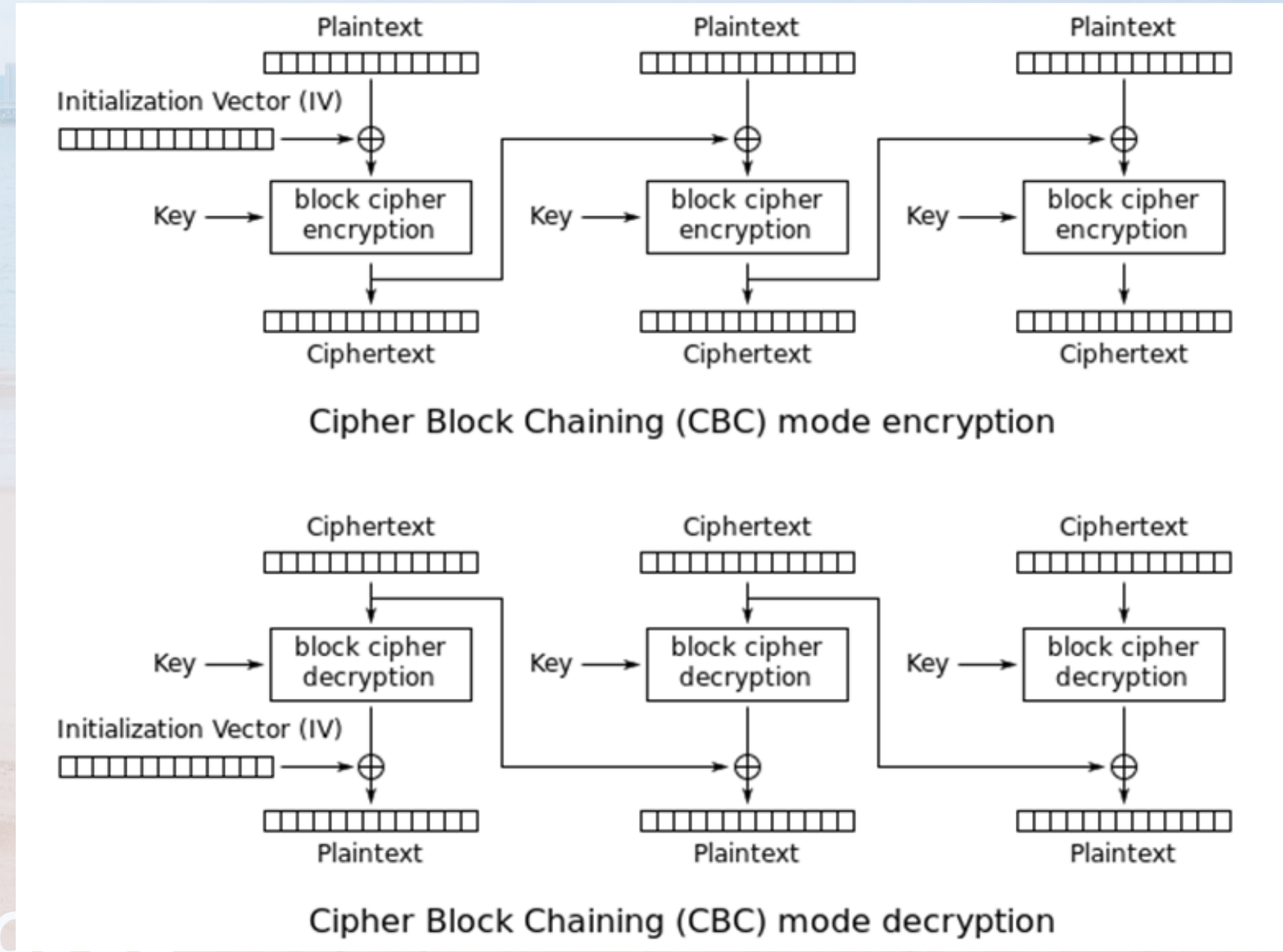
Electronic Codebook (ECB) mode encryption



Electronic Codebook (ECB) mode decryption

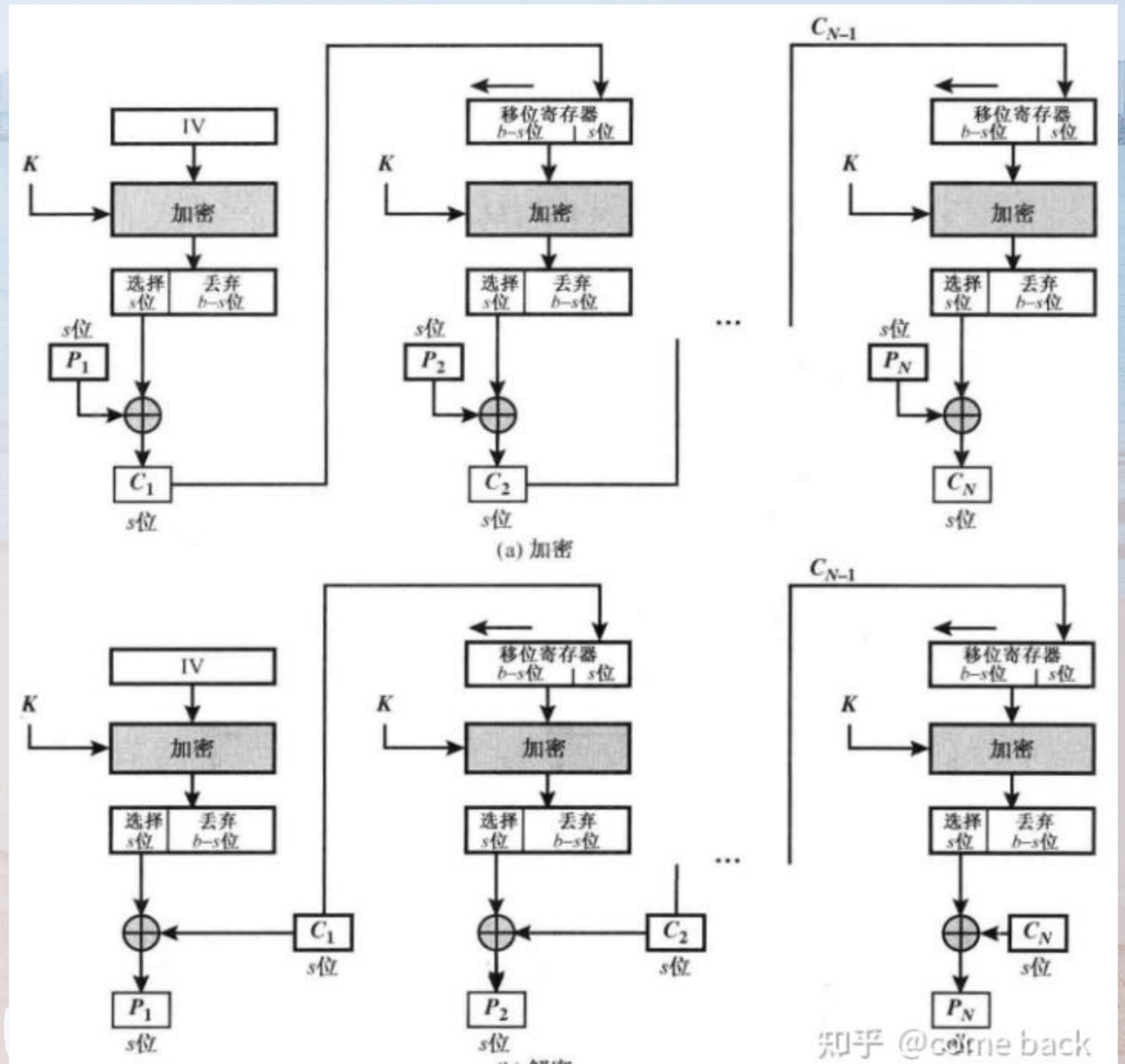
密码分组链接模式（CBC）

由于ECB在安全性上存在一些缺陷，为了处理好这个问题，可以让重复的明文分组产生不同的密文分组，又设计出一种CBC模式，它可满足这一要求。



密码反馈模式（CFB）

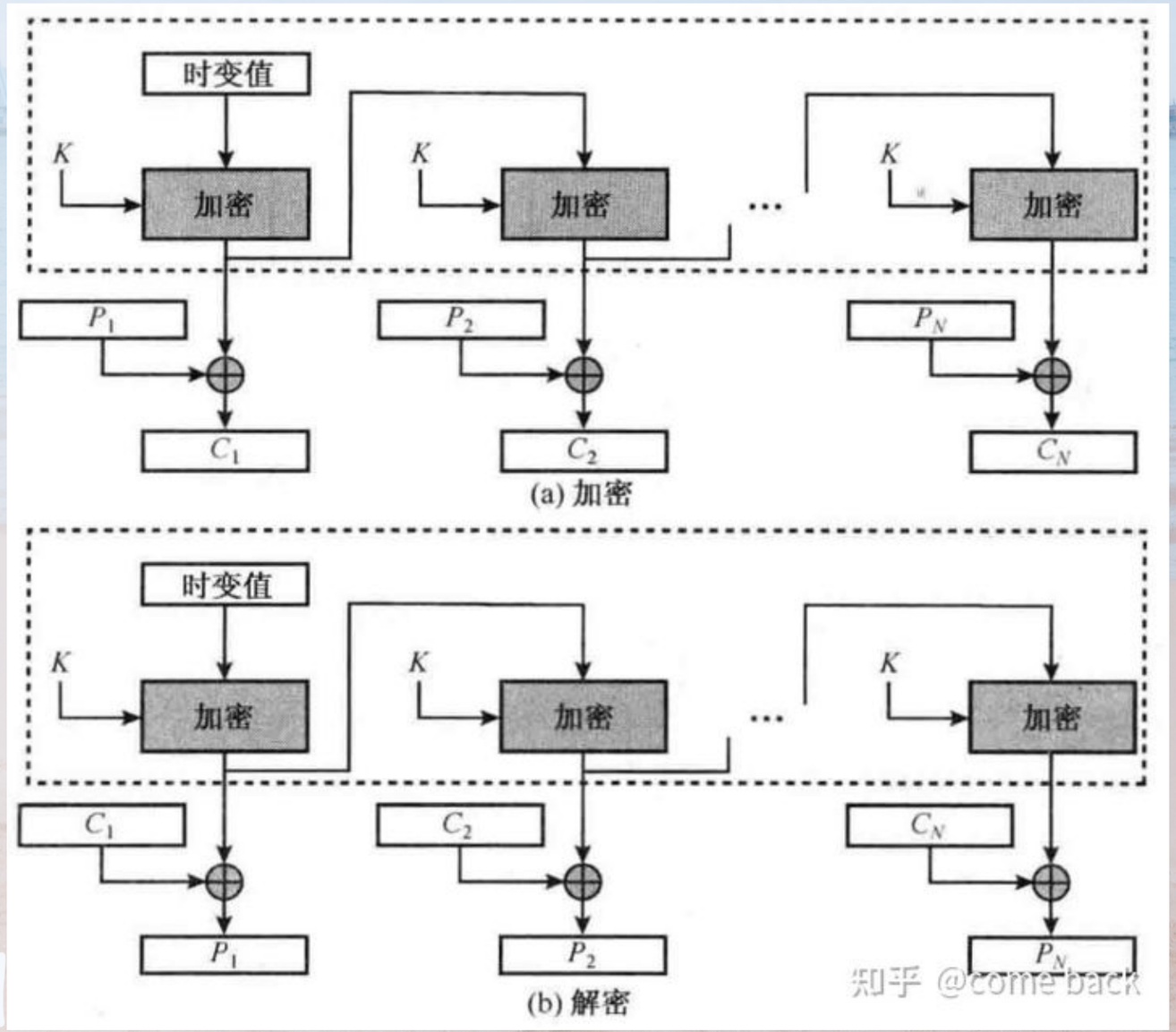
与ECB、CBC不同，在CFB（Cipher Feedback，CFB）模式下，明文本身并没有进入加密算法中进行加密，而是与加密函数的输出进行了异或，得到了密文。



输出反馈模式（OFB）

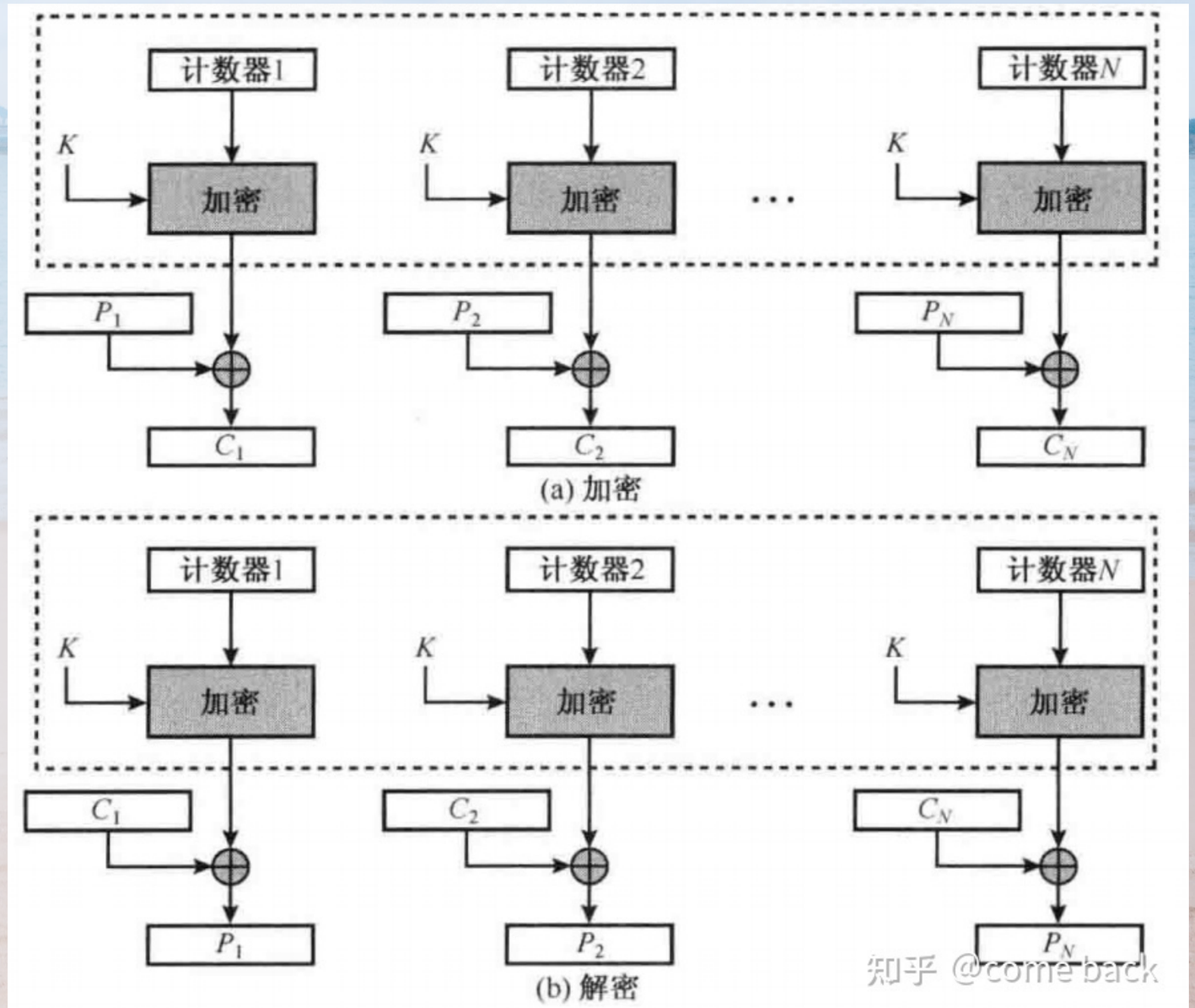
与CFB类似，输出反馈模式（Output Feedback，OFB）也是一种类似于流密码的工作模式。

不同的是，在OFB中，是直接将加密算法的输出反馈到下一分组加密算法的输入中。



计数器模式（CTR）

计数器模式（counter，CTR）也是一种类似于流密码的模式。加密算法只是用来产生密钥流与明文分组异或。



填充模式

当明文长度最后部分无法构成一个完整的块时，需要进行填充，两种最典型的填充模式是ZeroPadding，PKCS7Padding

- ZeroPadding，直接使用0进行填充到对齐 ’this is a test’ -> ’this is a test\x00\x00’
- PKCS7Padding，假设数据长度需要填充n(n>0)个字节才对齐，那么填充n个字节，每个字节都是n，’this is a test’ -> ’this is a test\x02\x02’

攻击手法

- ECB模式下的Orcale
- CBC模式下的字节翻转
- Padding Attack
- Padding Orcale

ECB模式下的Orcale

题目：ecb_test.py

要求窃取SECRET

核心逻辑如下：

```python
def encrypt_with_secret():
    msg = unhexlify(input("your msg: "))
    assert len(msg) <= 48
    return hexlify(encrypt(msg + SECRET)).decode()
```

ECB模式下的Orcale

Round 1

msg	secert

msg (15*A)	secert

15*A+secret[0] → cipher block0

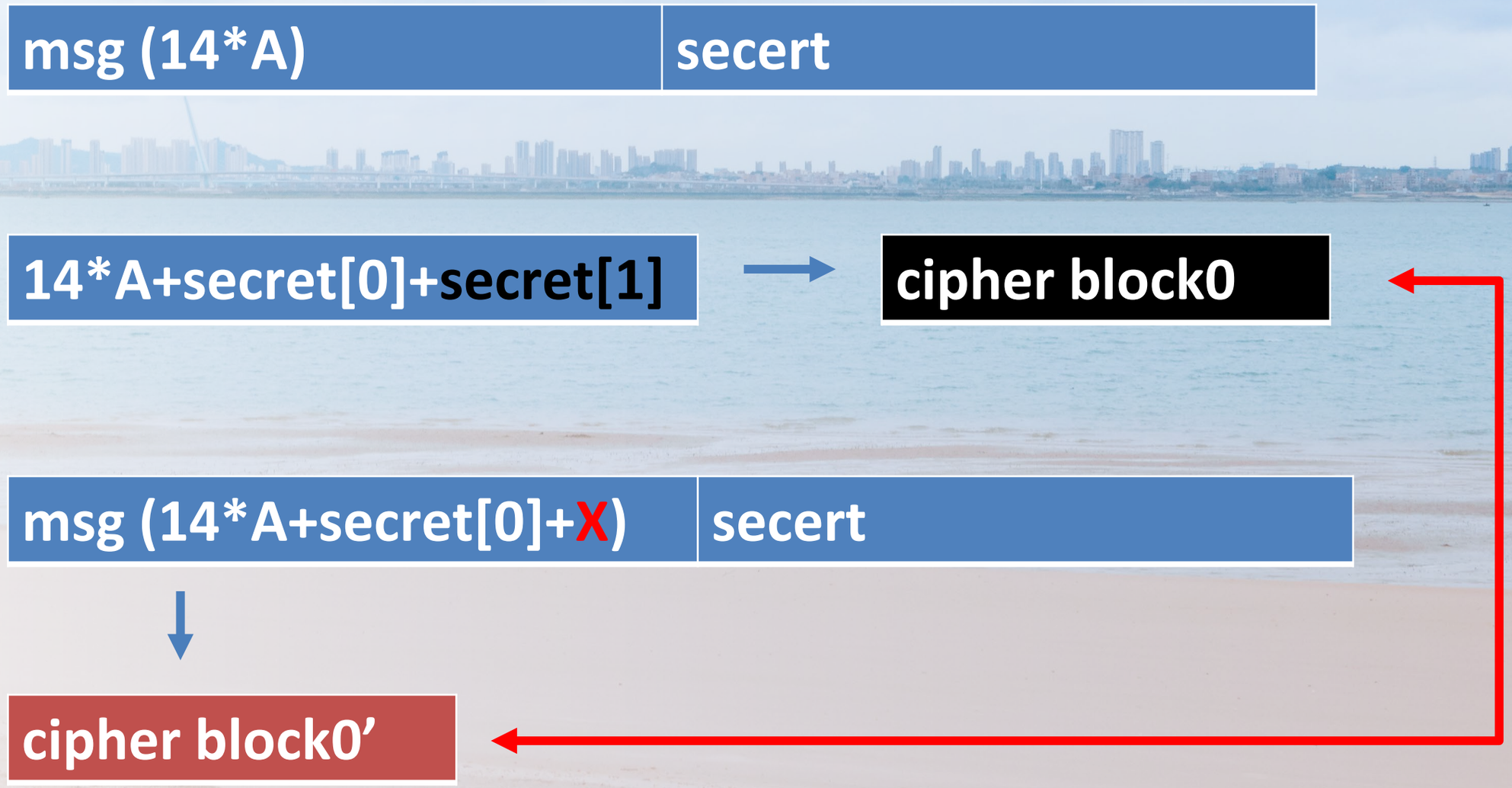
msg (15*A+X)	secert

↓

cipher block0’

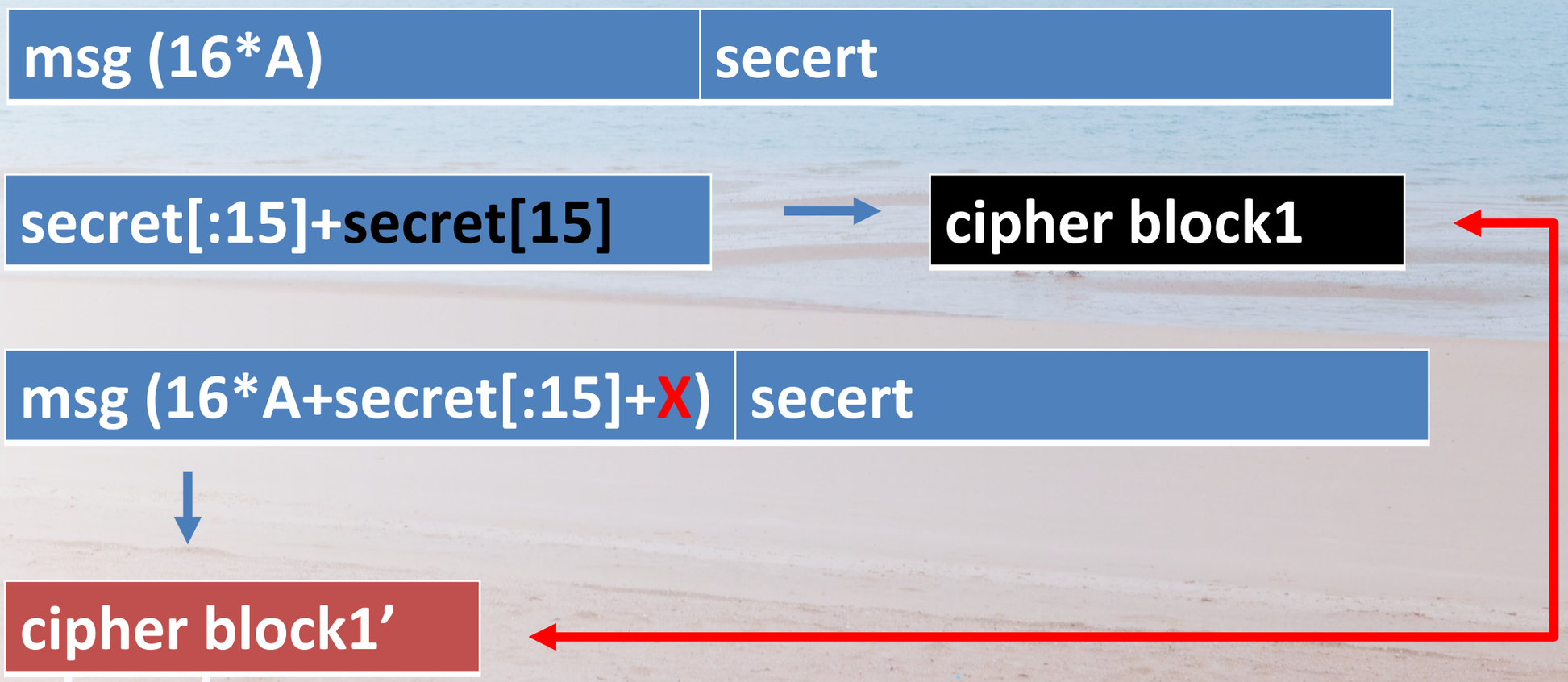
ECB模式下的Orcale

Round 2

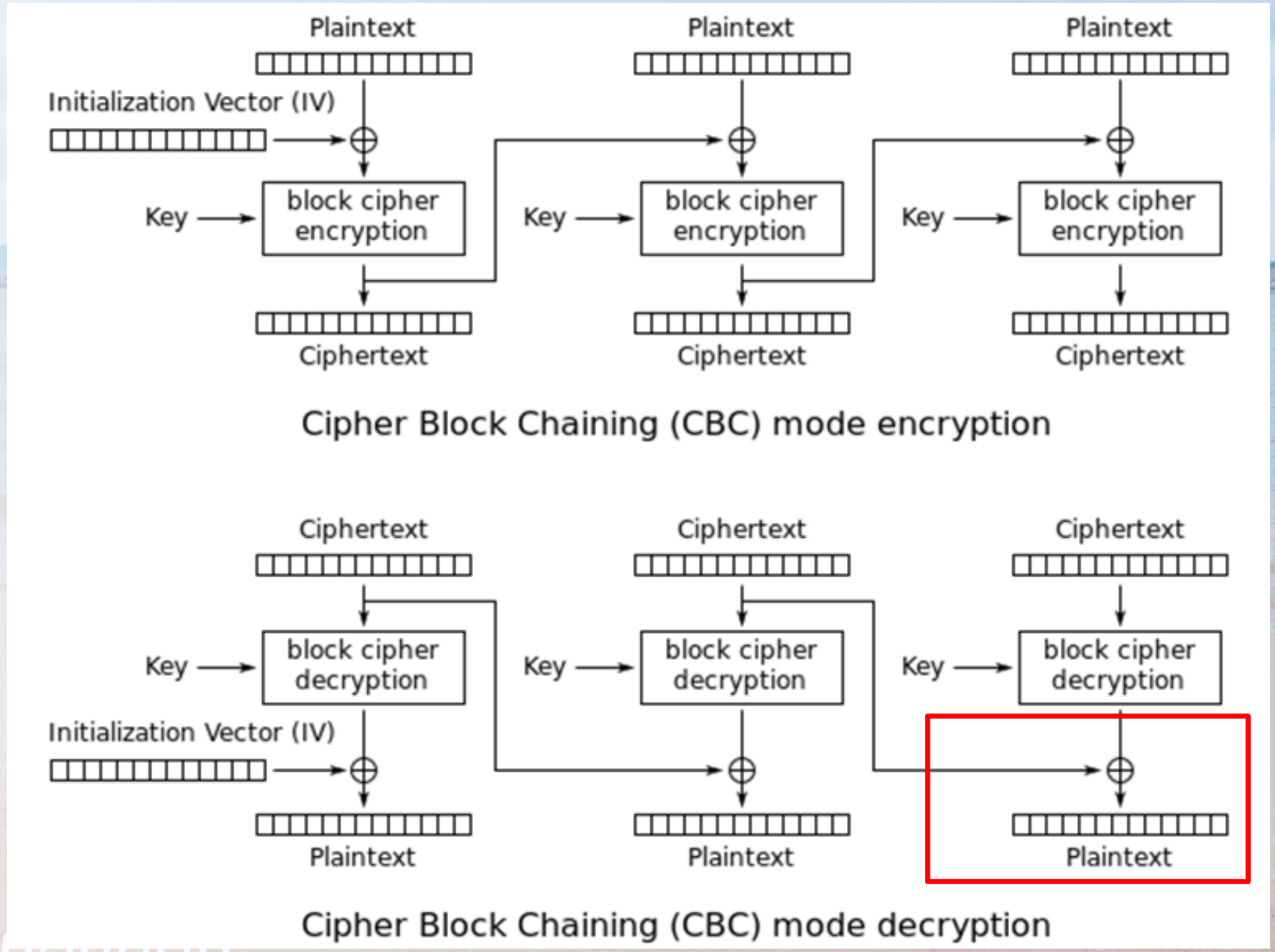


ECB模式下的Orcale

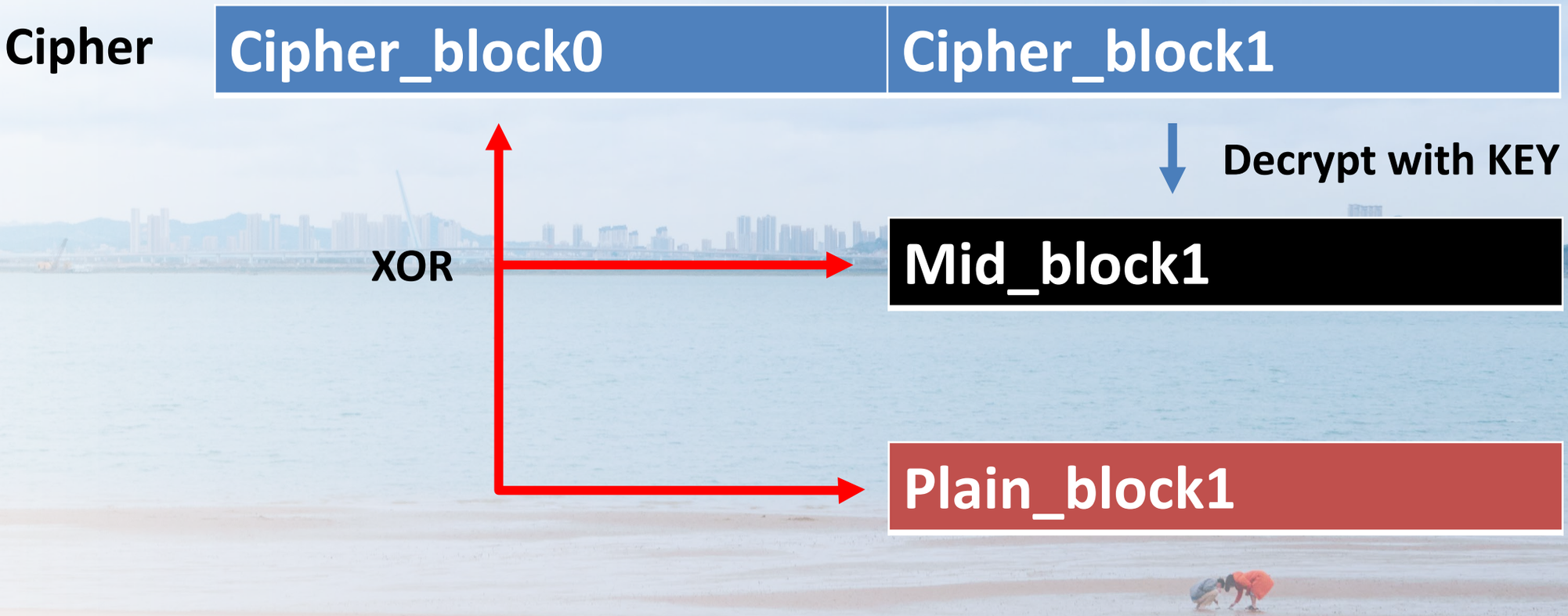
Round 16 (Current we know SECRET[:15])



CBC模式下的字节翻转



CBC模式下的字节翻转



What will happen if we can control the Cipher?

Construct arbitrary plaintext!!!

Padding Attack

由于加密前首先会对消息进行pad，因此在解密时需要进行unpad操作。但某些情形下，unpad的过程不会检查pad部分的合法性。导致非预期的行为。

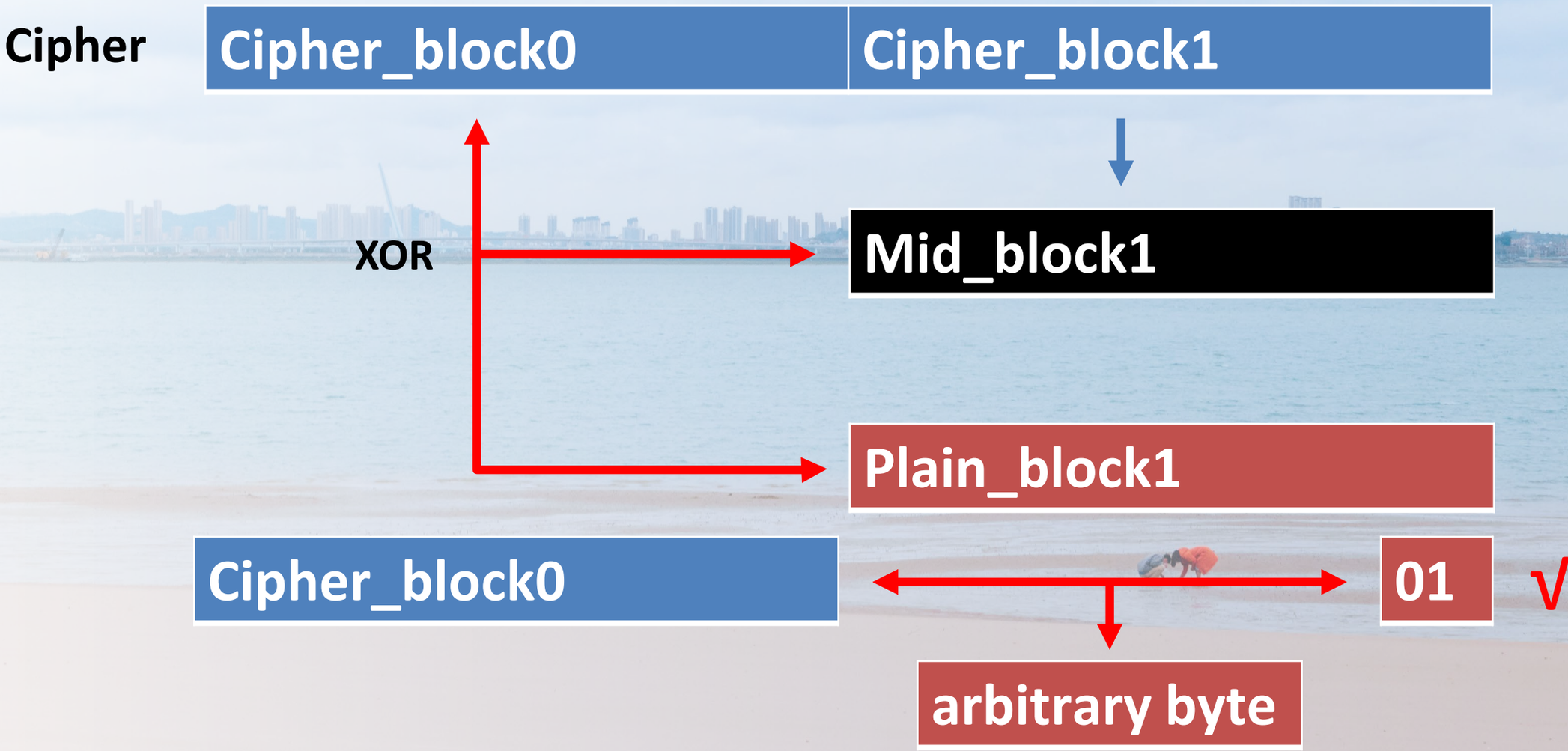
```python
def unpad(msg: bytes):
    # if bytes([msg[-1]]) * msg[-1] != msg[-msg[-1]:]:
    #     return b"padding error!?"
    return msg[:-msg[-1]]
```

Padding Orcale

现在，假设unpad的过程确实检测了pad的部分是否合法。这本身是一件好事，但是有些情形下将会泄露信息！

```python
def unpad(msg: bytes):
    if bytes([msg[-1]]) * msg[-1] != msg[-msg[-1]:]:
        return b"padding error!?"
    return msg[:-msg[-1]]
```

Padding Orcale



Another arbitrary plaintext construction!!!

ACTF2022 retros!!!

密码分析技术

- 线性分析（Linear Analysis）
- 差分分析（Differential Analysis）
- 滑动攻击（Slide Attack）

线性分析（Linear Analysis）

从S盒说起

一切都得从S盒说起，S盒是一个代换表，输入X，根据代换表，输出Y，则：

- 已知S盒，已知X，可解出Y
- 已知S盒，已知Y，可解出X

但假如我只知道输入X（输出Y）的某几个比特，在已知S盒的前提下，我们有可能得到什么信息呢？

答：可能知道输出Y（输入X）的某几个比特间的概率关系！

线性分析（Linear Analysis）

考虑如下S盒

input	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
output	E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7

线性分析（Linear Analysis）

X1	X2	X3	X4	Y1	Y2	Y3	Y4
0	0	0	0	1	1	1	0
0	0	0	1	0	1	0	0
0	0	1	0	1	1	0	1
0	0	1	1	0	0	0	1
0	1	0	0	0	0	1	0
0	1	0	1	1	1	1	1
0	1	1	0	1	0	1	1
0	1	1	1	1	0	0	0
1	0	0	0	0	0	1	1
1	0	0	1	1	0	1	0
1	0	1	0	0	1	1	0
1	0	1	1	1	1	0	0
1	1	0	0	0	1	0	1
1	1	0	1	1	0	0	1
1	1	1	0	0	0	0	0
1	1	1	1	0	1	1	1

线性分析（Linear Analysis）

X1	X2	X3	X4	Y1	Y2	Y3	Y4	Res
0	0	0	0	1	1	1	0	1
0	0	0	1	0	1	0	0	0
0	0	1	0	1	1	0	1	1
0	0	1	1	0	0	0	1	1
0	1	0	0	0	0	1	0	0
0	1	0	1	1	1	1	1	0
0	1	1	0	1	0	1	1	0
0	1	1	1	1	0	0	0	1
1	0	0	0	0	0	1	1	1
1	0	0	1	1	0	1	0	0
1	0	1	0	0	1	1	0	0
1	0	1	1	1	1	0	0	1
1	1	0	0	0	1	0	1	0
1	1	0	1	1	0	0	1	0
1	1	1	0	0	0	0	0	1
1	1	1	1	0	1	1	1	1

P(X1+X4+Y2==0)

= 1/2

P(X1+X4+Y2 ==1)

= 1/2

线性分析（Linear Analysis）

X1	X2	X3	X4	Y1	Y2	Y3	Y4	Res
0	0	0	0	1	1	1	0	1
0	0	0	1	0	1	0	0	1
0	0	1	0	1	1	0	1	1
0	0	1	1	0	0	0	1	1
0	1	0	0	0	0	1	0	0
0	1	0	1	1	1	1	1	1
0	1	1	0	1	0	1	1	1
0	1	1	1	1	0	0	0	1
1	0	0	0	0	0	1	1	1
1	0	0	1	1	0	1	0	0
1	0	1	0	0	1	1	0	1
1	0	1	1	1	1	0	0	1
1	1	0	0	0	1	0	1	1
1	1	0	1	1	0	0	1	1
1	1	1	0	0	0	0	0	1
1	1	1	1	0	1	1	1	1

P(X3+X4+Y1+Y4==0)

= 2/16

P(X3+X4+Y1+Y4 ==1)

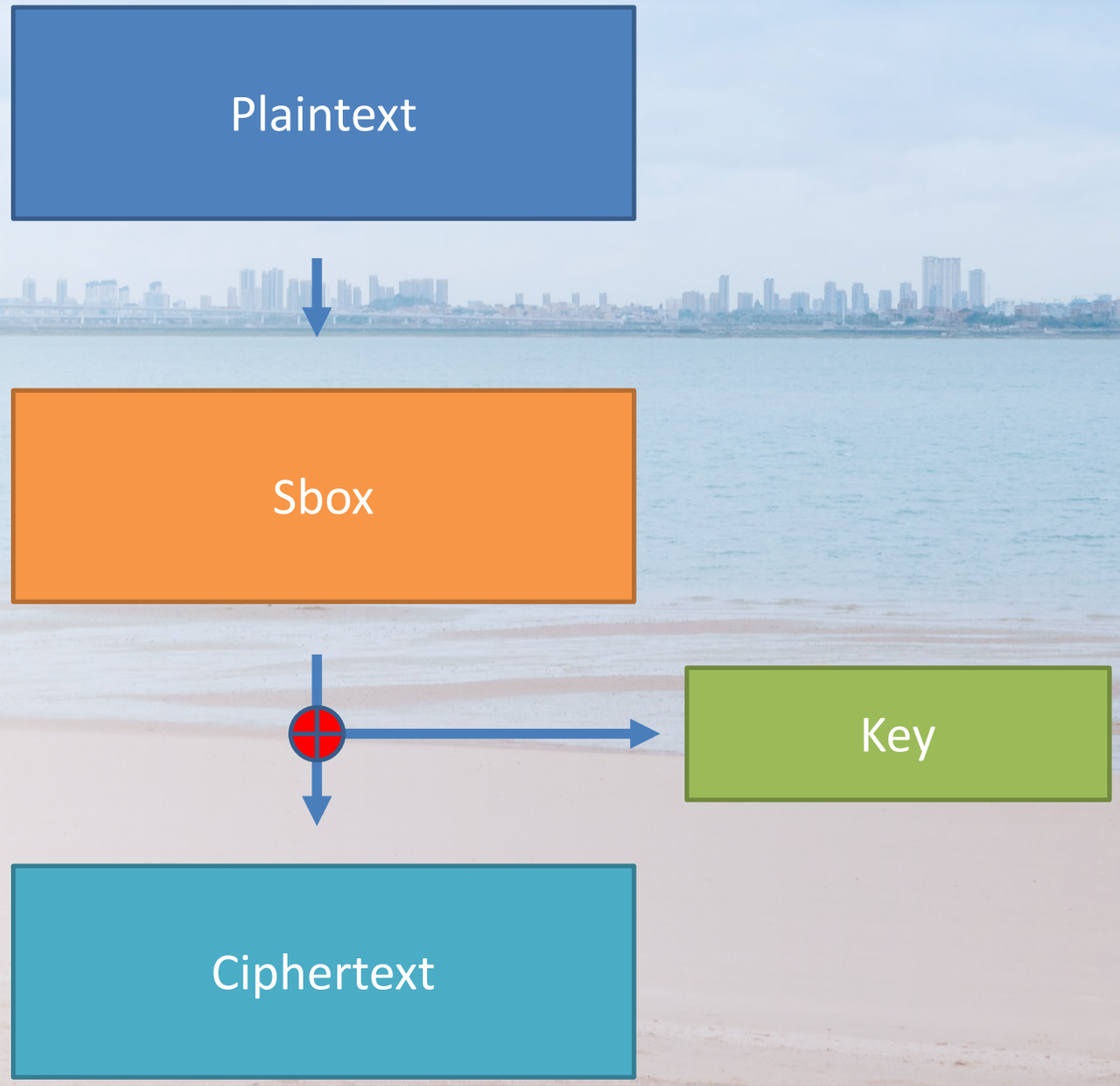
= 14/16

线性分析（Linear Analysis）

这种输入的某些比特和输出的某些比特异或的表达式，是一个只有0和1的异或表达式，结果也就是0和1两种情况，在没有任何信息可以利用时，输入是随机的，这个表达式等于0或1的概率应该均为1/2。

如果一个S盒设计有缺陷，则经过S盒变换后的输出和输入是有一定的概率关系的。所以，当知道输入X（输出Y ）的某几个比特，在已知S盒的前提下，就可能知道输出Y（输入X）的某几个比特间的概率关系！但是单独一个S盒，看起来好像并没有什么卵用。

线性分析（Linear Analysis）



线性分析（Linear Analysis）

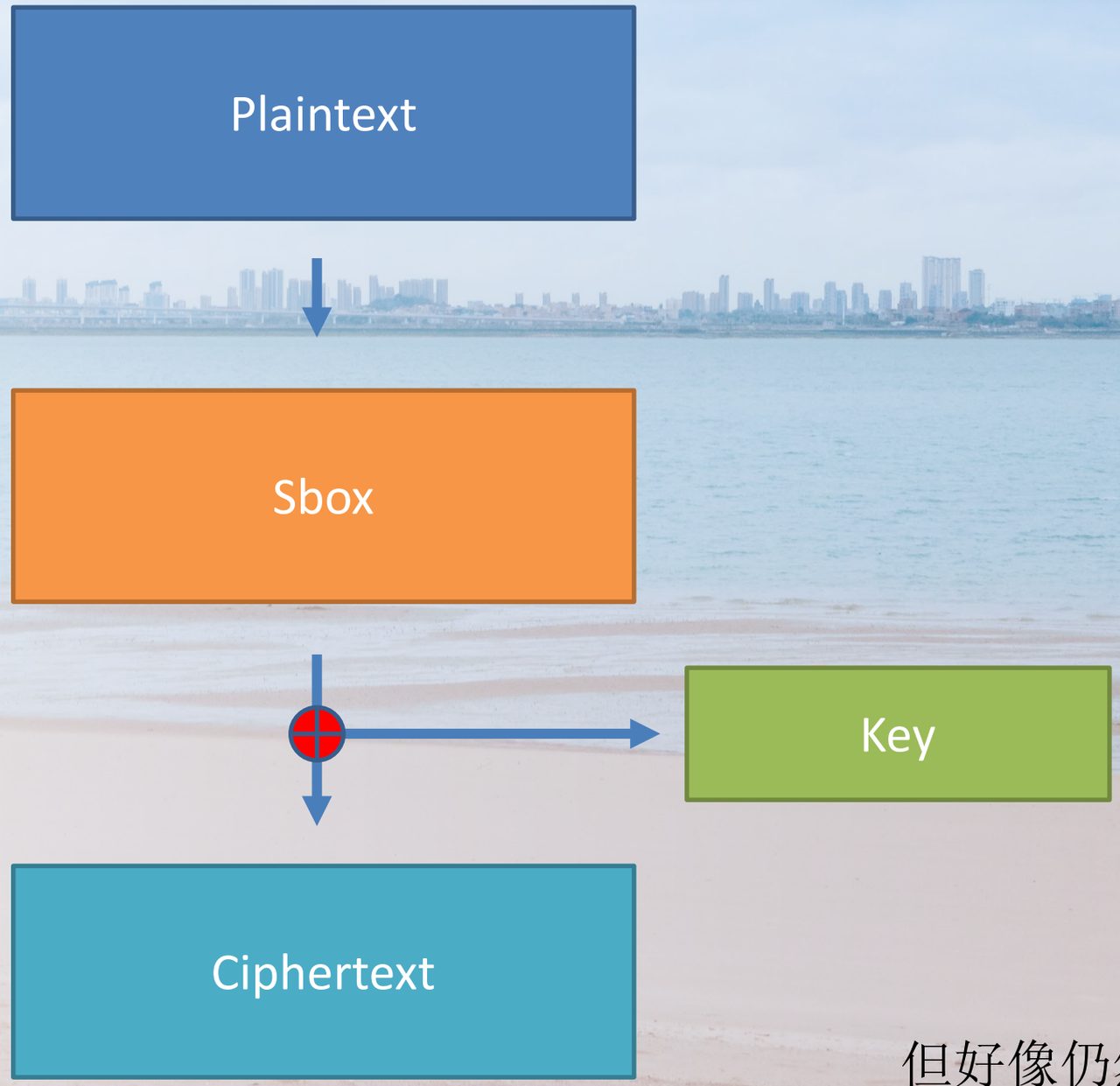
X1	X2	X3	X4	Y1	Y2	Y3	Y4	Res
0	0	0	0	1	1	1	0	1
0	0	0	1	0	1	0	0	1
0	0	1	0	1	1	0	1	1
0	0	1	1	0	0	0	1	1
0	1	0	0	0	0	1	0	0
0	1	0	1	1	1	1	1	1
0	1	1	0	1	0	1	1	1
0	1	1	1	1	0	0	0	1
1	0	0	0	0	0	1	1	1
1	0	0	1	1	0	1	0	0
1	0	1	0	0	1	1	0	1
1	0	1	1	1	1	0	0	1
1	1	0	0	0	1	0	1	1
1	1	0	1	1	0	0	1	1
1	1	1	0	0	0	0	0	1
1	1	1	1	0	1	1	1	1

由于输出Y进一步异或了一个Key，所以得到的结果Res会发生变化。

Cipher=Res^K1^K4

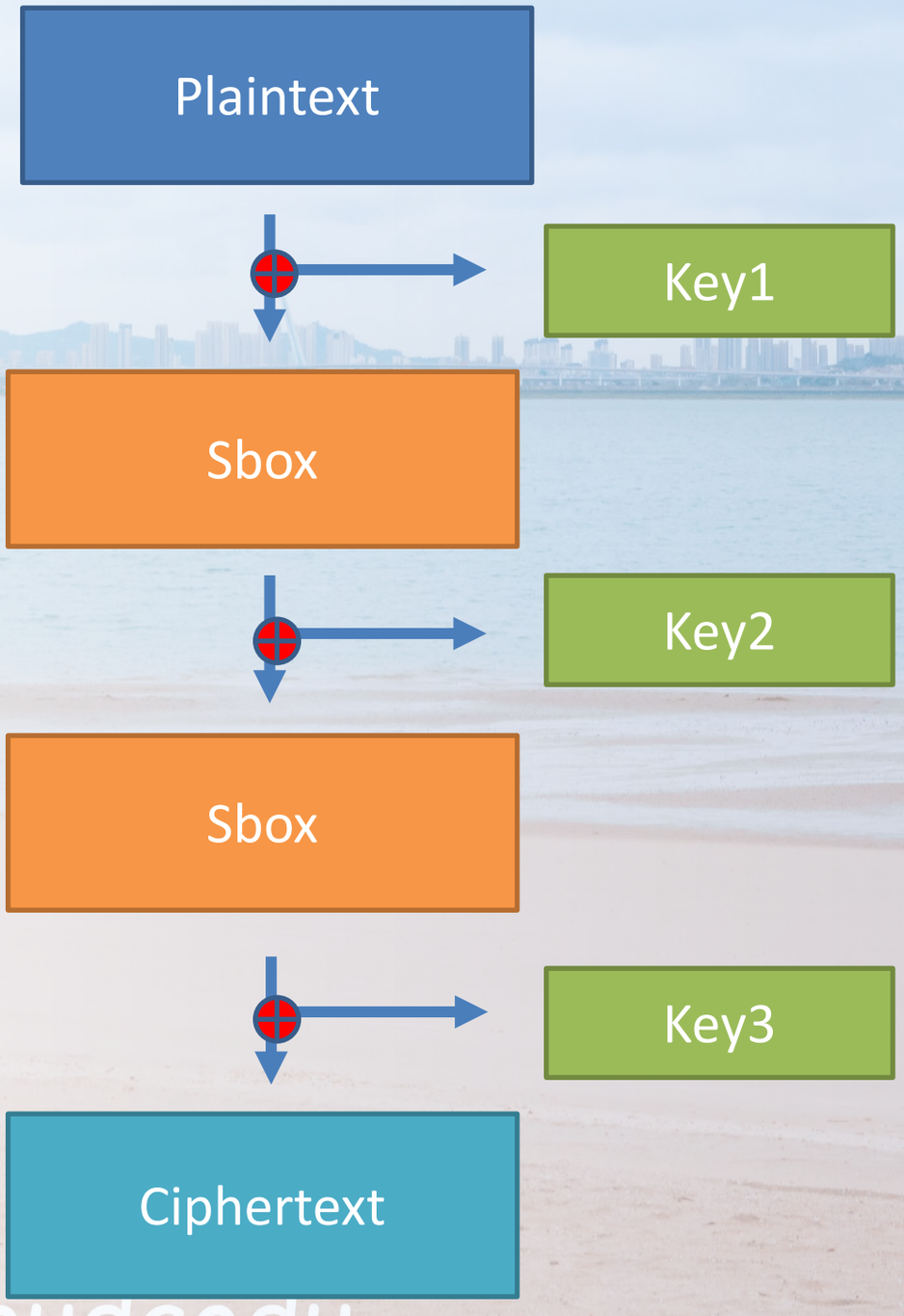
概率分布仍然满足!

线性分析（Linear Analysis）



但好像仍然没什么卵用

线性分析（Linear Analysis）



Key1/2/3分别用于每一轮异或。

目前我们有很多对明文以及对应的密文，如何破解Key？

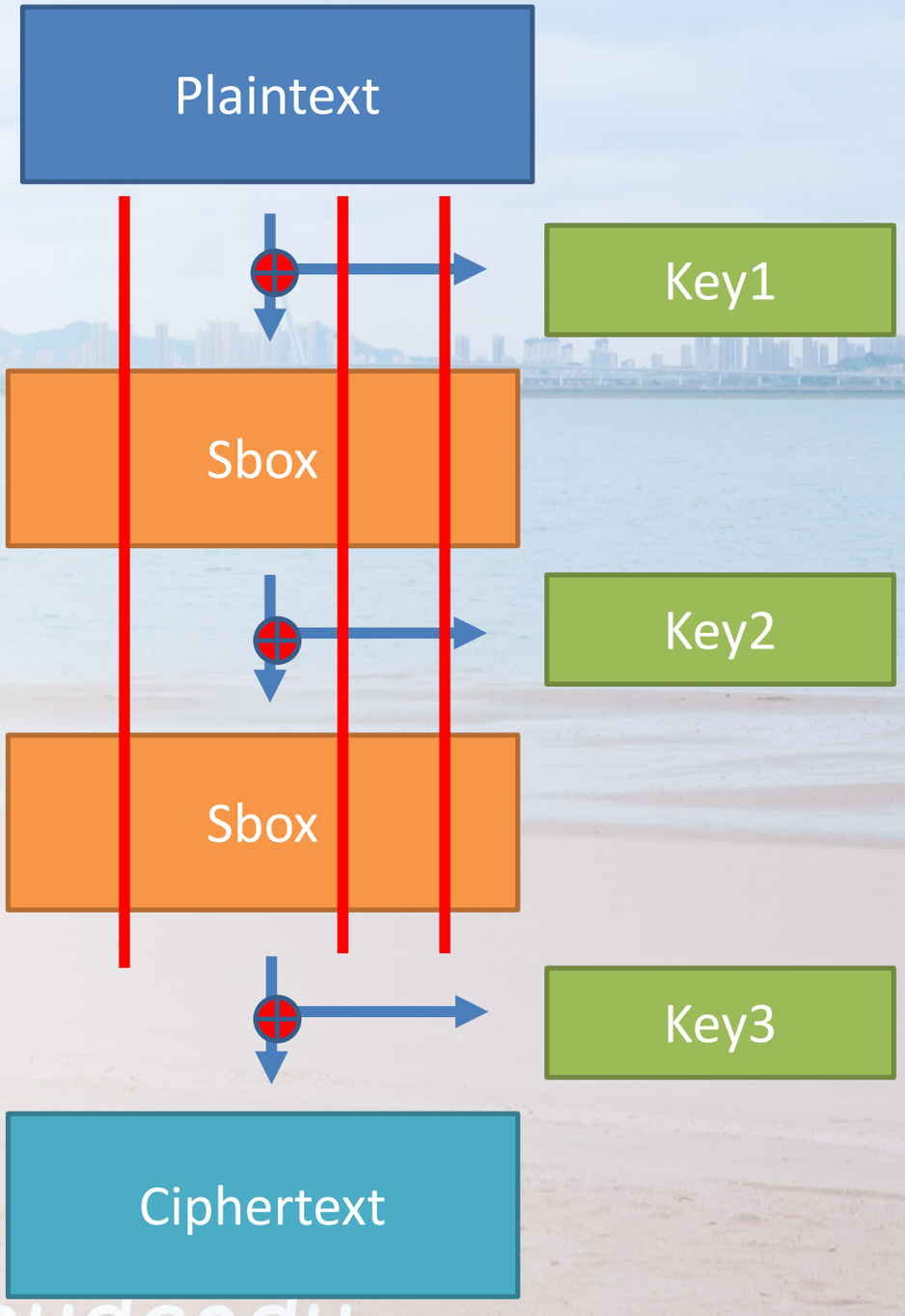
线性分析（Linear Analysis）

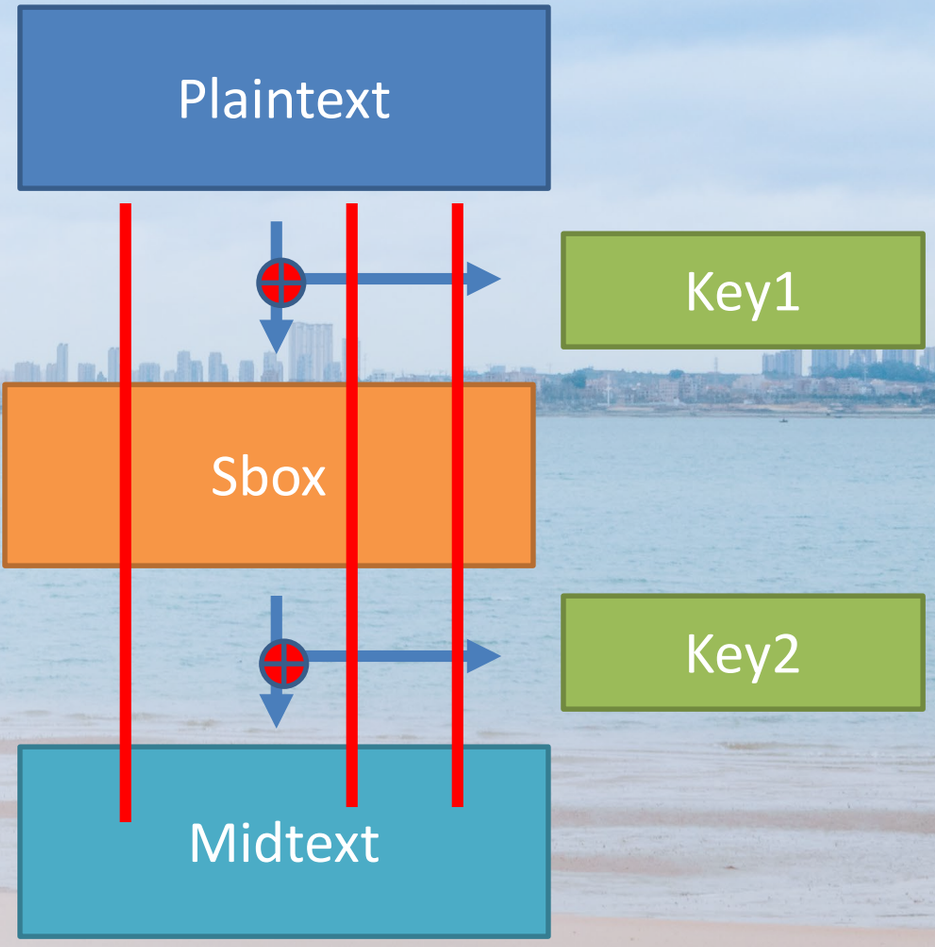
Substitution Box for Toy Cipher	
0 -> 9	8-> 13
1 -> 11	9 -> 7
2 -> 12	10 -> 3
3 -> 4	11 -> 8
4 -> 10	12 -> 15
5 -> 1	13 -> 14
6 -> 2	14 -> 0
7 -> 6	15 -> 5

P(X1+X3+X4+Y1+Y3+Y4==0)

=14/16

线性分析（Linear Analysis）





P1^P3^P4^M1^M3^M4

线性分析（Linear Analysis）

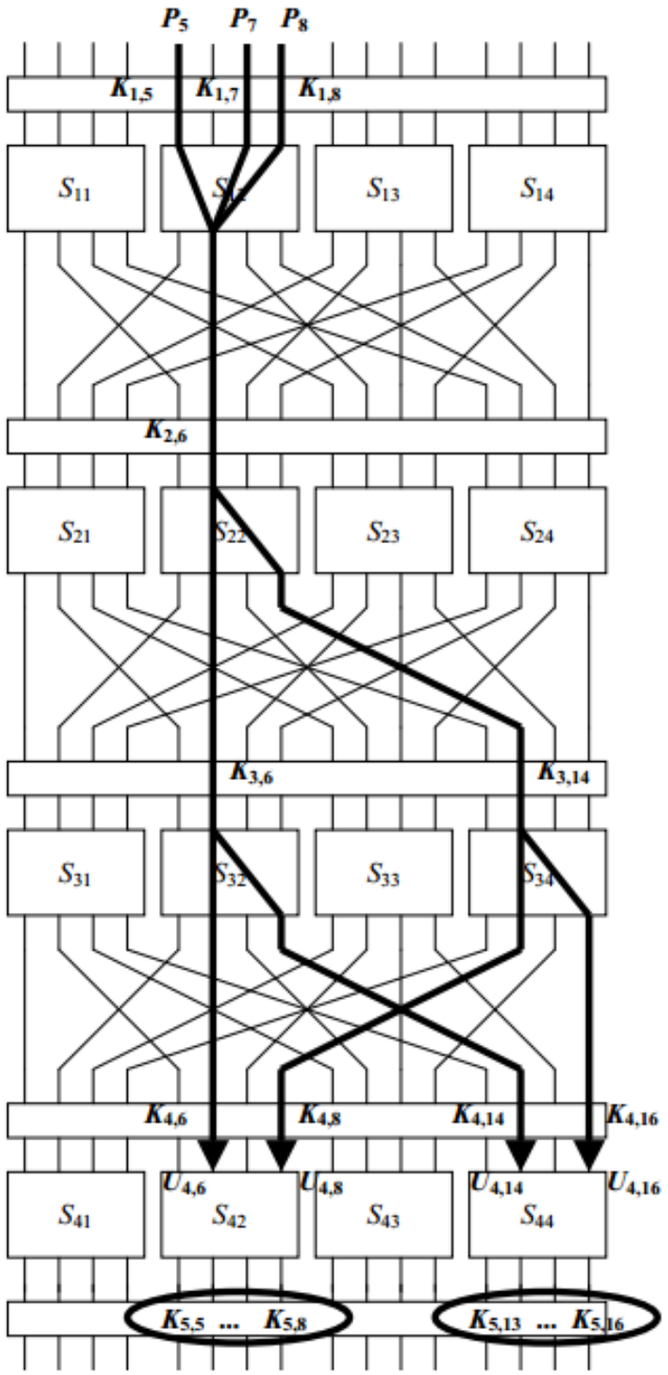
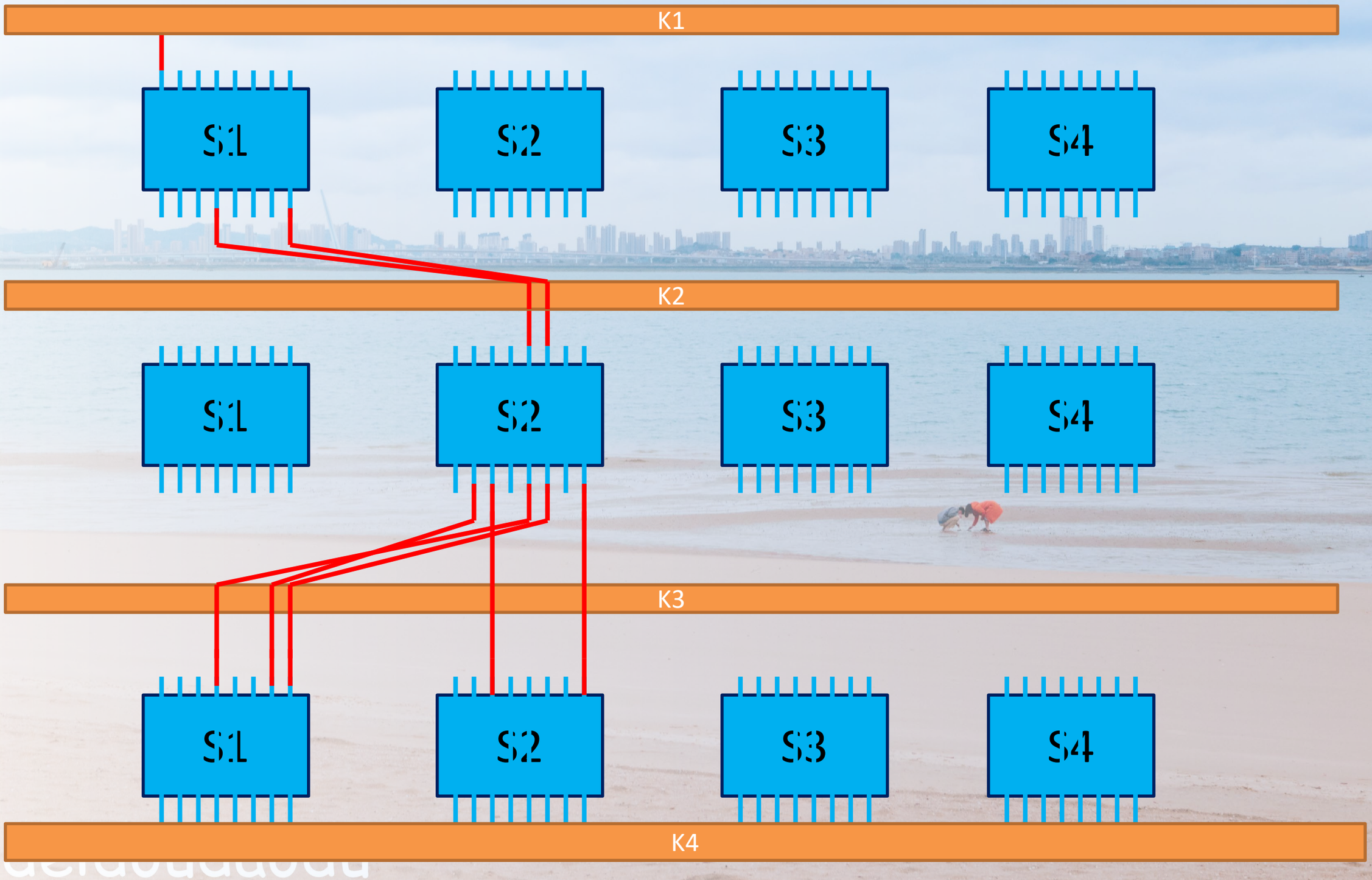


Figure 3. Sample Linear Approximation

线性分析（Linear Analysis-ASPN）



差分分析（Differential Analysis）

根据比较明文与密文之间的差异，从而基于这些差异，对密钥进行合理的猜测。

假设一个简单的加密算法，仅仅是异或一个密钥Key

针对输入P1，得到P1^Key

针对输入P2，得到P2^Key

明文的差分值为P1^P2，输出的差分值为P1^Key^P1^Key=P1^P2

因此异或操作不会改变差分特性

差分分析（Differential Analysis）

但是Sbox会消除差分特性，因为发生了随机变换。但是，Sbox并不完美，有时会保留部分差分特性，这使得密码攻击的开销大大减小。

S-Box

0	-->	3	8	-->	8
1	-->	14	9	-->	11
2	-->	1	10	-->	15
3	-->	10	11	-->	2
4	-->	4	12	-->	13
5	-->	9	13	-->	12
6	-->	5	14	-->	0
7	-->	6	15	-->	7

[0,4] -> [3,4]

[4,0] -> [4,3]

[1,5] -> [14,9]

[5,1] -> [9,14]

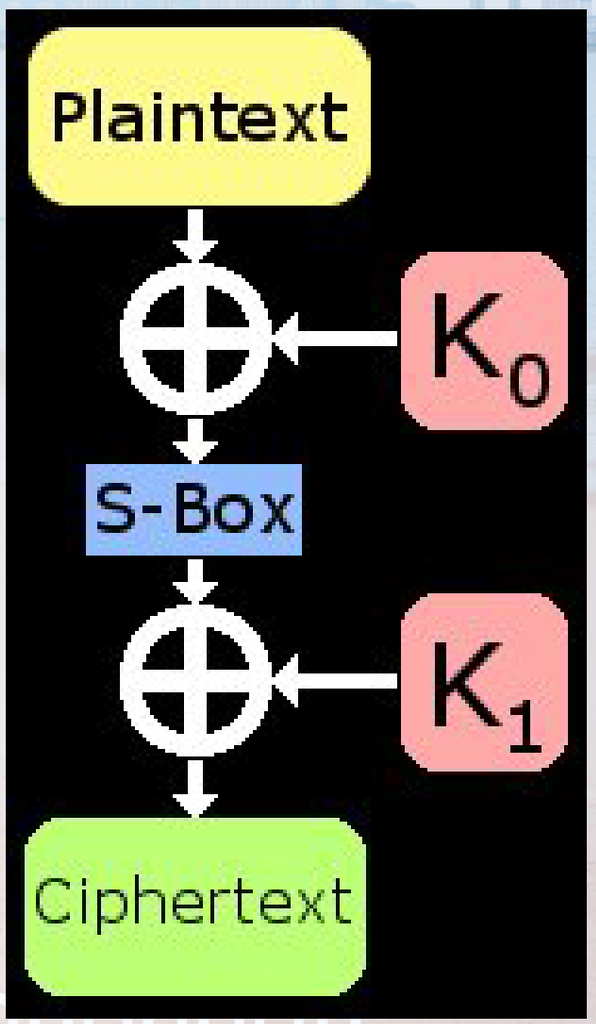
[9,13] -> [11,12]

[13,9] -> [12,11]

4 -> 7

差分分析（Differential Analysis）

输入一对明文，其差分值为4，如果输出差分值为7，说明为以下6种情况。然后随机选取其中一对，求出K0,K1。用其他明密文对进行验证。



[0,4] -> [3,4]

[4,0] -> [4,3]

[1,5] -> [14,9]

[5,1] -> [9,14]

[9,13] -> [11,12]

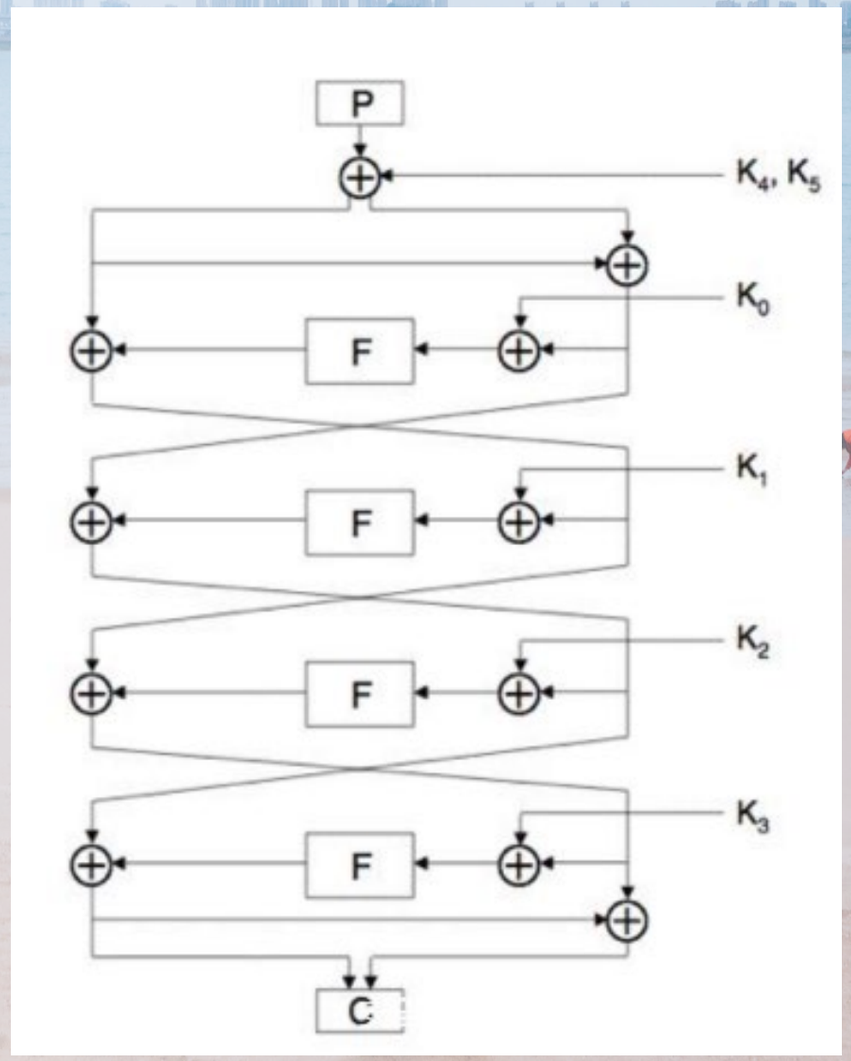
[13,9] -> [12,11]

4 -> 7

Feal-4

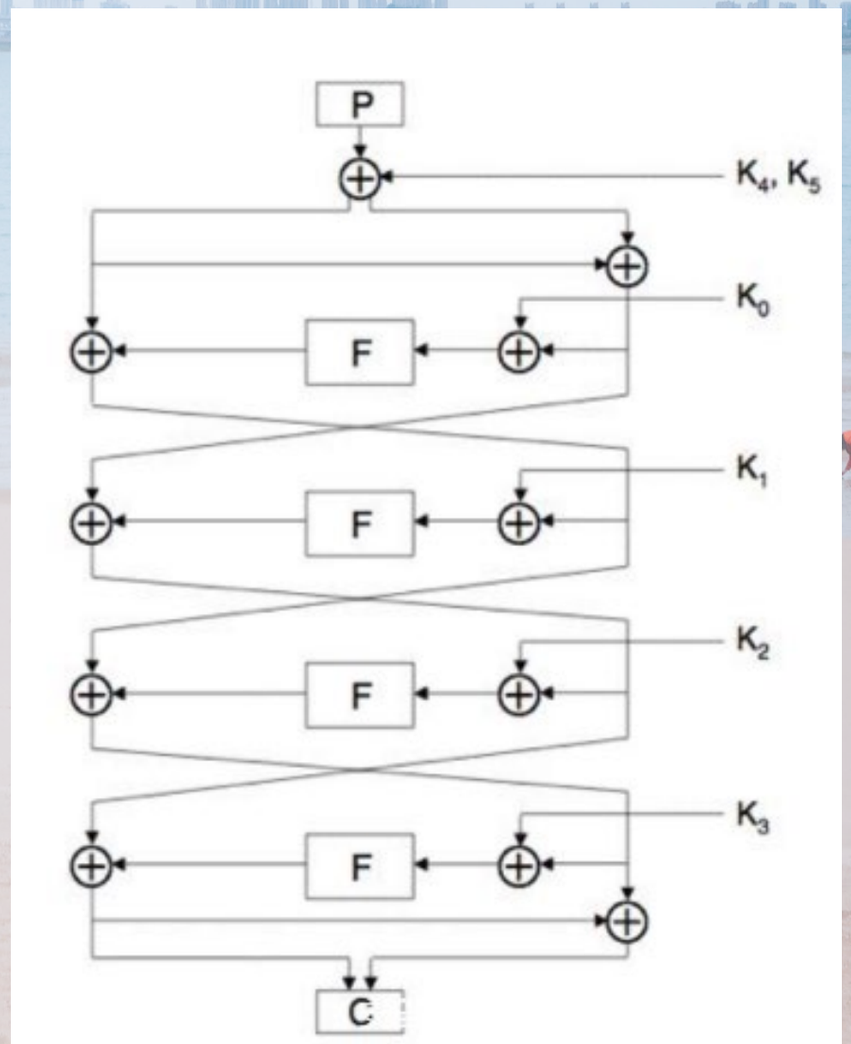
Feal-4属于分组加密算法，采用Feistel架构。密钥长度为8字节，分组长度也是8字节。

密钥进行扩展后得到K0-K5



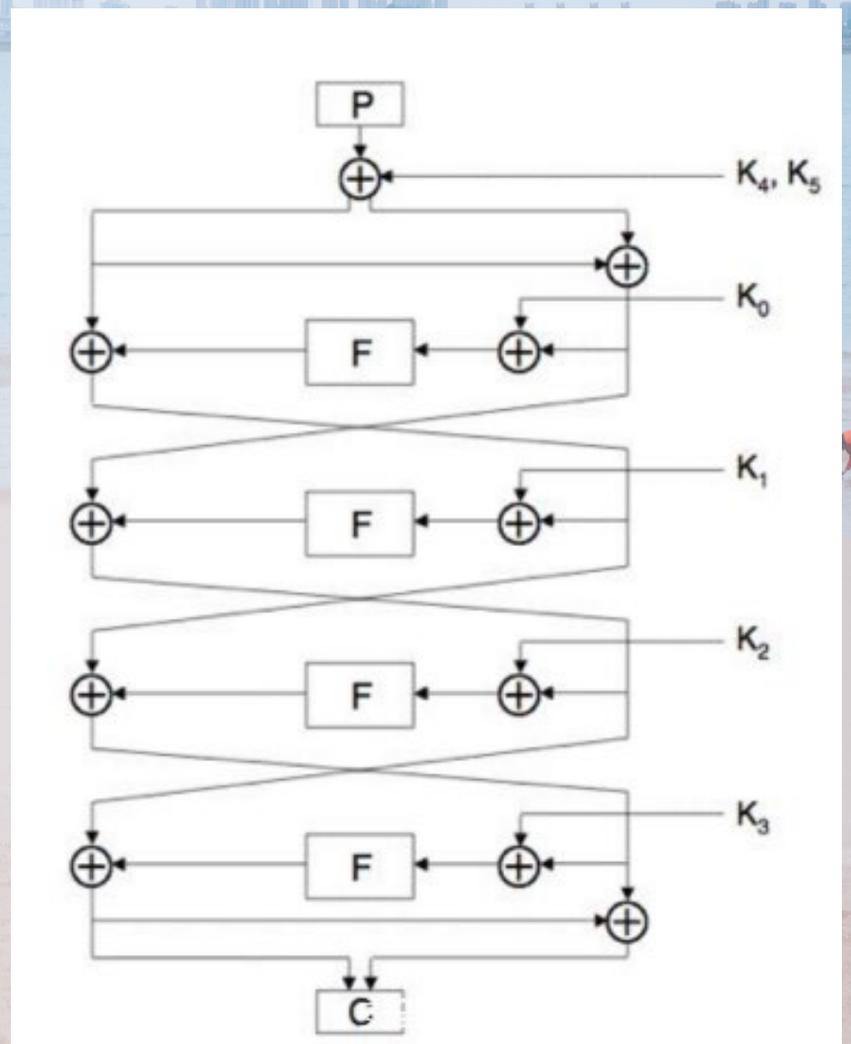
Feal-4

首先8字节明文分组被分成左右两个部分，左半部与子密钥K4异或，右半部与子密钥K5异或。两个半部再异或作为右半部。接着进入4轮轮函数。每一轮里，右半部直接成为下一轮的左半部。



Feal-4

第一轮里，右半部与K0异或，然后进入F函数，F函数实现非线性部分，F函数的结果与左半部进行异或，成为下一轮的右半部。后3轮同理，依次使用子密钥 K1 K2 K3。



Feal-4

轮函数F

轮函数F将32比特映射为32比特。轮函数中使用了一个名为G函数的操作，G函数有两个定义如下

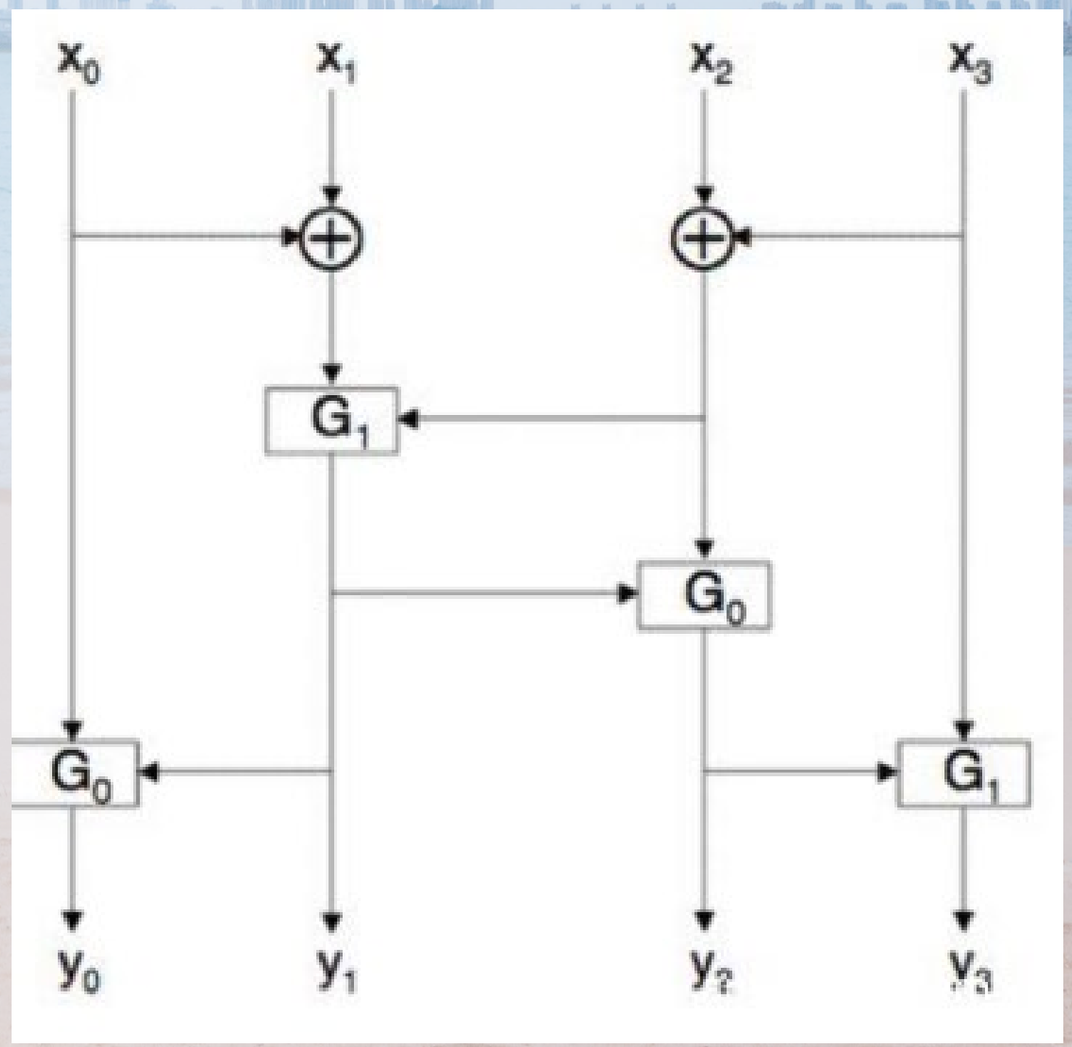
$$G_0(a,b) = (a + b \pmod{256}) <<< 2$$

$$G_1(a,b) = (a + b + 1 \pmod{256}) <<< 2$$

Feal-4

轮函数F

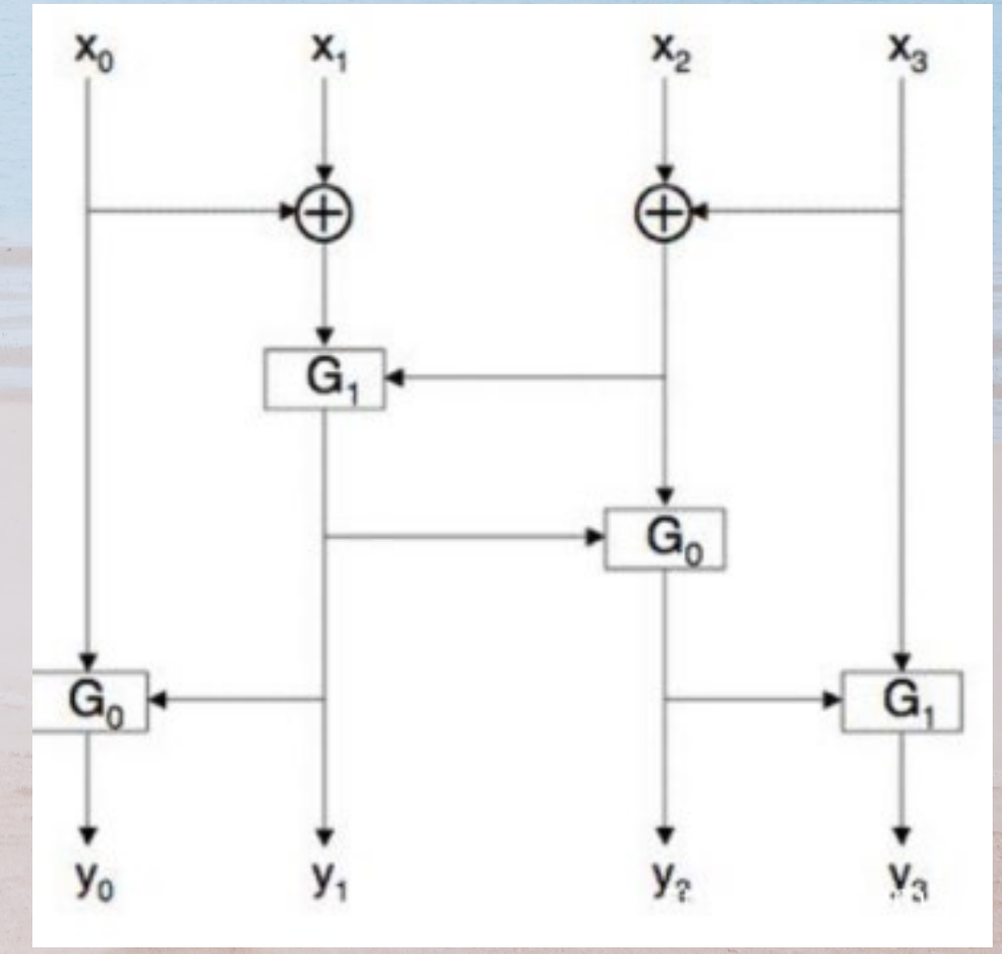
输入到输出的流程如下



Feal-4

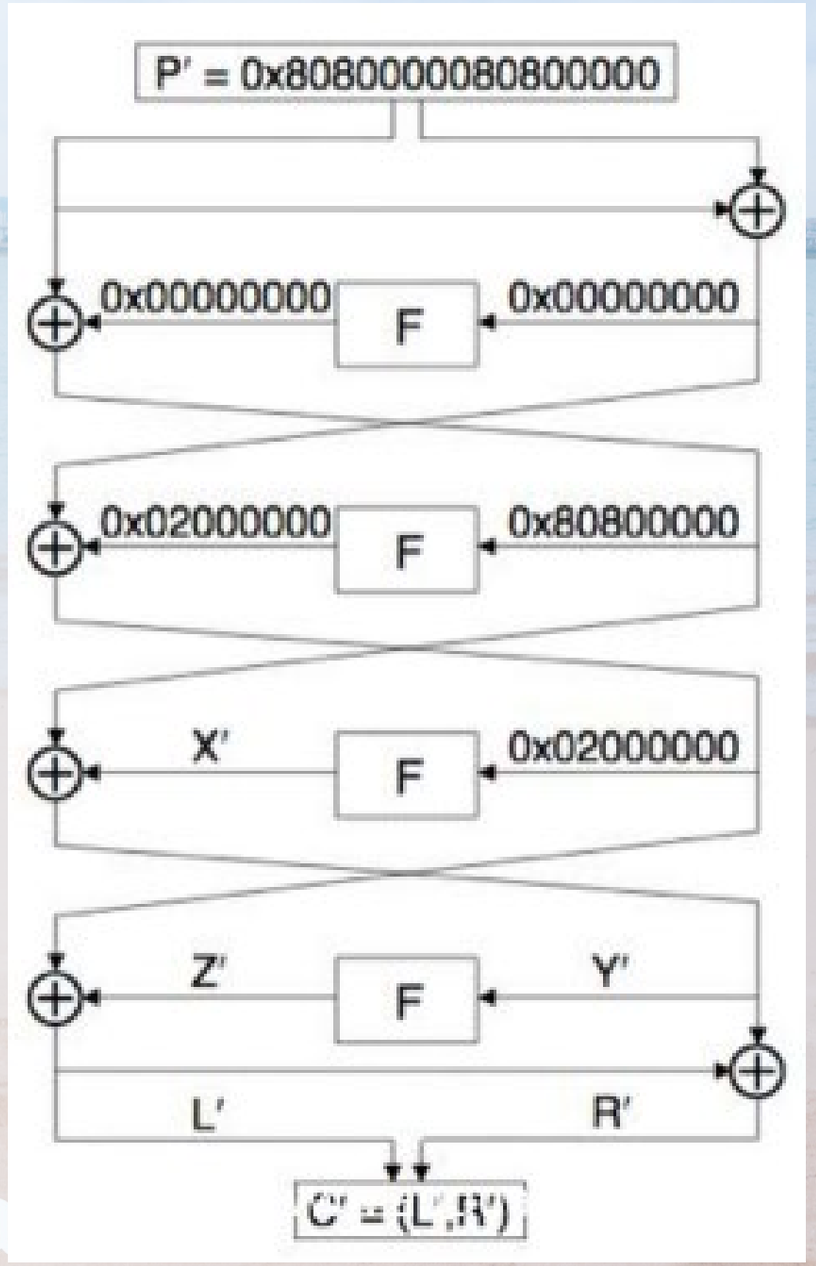
轮函数F

当轮函数的输入差值为0x00000000时，轮函数的输出差值一定时0x00000000，概率为1。第二个，当轮函数的输入差值为0x80800000时，轮函数的输出差值一定时0x02000000，概率为1。



Feal-4

构造差分



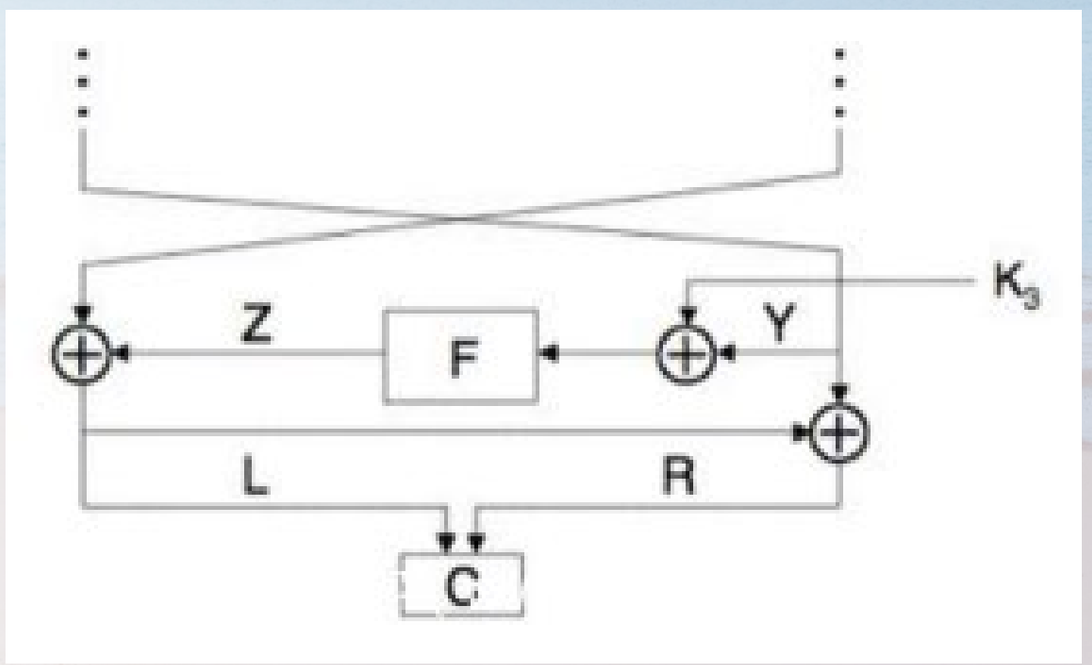
Y' = L' ^ R'

Z' = 0x02000000 ^ L'

X' = 0x80800000 ^ Y'

Feal-4

构造差分



Y0 = L0 ^ R0

Y1 = L1 ^ R1

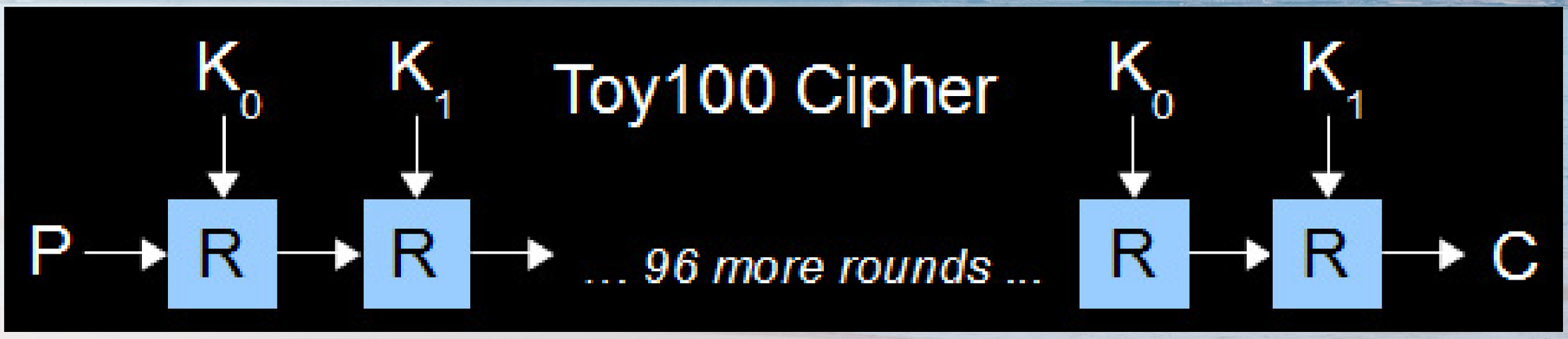
Z0 = F(Y0 ^ K3)

Z1 = F(Y1 ^ K3)

Z0^Z1 = F(Y0^K3) ^ F(Y1^K3) = Z’

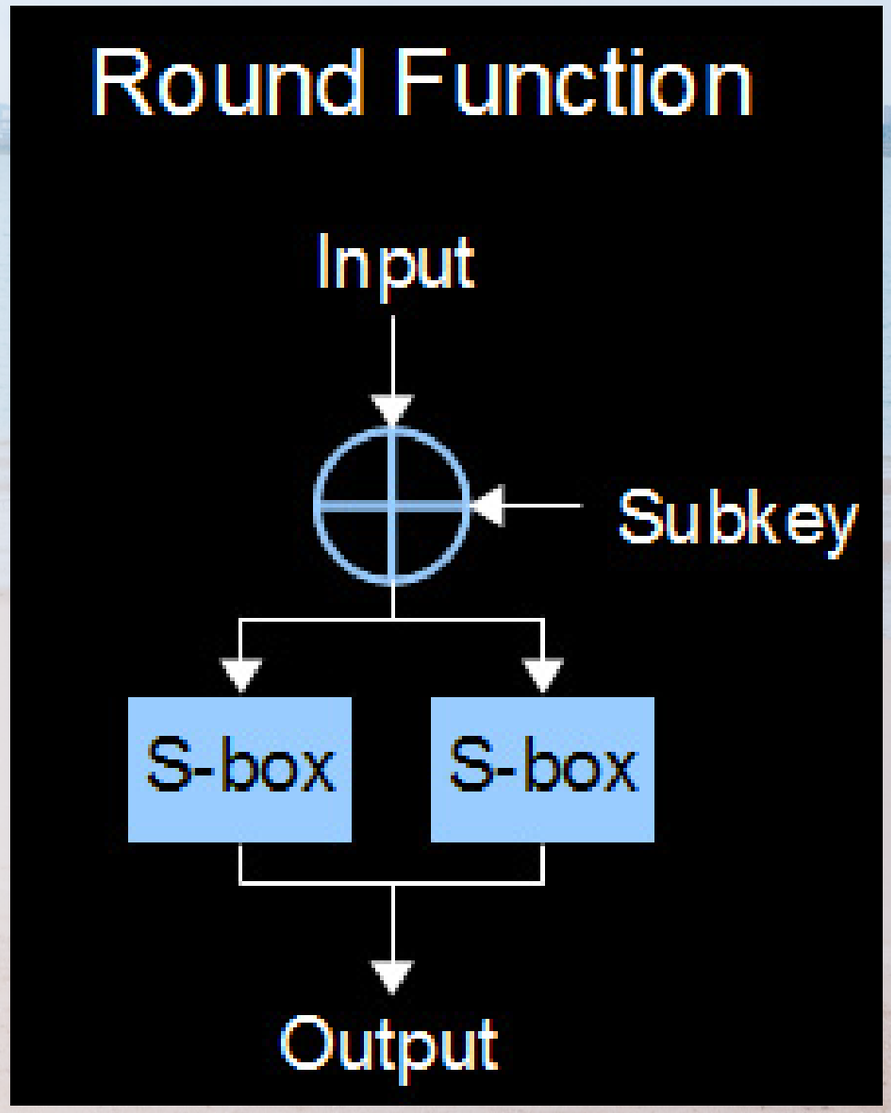
滑动攻击（Slide Attack）

现在我们考虑另一种极其简单的加密逻辑：进行100轮相同的加密算法，奇数轮用密钥K0，偶数轮用密钥K1



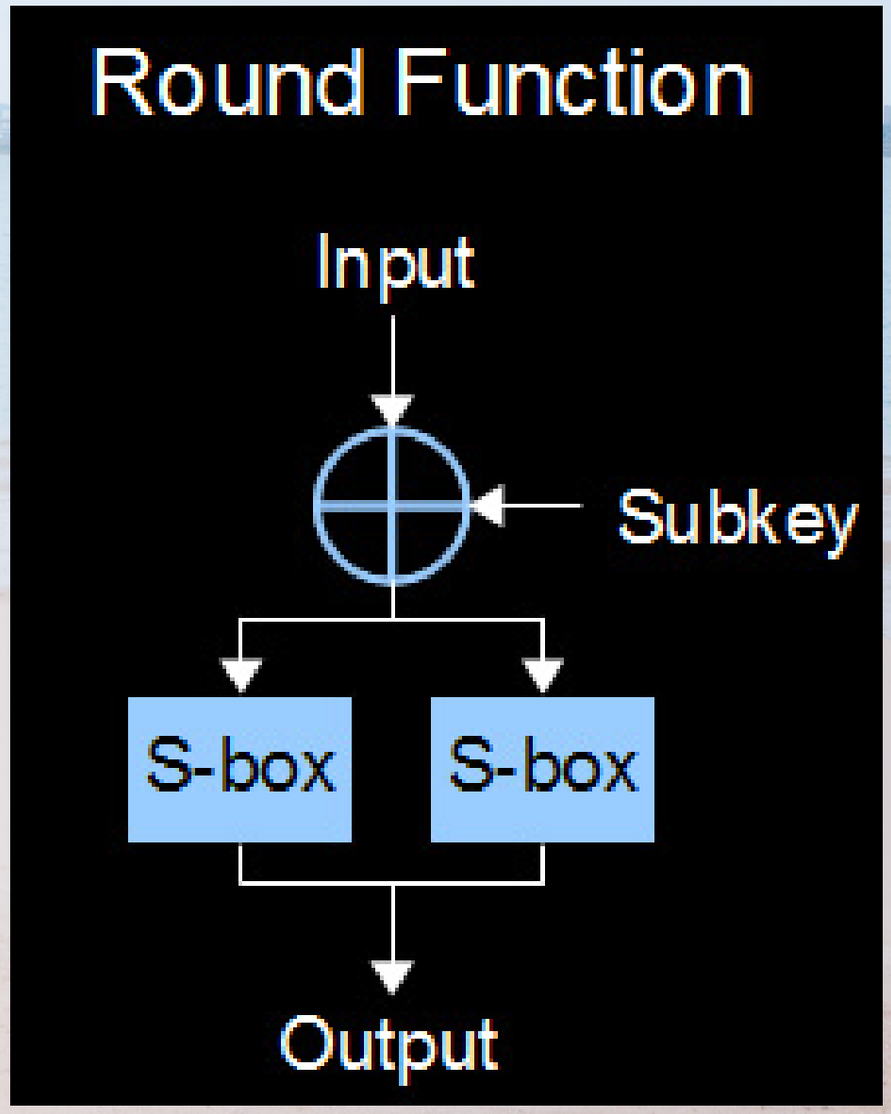
滑动攻击（Slide Attack）

每一轮的操作如下，即异或，然后分割，然后Sbox，然后合并



滑动攻击（Slide Attack）

由于轮数过多，前面提到的方法不再适用

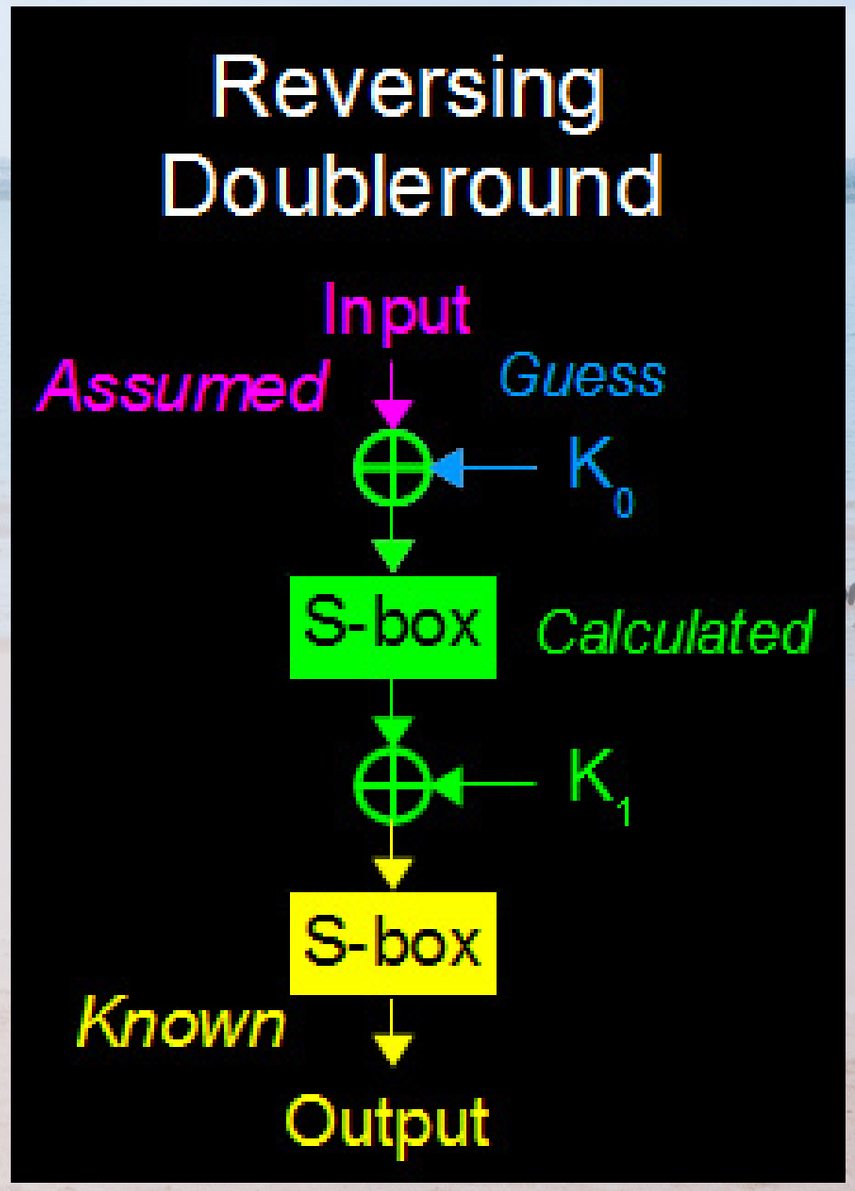


滑动攻击（Slide Attack）

这100轮的操作可以进一步变为50轮相同的操作，假设此时有Input和Output，猜测K0

可以推出K1

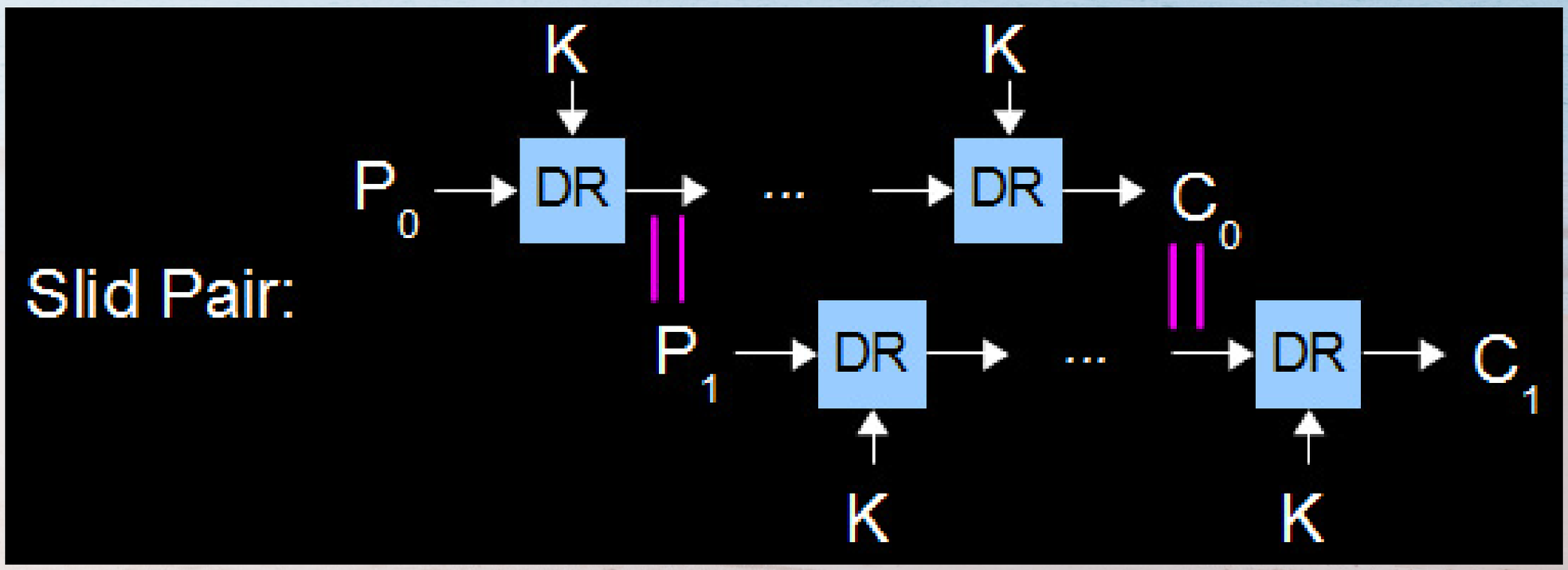
但有太多可能性！



滑动攻击（Slide Attack）

接下来我们就是尝试串起两个块！

P0的输出C0，P1的输出C1。假如P1是P0加密过程中第一轮的输出？！此时C0就是C1输入！通过P0,P1去猜测K0，然后能得到K1，然后用C0,C1进行验证！



滑动攻击（Slide Attack）

所以，滑动攻击的目标就是找到这样一对明密文。那么，概率是多少？

假设每一轮的功能就是x+1，那么我们的目标就是找到这样一对输入P0,P1，其中P1=P0+1。当我们所拥有的输入输出对越来越多时，找到的概率将不断飙升。

生日悖论！在不少于 23 个人中至少有两人生日相同的概率大于 50%。30 人时相同的概率为 70%。 60 人时，这种概率要大于 99%。

作业

- 完成一道AES byte flip+Orcale的题目
- 完成一道AES Padding Orcale的题目
- 利用线性分析求出ASPN的密钥