物联网安全

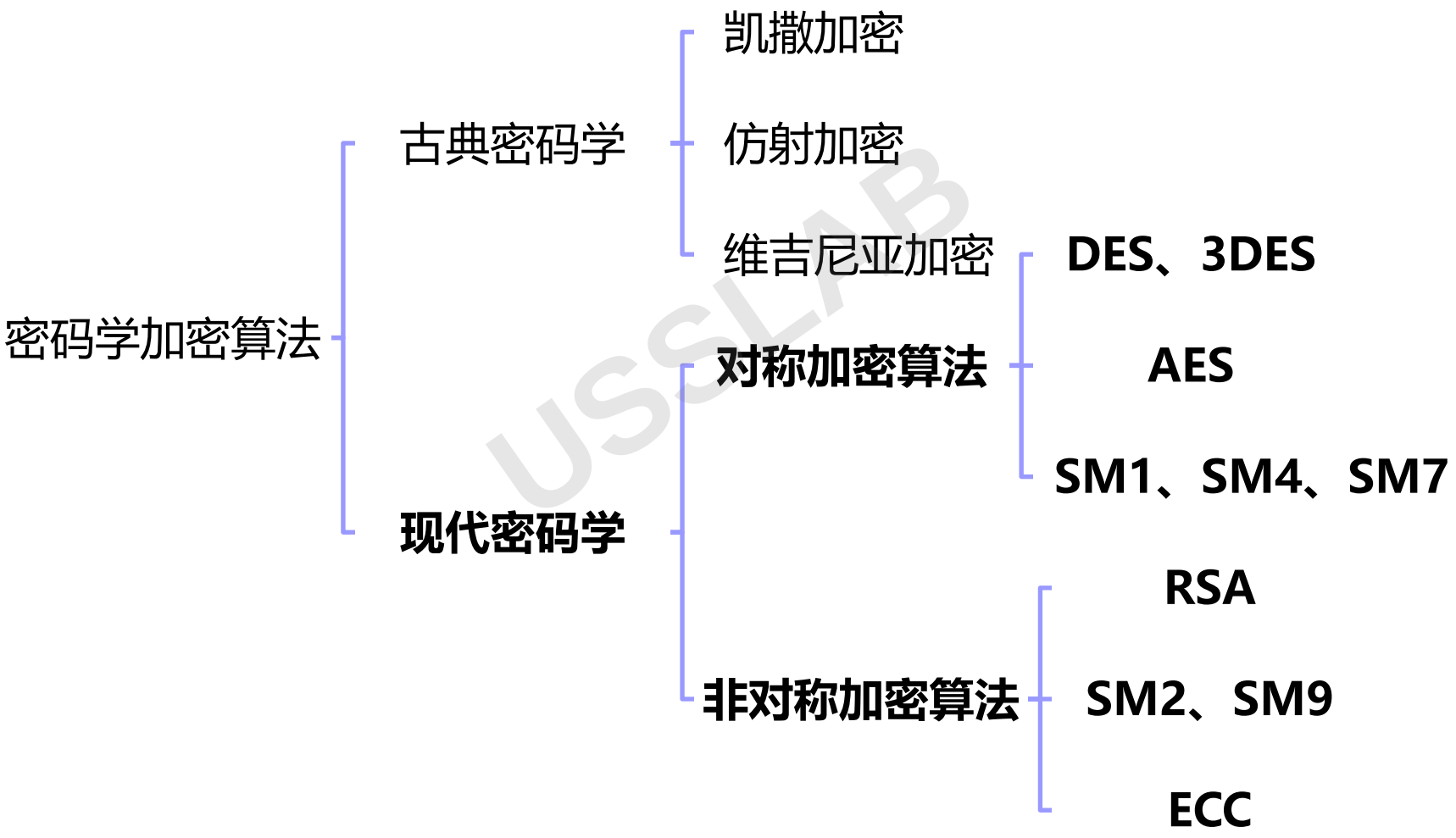
第三章：密码学基础——现代密码学

冀晓宇
浙江大学

课程概要

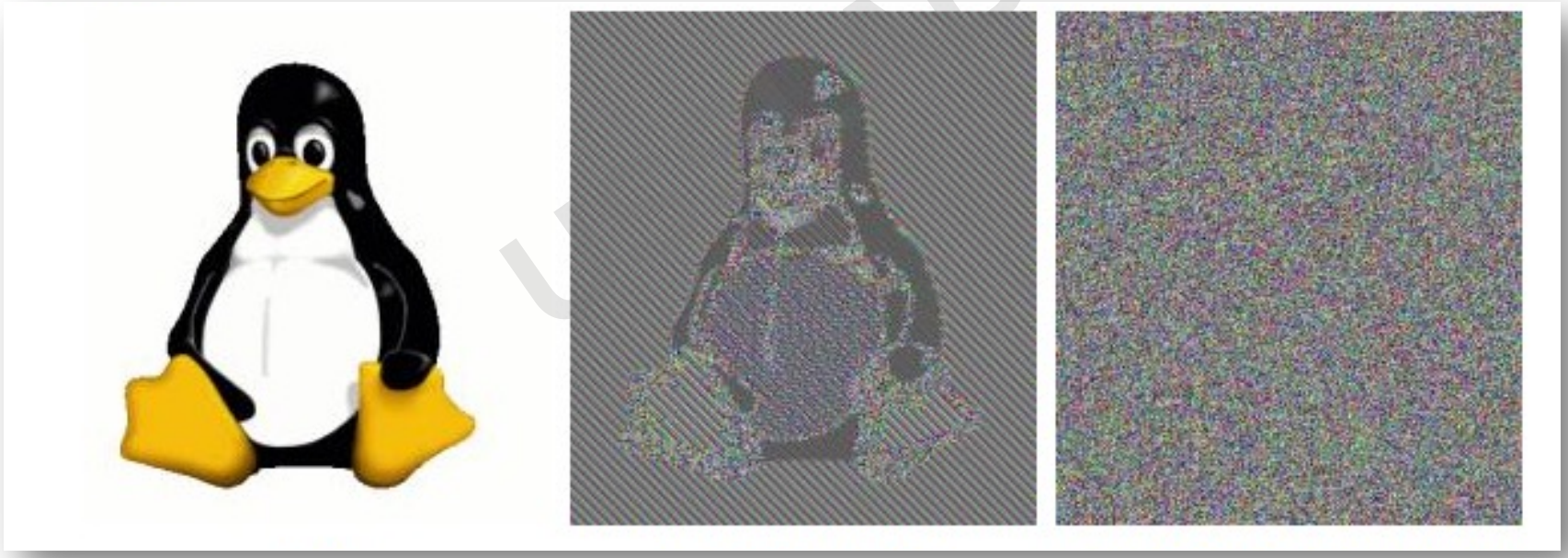
- **现代密码学重要思想**
- **私钥密码（对称密码）**
  - DES
  - 3DES
  - AES
- **公钥密码（非对称密码）**
  - 公钥密码体制
  - RSA加密
- **DH密钥交换协议**
- **国密算法**
- **物联网密码学技术应用**

密码学加密算法分类



现代密码学重要思想：混淆与扩散

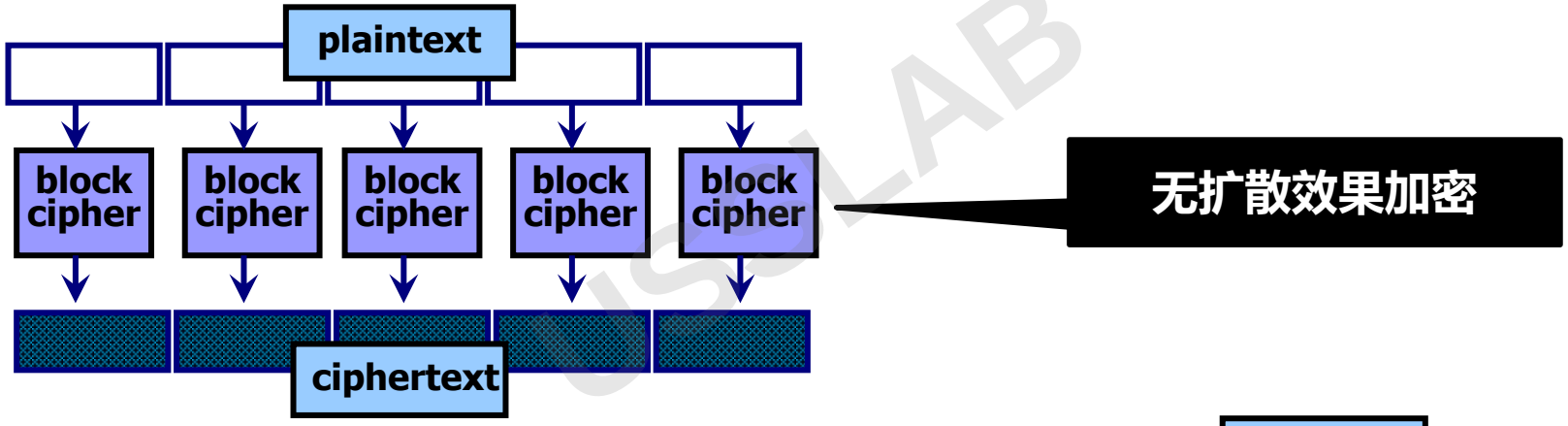
- **混淆（Confusion）**：使明文、密文和密钥之间的**关系尽可能模糊**的加密操作，在DES和AES中都有使用。

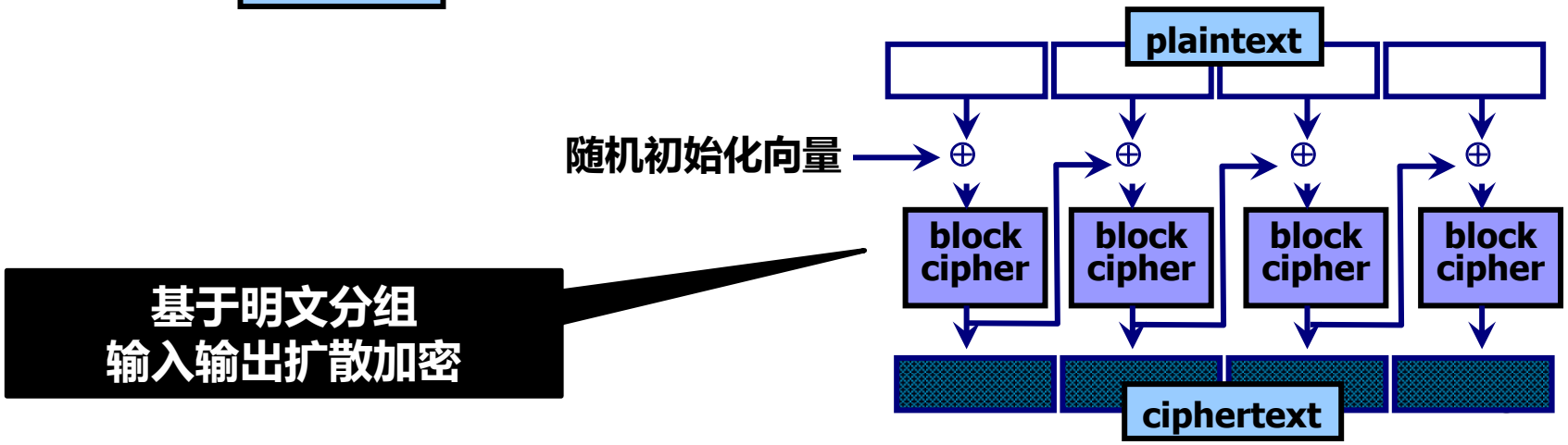


上图中，原文（左），混淆效果不佳的加密（中），混淆效果好的加密（右）

现代密码学重要思想：混淆与扩散

- **扩散（Diffusion）**：为了隐藏明文统计属性而将**一个明文符号的影响扩散到多个密文符号的加密操作**，如位置换。通常修改明文中1位会导致平均一半的输出位发生变化。





混淆与扩散——S盒（S-Box）

- **S盒**（Substitution-box，替换/代替盒）：对称密钥加密算法**执行替换计算的基本结构**，通常用于**模糊密钥与密文之间的关系**。
- 通常，S-Box接受特定数量的输入比特m，并将其转换为特定数量的输出比特n，其中n不一定等于m。一个m×n的S盒可以通过包含2^m条目，每条目n比特的查找表实现。
- S盒通常是固定的（例如DES和AES加密算法），**实现也是保密的**，也有一些加密算法的S盒是基于密钥动态生成的。

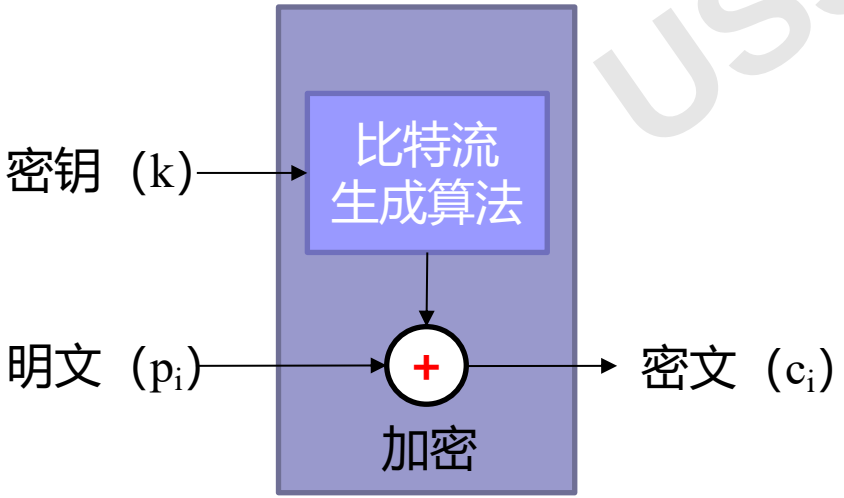
DES 6×4盒子，” 011011” → “1001”

S_5		中间四个比特															
		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
首尾比特	00	0010	1100	0100	0001	0111	1010	1011	0110	1000	0101	0011	1111	1101	0000	1110	1001
	01	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1010	0011	1001	1000	0110
	10	0100	0010	0001	1011	1010	1101	0111	1000	1111	1001	1100	0101	0110	0011	0000	1110
	11	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1010	0100	0101	0011

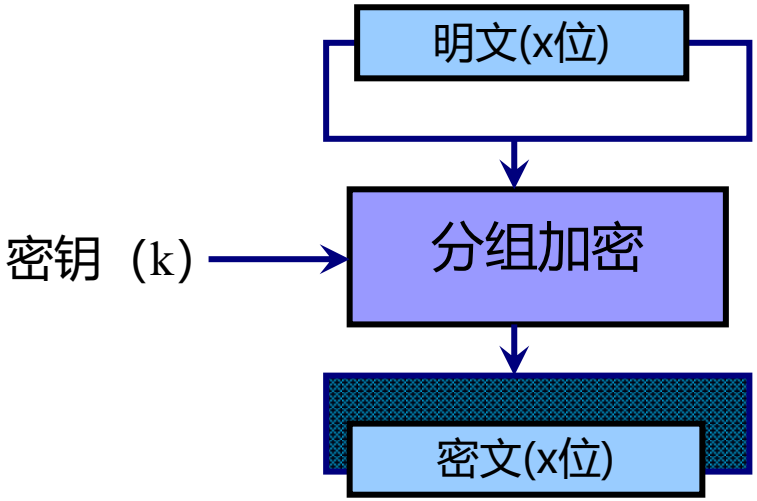
流密码与分组密码

- **流密码（streaming cipher）**：每次加密数据流的**一位或者一个字节**，如古典密码学。

- **分组密码（block cipher）**：将**明文分组作为整体加密**并且通常得到与明文等长的密文分组，也称之为块密码。



流加密



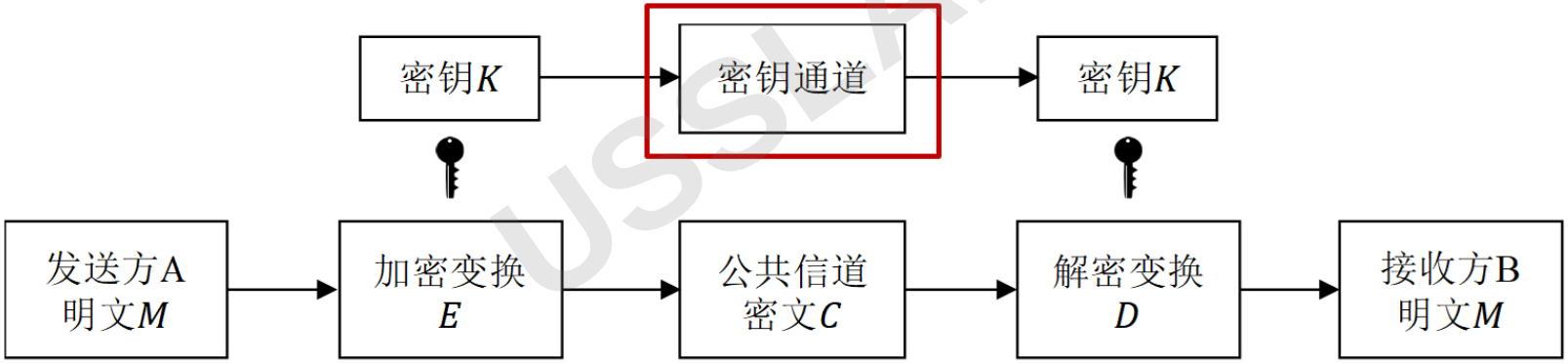
分组加密

3.1 Symmetric Encryption

私钥密码（对称密码）

对称加密

- **定义**：也称为**私钥/单钥加密**，特点是加密和解密都使用同一把密钥
- 是20世纪70年代公钥密码诞生之前唯一的加密类型，也是应用较为广泛的加密类型。
- 通信流程



加密：$C = E(K, P)$

解密：$P = D(K, C)$

Q：如果没有安全的信道传输密钥怎么办？

对称加密

- **对称加密安全性由以下两个条件保证：**
  - **加密算法具有安全性**。该条件要求即使攻击者窃取到一定数量的密文和对应的明文，也无法破译；然而，实际情况下需要公开加密算法。
  - 通信双方**通过安全信道获得对方的密钥并进行安全存储**。该条件要求密钥具备保密性，否则所有密文都会被破译。

对称加密的保密性的关键是什么？

密钥的保密性

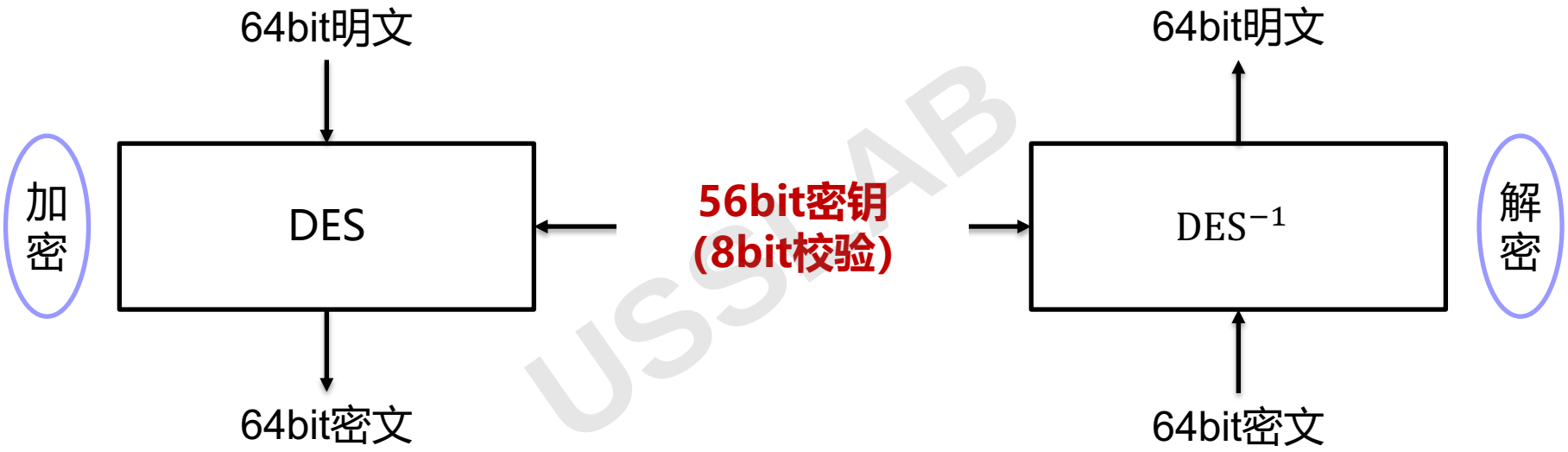
DES

- **定义**：DES(Data Encryption Standard)是一种使用56位密钥对64位长分组进行加密的密码，是一种迭代算法。DES是第一个公开的分组加密算法。

- **特点**：DES对明文中每个分组的加密过程都包含16轮，且每轮到操作都完全相同。每轮使用不同的子密钥，但是子密钥是从主密钥中推导而来。

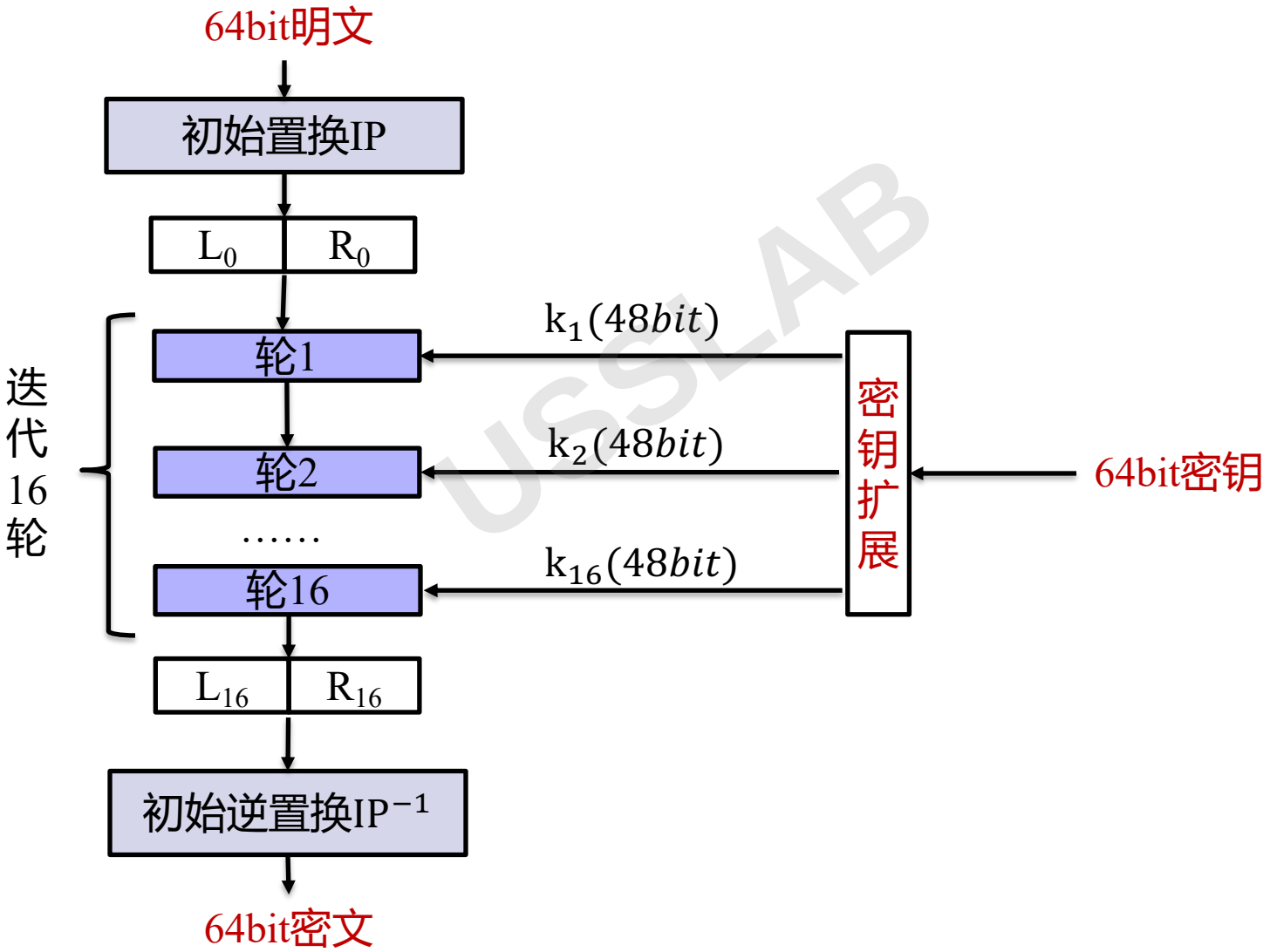
- 历史过程：算法征集及标准化
  - 1972年美国国家标准局NBS（National Bureau of Standards）开始实施计算机数据保护标准开发计划
  - 1973年，NBS公开征集计算机数据加密算法
  - 1977年，NBS采纳了IBM的Lucifer算法的修正版作为DES

DES加密示意图



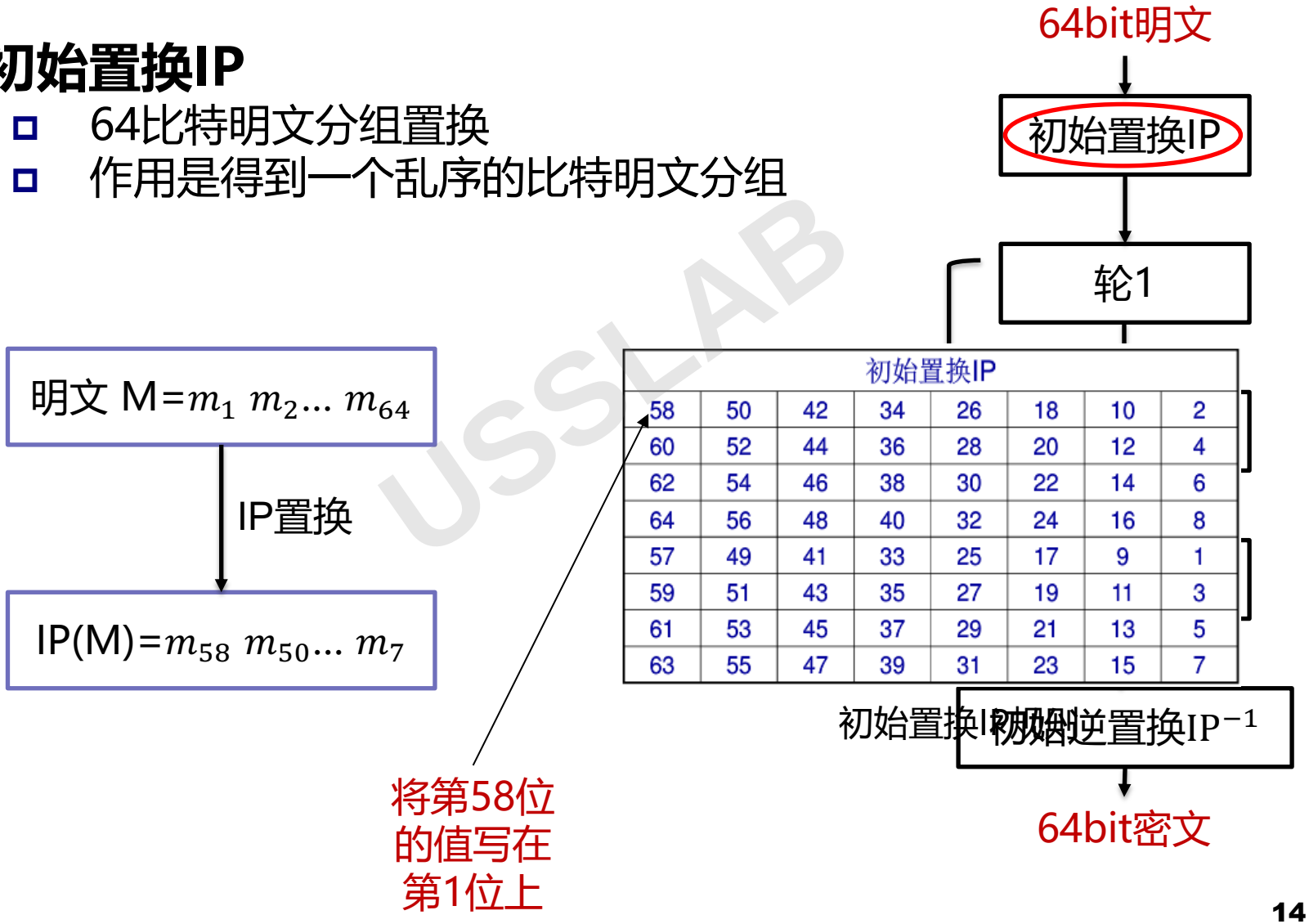
分组长度：64比特　密钥长度：64比特　有效密钥长度：56比特

DES的加密流程



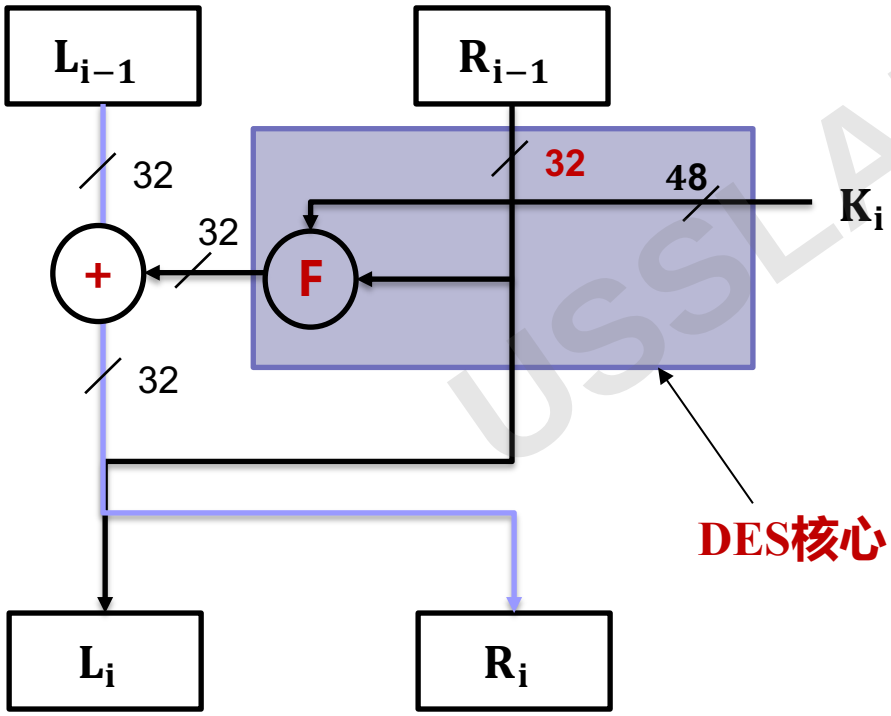
DES的加密流程1—置换

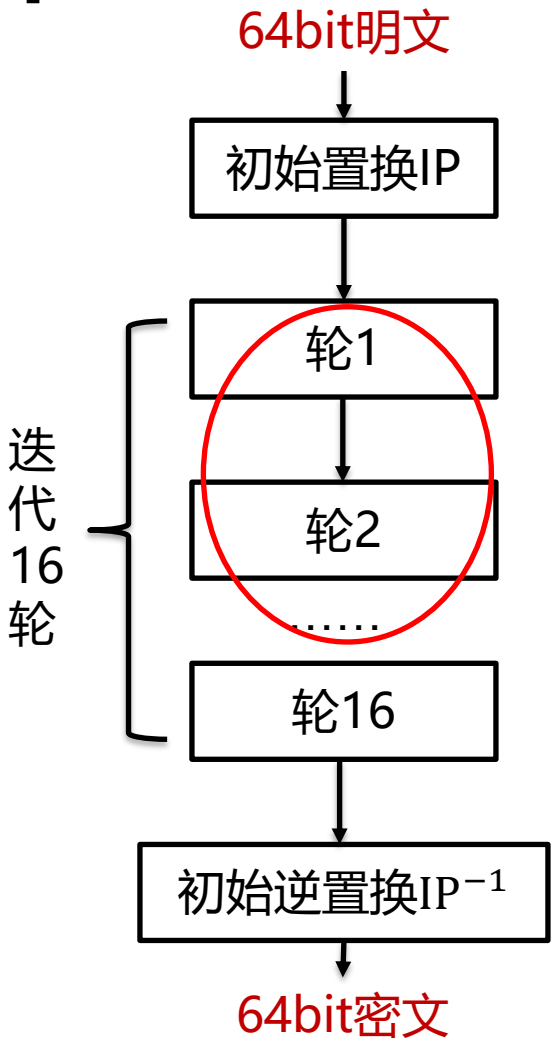
- **初始置换IP**
  - 64比特明文分组置换
  - 作用是得到一个乱序的比特明文分组



DES的加密流程2—轮内加密

- 基于Feistel结构的加密方法。

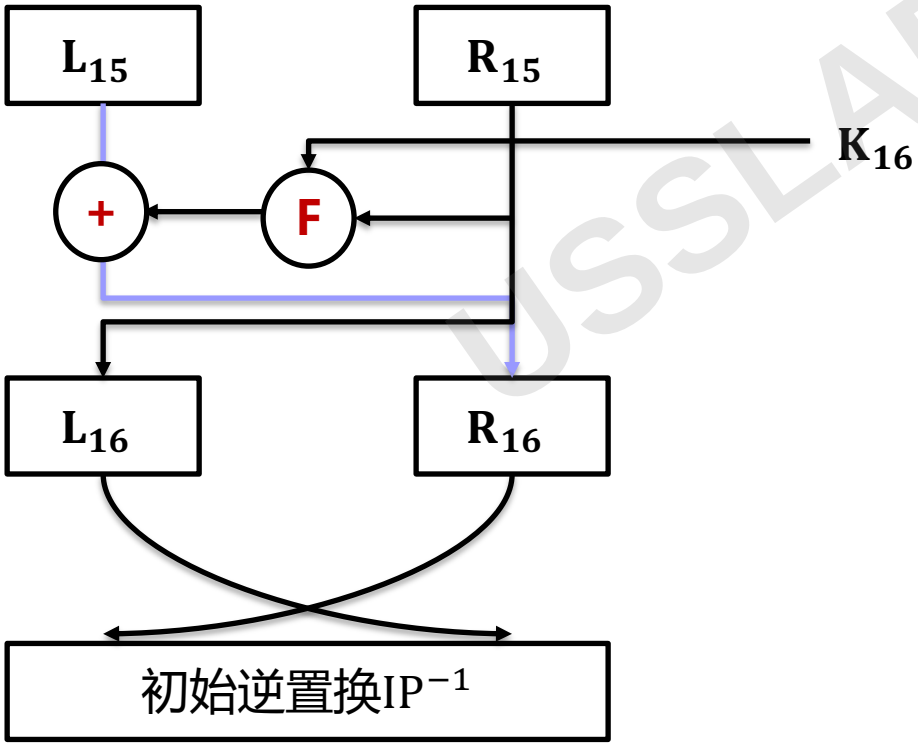


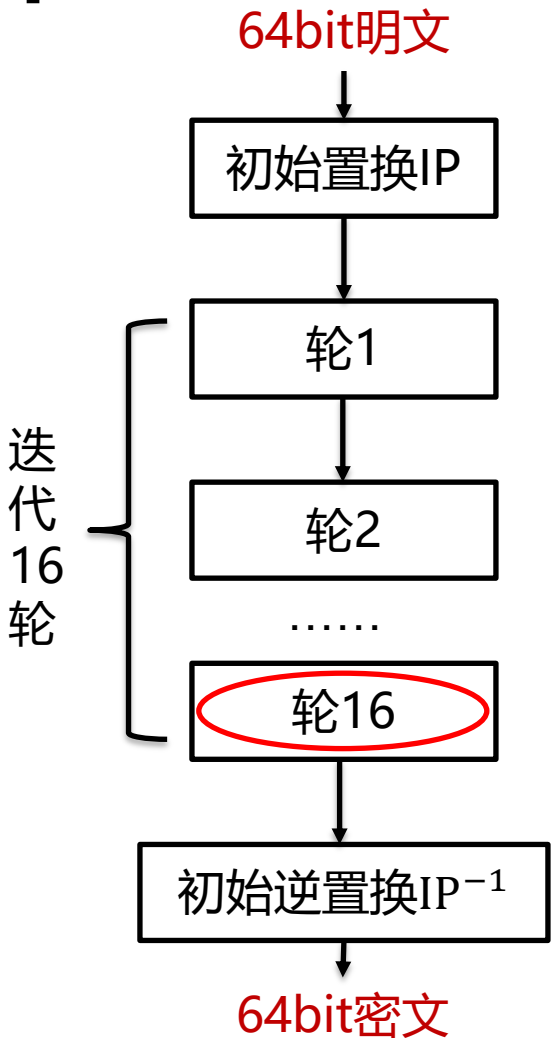


Q：48位和32位的数如何进行运算?

DES的加密流程2—轮内加密

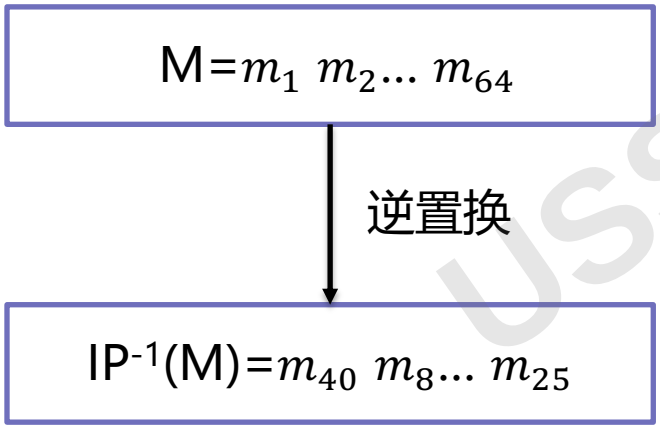
- 第16轮迭代

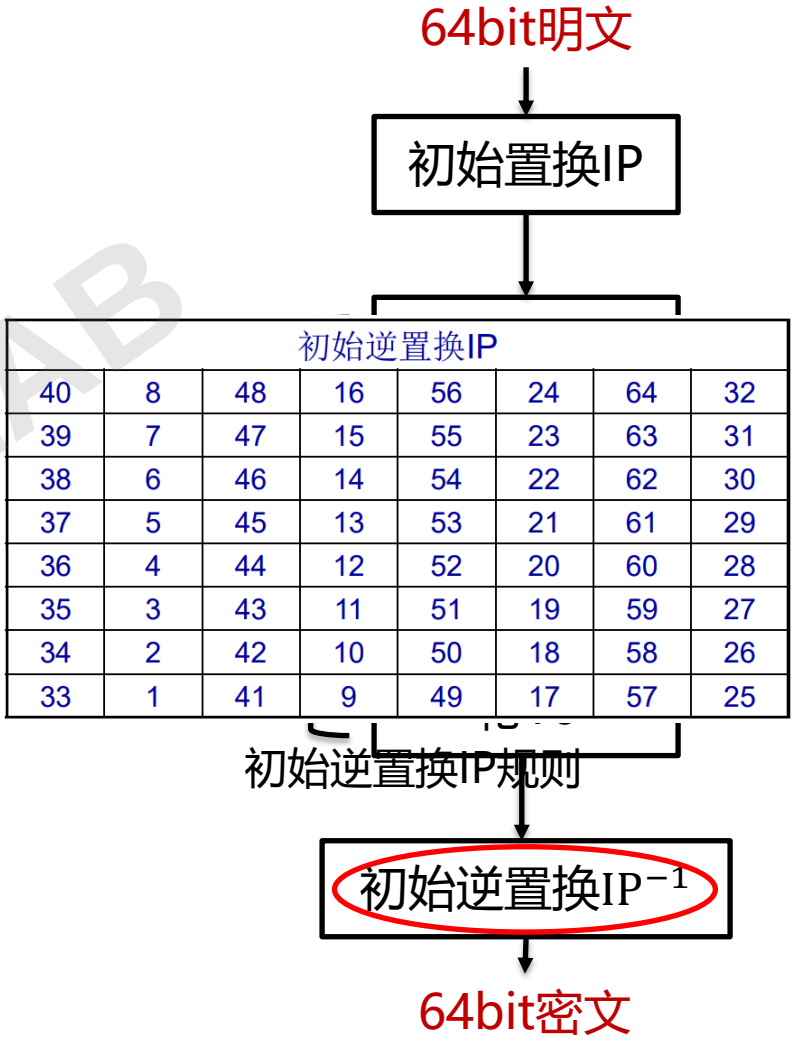




DES的加密流程3—逆置换

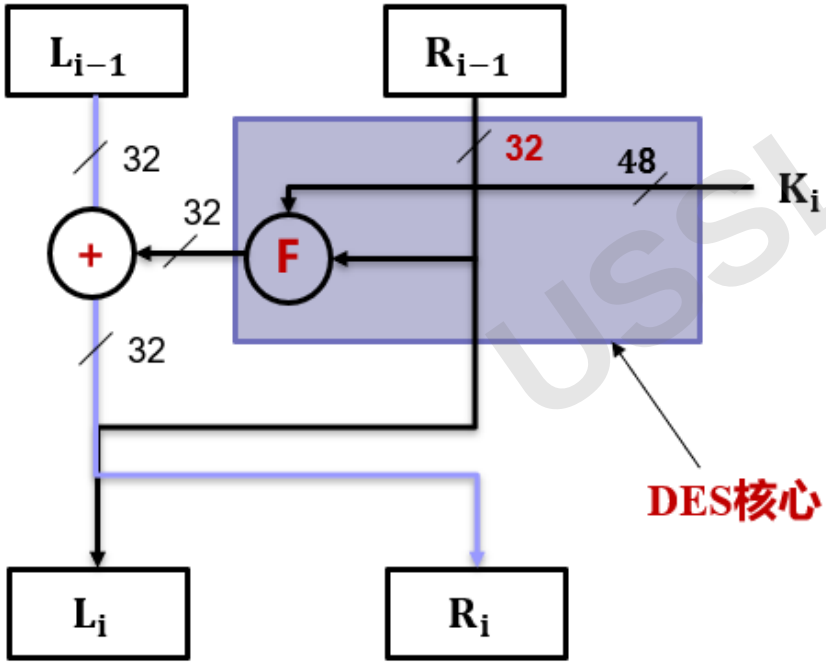
- **逆置换IP^{-1}**
  - 初始置换IP的逆变换

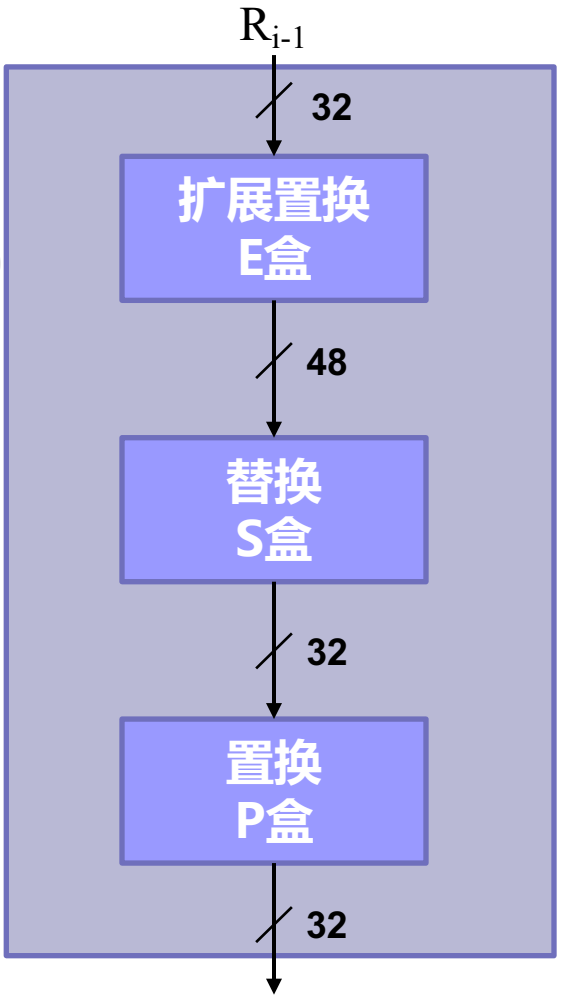




DES的核心-F函数

- F函数实现了混淆和扩散的功能。





F函数计算-示例演示

- 已知：

  明文：$30\ 31\ 32\ 33\ \ 34\ 35\ 36\ 37_{16}$

  密钥：$31\ 32\ 33\ 34\ \ 35\ 36\ 37\ 38_{16}$

- 求密文

DES示例演示

- **1. 初始置换IP**

30 31 32 33 34 35 36 37

00110000 00110001 00110010 00110011 00110100 00110101 00110110 00110111

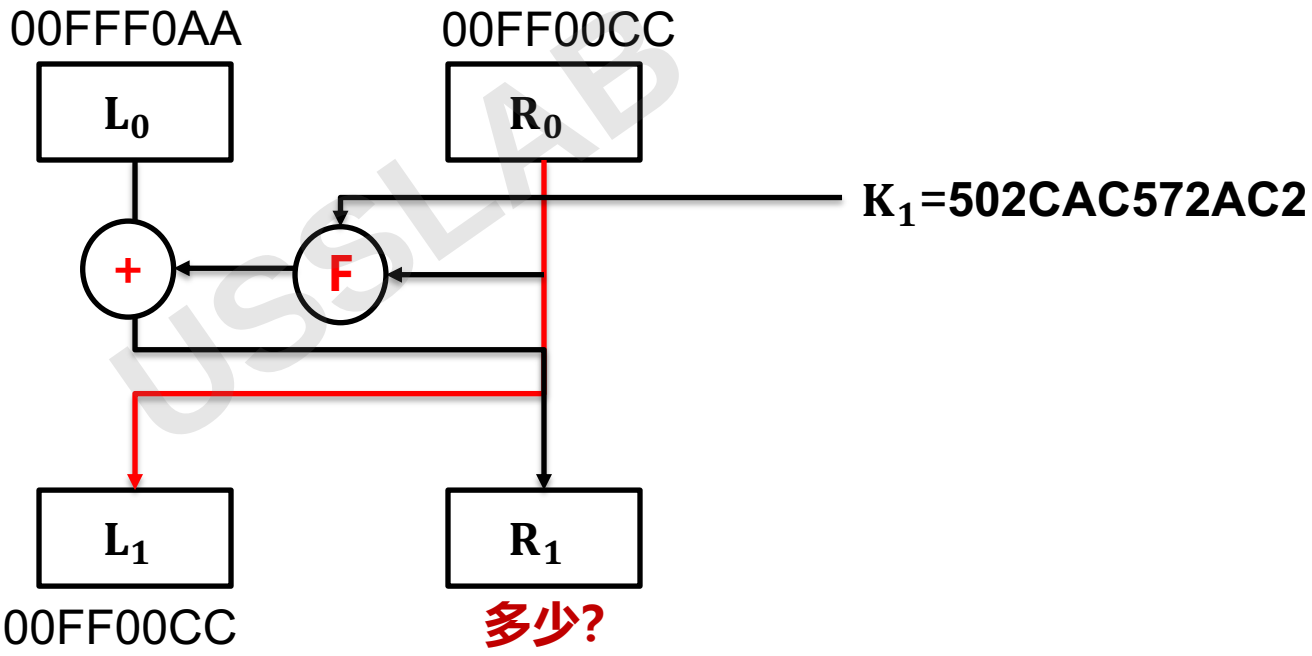
初始置换IP							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

00000000 11111111 11110000 10101010 00000000 11111111 00000000 11001100

00 FF F0 AA 00 FF 00 CC

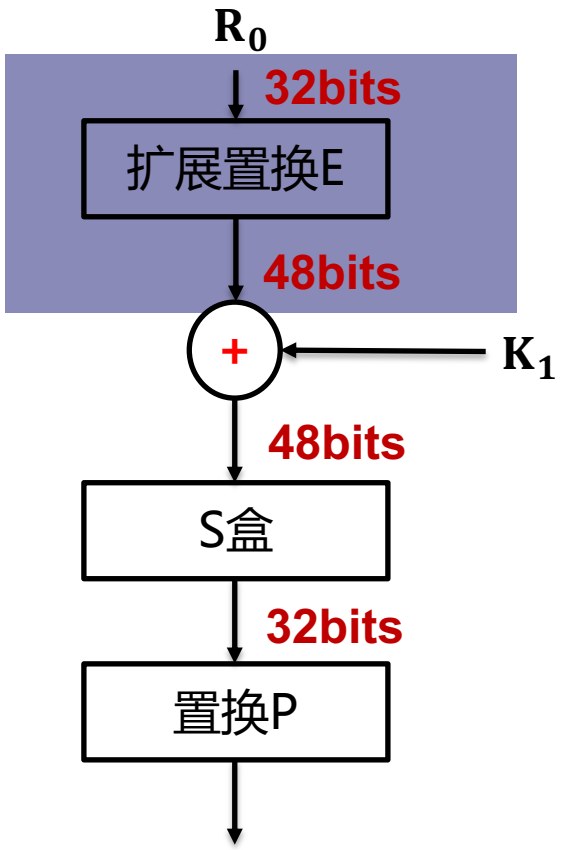
DES示例演示

- **2. 第一轮加密**
  - **假定子密钥 K_1=502CAC572AC2$_{16}$**



DES示例演示

- **3. F函数-扩展置换E**



R0=00FF00CC

输入：0000 0000 1111 1111 0000 0000 1100 1100
第一行 第二行 ……

① 假设对第二行“0000”进行扩展，也就是5、6、7、8位

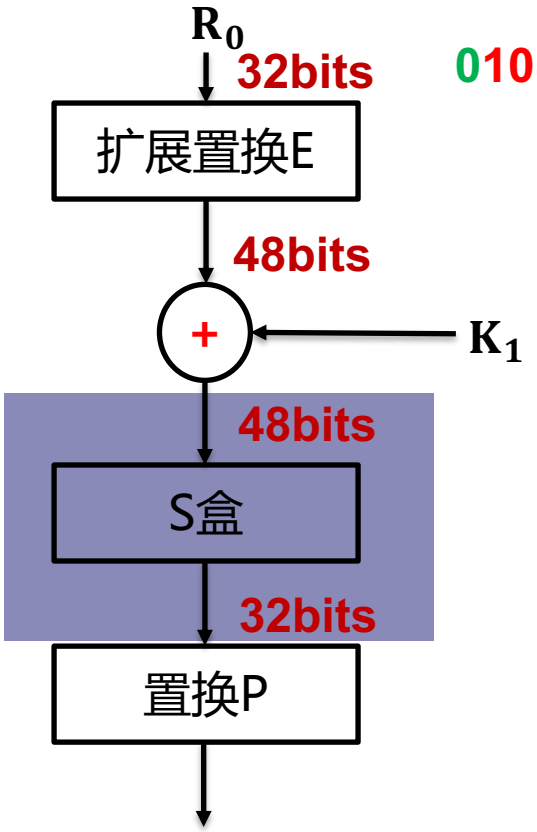
扩展置换E					
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

② 5、6、7、8开头和结尾分别补充第4、9位的值

输出：000000 000001 011111 111110 100000 000001 011001 011000

DES示例演示

- **4. F函数-S盒**



假设对“010100”进行替换，分别以“1010”（二进制10）和00（二进制0）作为index查表，得到的值即为替换结果

010100 000011 101101 010010 110101 110011 110010 011010

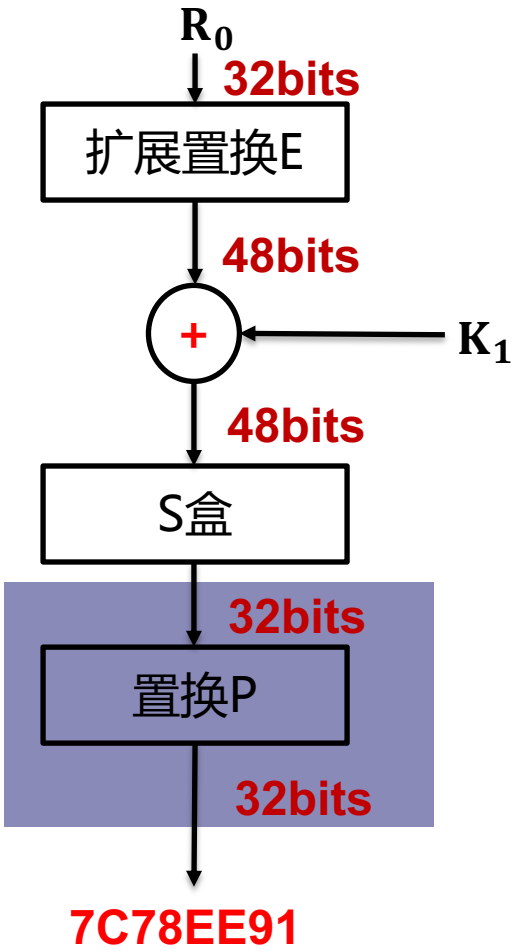
（上页扩展置换E输出结果和密钥K_1异或之后的结果）

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

0110 1111 0001 1010 0011 1011 1100 1001

DES示例演示

- **5. F函数-置换P**



0110 1111 0001 1010 0011 1011 1100 1001

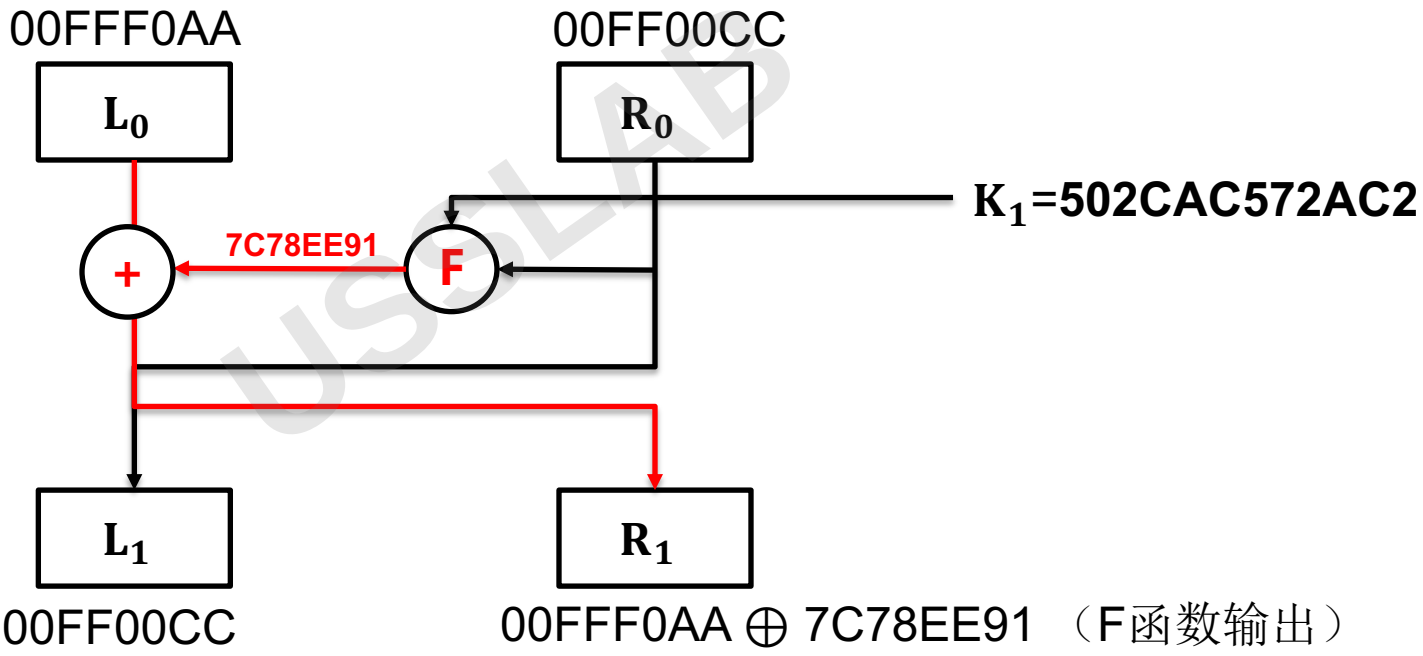
置换P			
16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

0011 1100 0111 1000 1110 1110 1001 0001

7C78EE91

DES示例演示

- **第一轮加密**
  - **假定子密钥 $K_1=502CAC572AC2_{16}$**



DES示例演示

- **6.初始逆置换IP^{-1}**

D4 16 8A A1 33 F6 AD 45

初始逆置换IP							
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

8B B4 7A 0C F0 A9 62 6D

思考

- **安全性**：Feistel每轮中仅加密（解密）输入位的一半，F函数只加密左半部分，不加密右半部分。

- **优势**：DES优势是**解密的过程和加密过程完全相同**，只是解密过程第i轮需要用到第16-i轮到密钥。

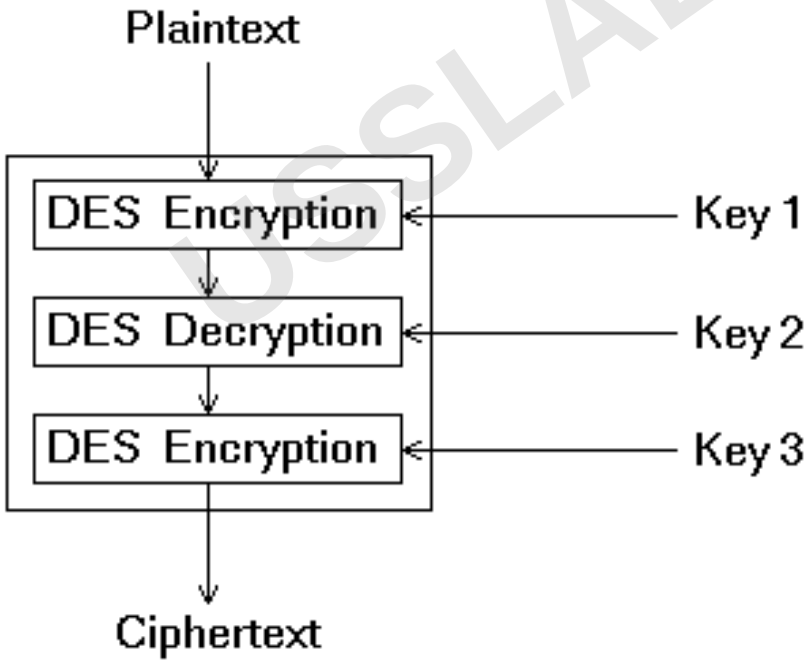
DES安全性如何?

- **词频统计分析**：利用密钥之间的关联性，实际几乎没有

- **暴力穷举攻击**：$2^{56}=7.2*10^{16}$
  - 1997年Diffie和Hellman提出制造一个每秒测试106个密钥的芯片，一天可以搜索整个密钥空间，大约要两千万美元；
  - Deep Crack：25万美金，平均15天；
  - COPACOBANA：1万美金，平均7天；
  - 1997年1月28日，美国RSA数据安全公司在互联网上开展了“**密钥挑战**”的竞赛，一位名叫Rocke Verser的程序员设计了一个通过互联网分段运行的密钥穷举搜索程序，成千上万志愿者加入其中，在1997年6月17日成功找到了密钥。
- 随着计算能力的不断提升， DES加密安全性问题暴露。

3.1.2 DES的改进—3DES

- **3DES**：由三个连续的DES加密组成，也称之为三重DES。
- 三个密码组件既可以是一个加密函数又可以是一个解密函数。

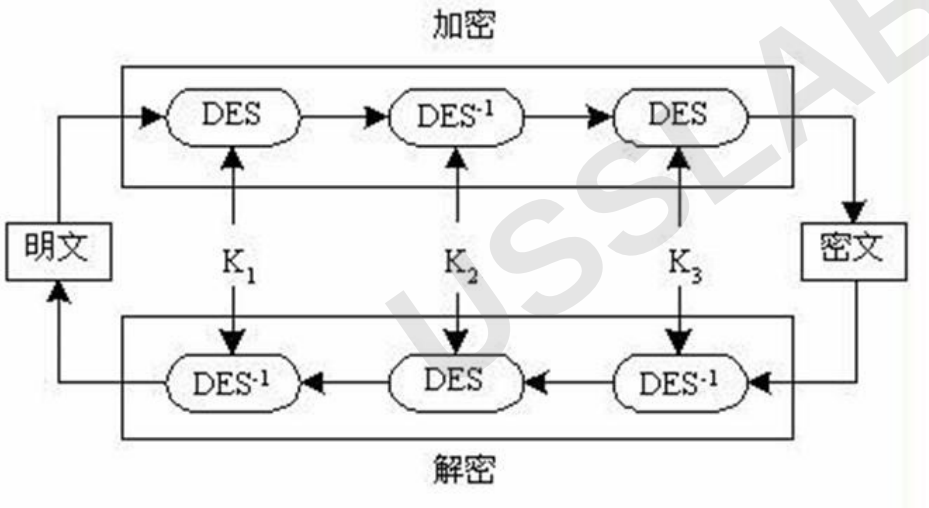


3DES

- **3DES加密解密流程**

  加密：$y = E_{k_3}[D_{k_2}[E_{k_1}[x]]]$

  解密：$x = D_{k_3}[E_{k_2}[D_{k_1}[y]]]$



- **3DES安全性**
  - 若k_1 、k_2、 k_3互不相等，密钥长度56*3=168，暴力破解需要$5.8*10^{29}$年!
  - 3DES目前仍有足够的安全性

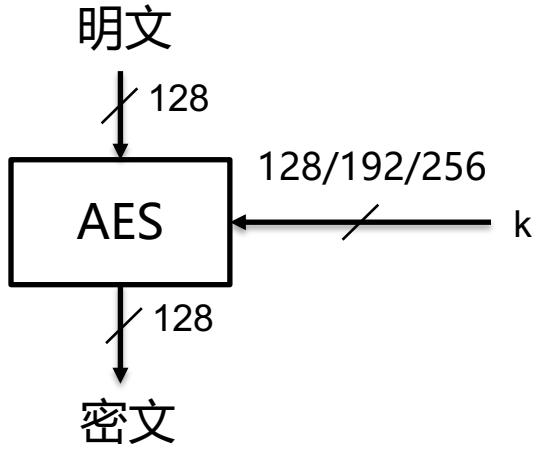
3DES的问题

- Pros: DES用于历史遗留系统，3DES兼容用于新系统

- Cons:
  - 实现效率：3DES实现时间是DES的3倍；
  - 安全性：分组较小，仍然只有64位。

3.1.3 AES

- **高级加密标准**（Advanced Encryption Standard）：分组为128位、支持3种密钥长度（128、192、256）且软硬件实现都很高效。
- 是当前应用最广泛的对称算法之一



- 1997年美国国家标准与技术研究院（NIST）向社会征集新的高级加密标准。
- 2001年比利时两位年青科学家Daemen和Rijmen提交的分组密码Rijndael成为新的高级加密标准。

（注：Rijndael 读成Rain Doll。）

Daemen和Rijmen

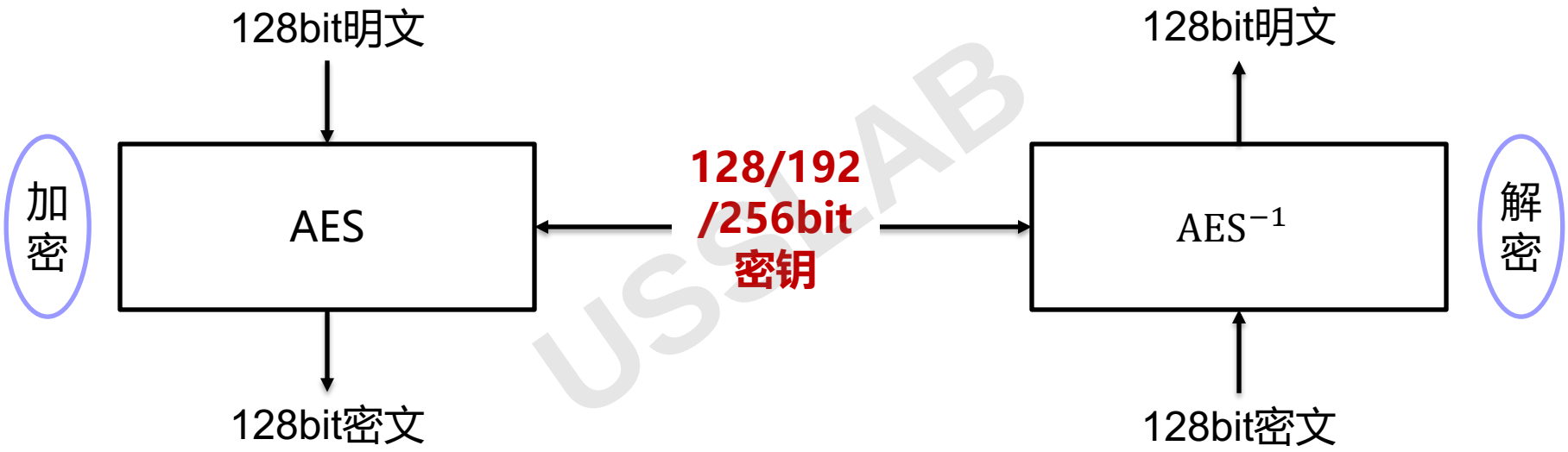
http://www.esat.kuleuven.ac.be/~rijmen/rijndael/

AES评估准则

- 一般安全性
  - 依赖于密码学界的公共安全分析
- 软件实现
  - 软件执行速度，跨平台执行能力及密钥长度改变时速度变化
- 受限空间环境
  - 在比如智能卡中应用
- 硬件实现
  - 硬件实现时能够提高执行速度或缩短代码长度
- 抵御密码分析攻击
- 密钥灵活性
  - 快速改变密钥长度的能力
- 其他的多功能性和灵活性
- 指令级并行执行的潜力

3.1.3 AES

- **AES算法参数**



	AES-128	AES-192	AES-256
分组长度	128		
密钥长度	128	192	256
迭代轮数	10	12	14
应用场景	常规数据加密（主流）	高安全需求系统	军事/金融级加密

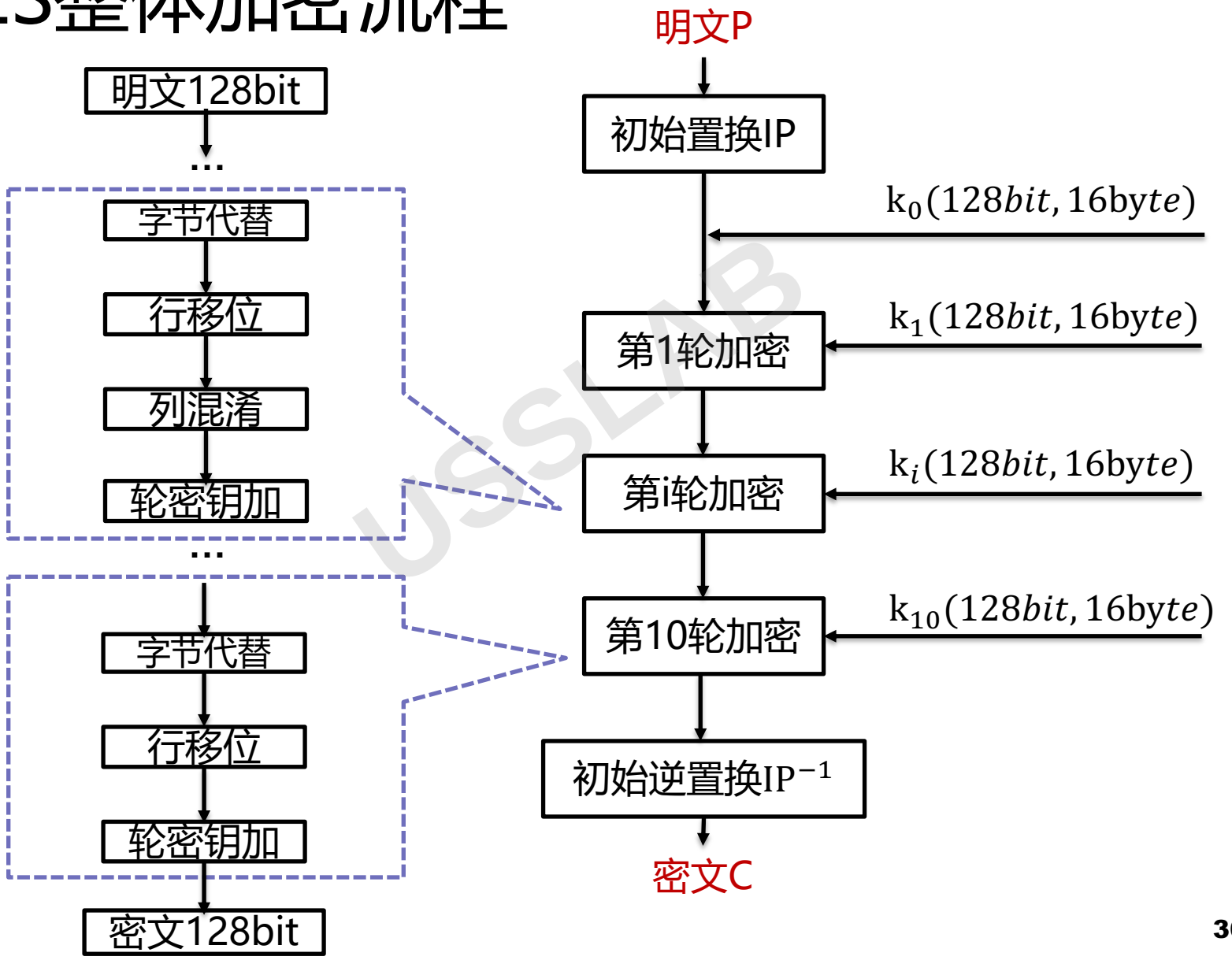
AES的结构

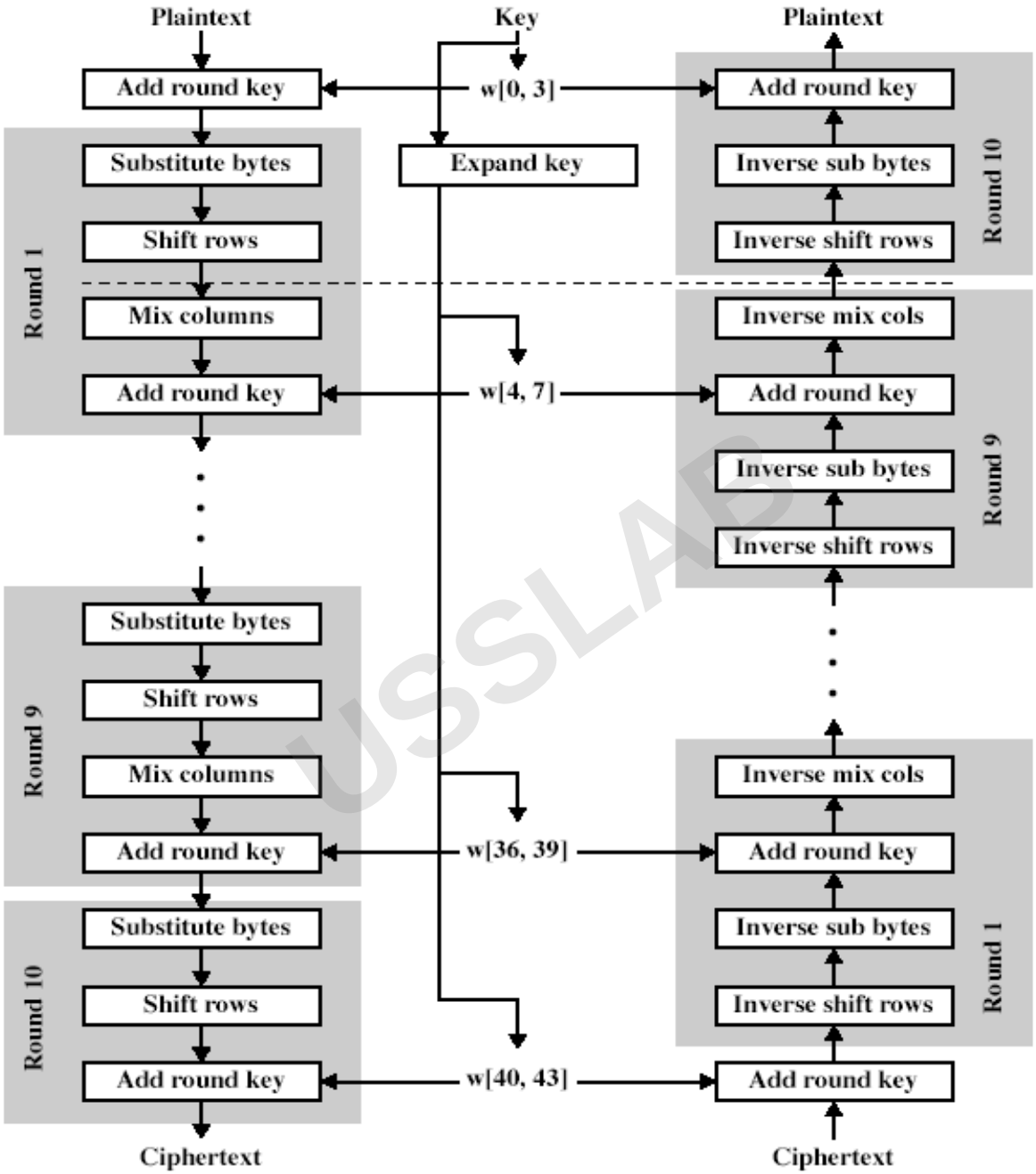
- 非Feistel结构
- 密钥：扩展成由**44个32位字（4字节）**所组成的数组
- 每轮密钥：4个字w[*i*, *i+4*]，即16字节，作为该轮轮密钥

- 每一轮内部加密的核心思想：
  - **字节代替**：用一个S盒完成分组中的按字节代替
  - **行移位**：一个简单的置换
  - **列混淆**：利用在伽罗瓦域上的算术特性的代换
  - **轮密钥加**：利用当前分组和扩展密钥的一部分进行按位XOR

- 特点：
  - 算法结构简单：**仅在轮加密阶段中使用密钥！**
  - 每个阶段均可逆
  - 加密和解密过程的最后一轮均只包括三个阶段

AES整体加密流程

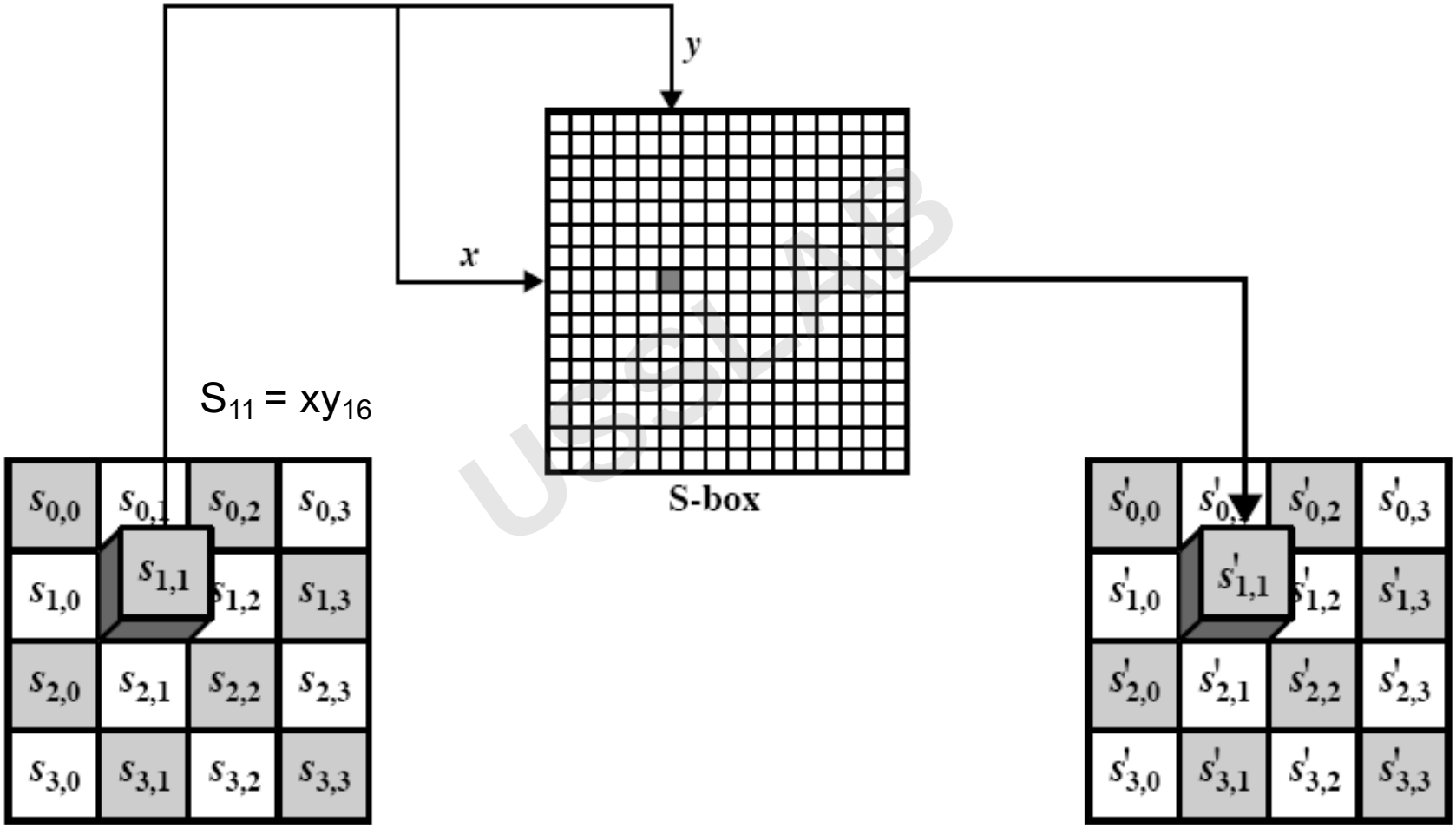



Plaintext
Key
Plaintext
Add round key
w[0, 3]
Add round key
Substitute bytes
Expand key
Inverse sub bytes
Round 10
Shift rows
Inverse shift rows
Round 1
Mix columns
Inverse mix cols
Add round key
w[4, 7]
Add round key
Inverse sub bytes
Round 9
Inverse shift rows
Substitute bytes
Shift rows
Inverse mix cols
Round 9
Mix columns
Add round key
w[36, 39]
Add round key
Substitute bytes
Inverse sub bytes
Shift rows
Round 1
Round 10
Inverse shift rows
Add round key
w[40, 43]
Add round key
Ciphertext
Ciphertext
(a) Encryption
(b) Decryption

AES每一轮内部：四个阶段

- 字节代替（Substitute bytes）
- 行移位（ShiftRows）
- 列混淆（MixColumns）
- 轮密钥加（AddRoundKey）

第一步：字节代替



(a) Substitute byte transformation

字节代替：AES的S盒

Table 5.4 AES S-Boxes

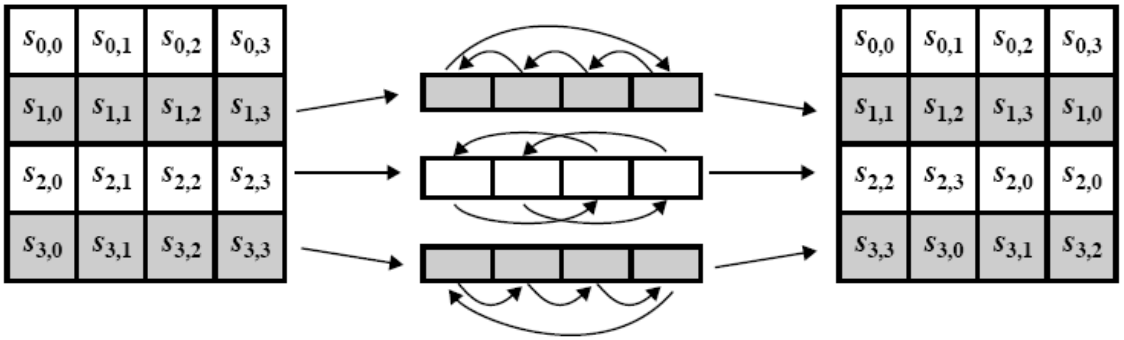
(a) S-box

		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

第二步：行移位

- 移动规则：
  - 第一行不变
  - 第二行左移动1个字节
  - 第三行左移动2个字节
  - 第四行左移动3个字节

- 实现了列间的置换

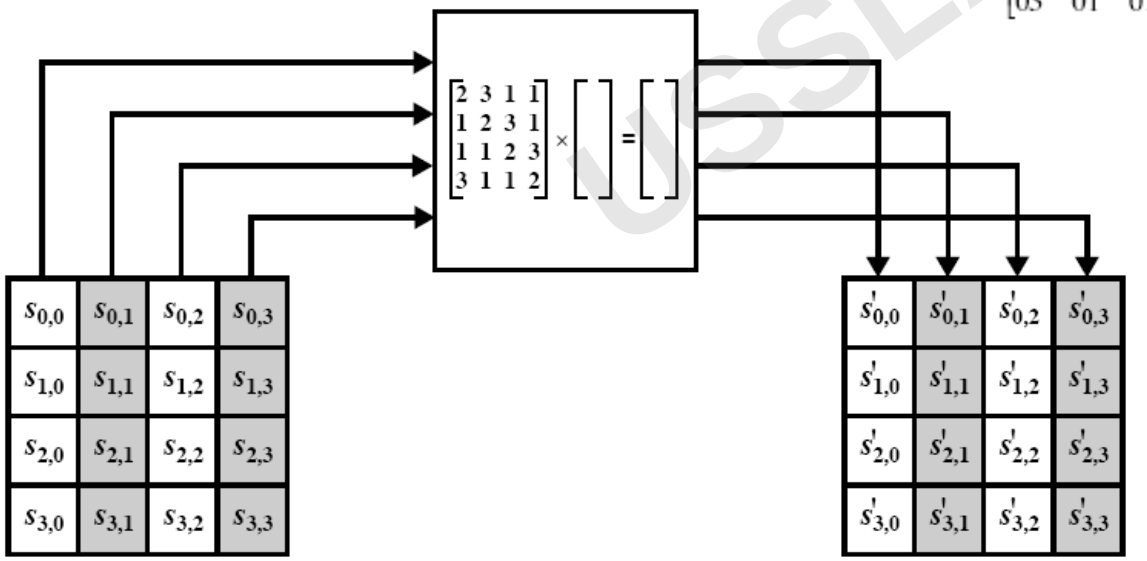


(a) Shift row transformation

第三步：列混淆

- 每一列单独处理
- 每列中每个字节被替换成**本列中所有4个字节都相关**的结果，实现**扩散效果**
- 基于伽罗瓦域运算

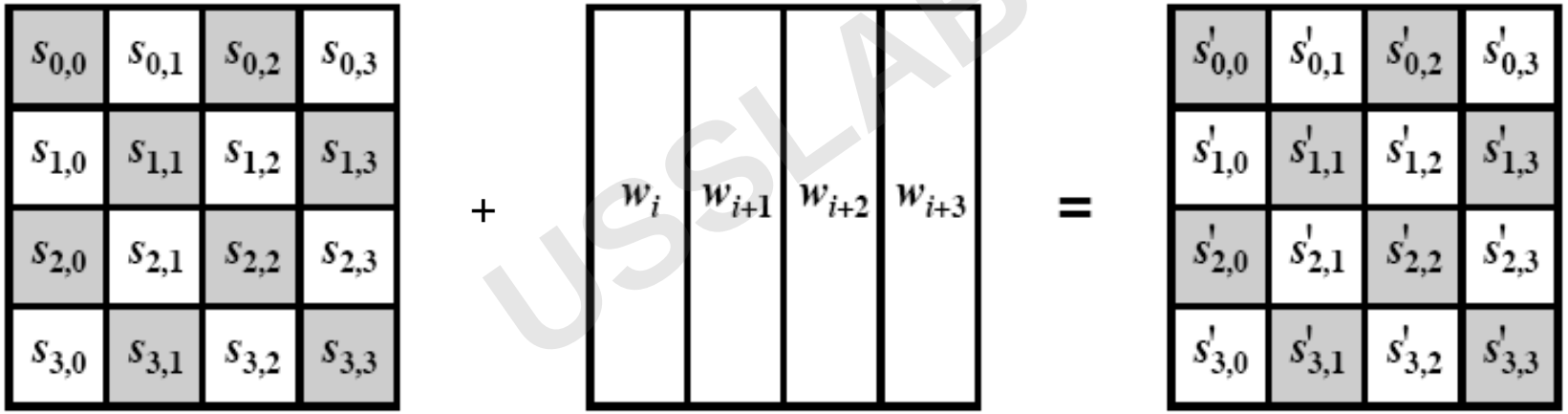
$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$



(b) Mix column transformation

第四步：轮密钥加

- 将16字节的当前状态矩阵和长度为16字节的子密钥进行异或操作。



(b) Add Round Key Transformation

AES加密例子

- **已知:**

明文：32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34

密钥：2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c

- **计算密文**

AES加密例子

- **1. 先进行1次轮密钥加**

明文

32	88	31	e0
43	5a	31	37
f6	30	98	07
a8	8d	a2	34

⊕

密钥

2b	28	ab	09
7e	ae	f7	cf
15	d2	15	4f
16	a6	88	3c

=

19	a0	9c	e9
3d	f4	c6	f8
e3	e2	8d	48
be	2b	2a	08

AES加密例子

- **2. 字节代替**

19	a0	9c	e9
3d	f4	c6	f8
e3	e2	8d	48
be	2b	2a	08

S盒代替

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

AES加密例子

- **3. 行移位**

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

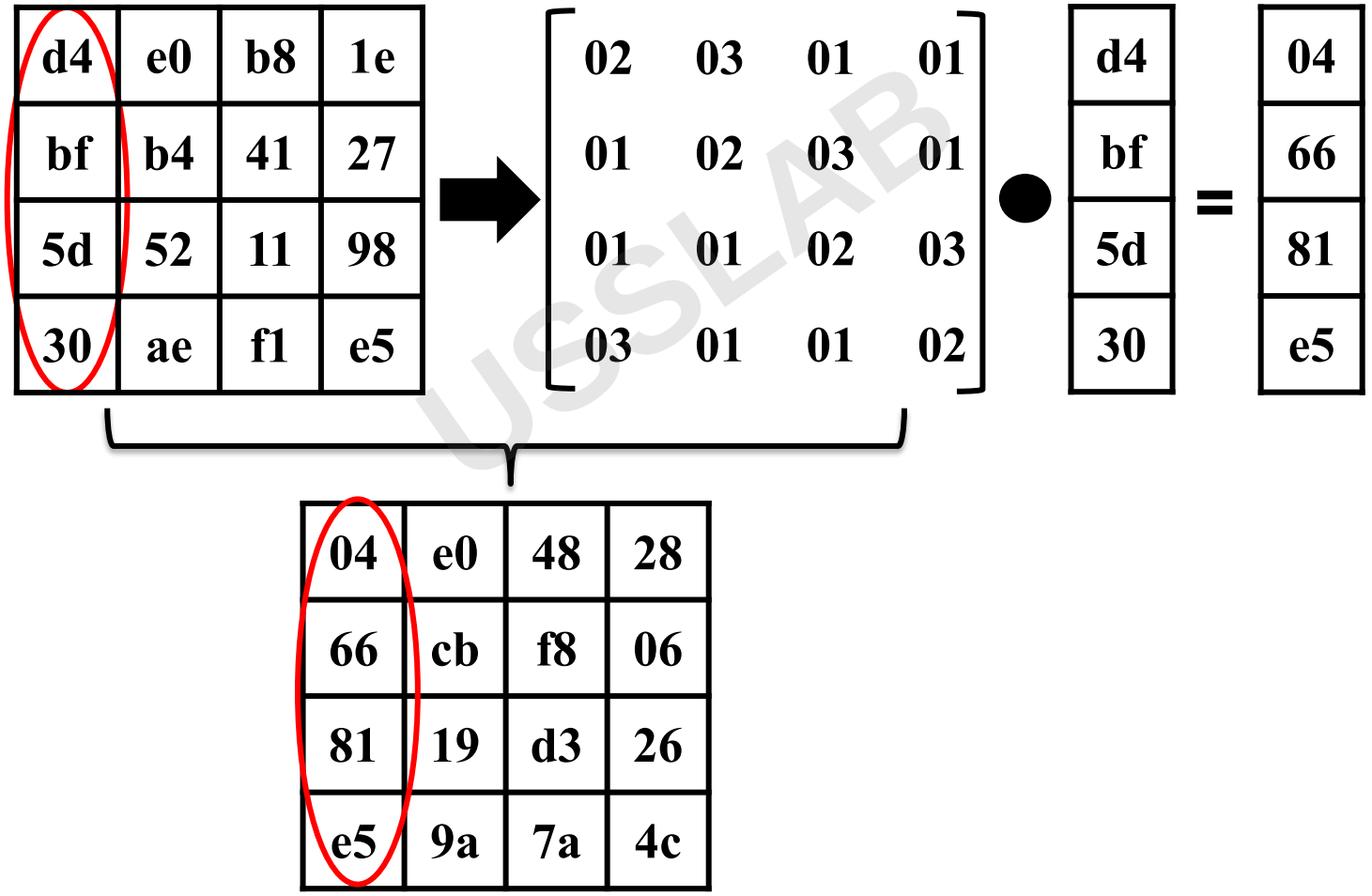
每一行分别滚动
0、1、2、3次

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

AES加密例子

- **4. 列混淆**

以第一列为例



AES加密例子

- **5. 轮密钥加**

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

⊕

a0	88	23	2a
fa	54	a3	6c
fe	2c	39	76
17	b1	39	05

W_i W_{i+1} W_{i+2} W_{i+3}

=

a4	68	6b	02
9c	9f	5b	6a
7f	35	ea	50
f2	2b	43	49

AES加密例子

- **6. 重复10次步骤2~5**
  - **加密后的密文**

32	88	31	e0
43	5a	31	37
f6	30	98	07
a8	8d	a2	34

明文

39	02	dc	19
25	dc	11	6a
84	09	85	0b
1d	fb	97	32

密文

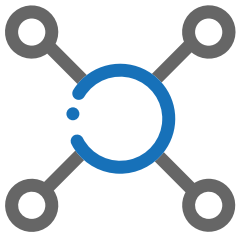
3.2 Asymmetric Cryptograph

公钥密码概念（非对称密钥）

对称密钥体制缺陷

- **安全性**：对称密码体制(例如DES, AES) 需要两个用户利用提前共享的秘密来建立“安全密钥”，然而通信双方共享秘密并不容易。
- **实用性**：一个具有N个用户的团体，如果用户两两之间都需要进行安全通信，采用对称密码体制来保护用户之间的通信：每个用户需要与其余的N-1个用户共享私钥，整个系统需要管理**N(N -1)/2**个密钥。
- 无法保证可追溯性（accountability），Why?

Q：怎么办？回到锁头和钥匙……



密钥分发信道不得公开

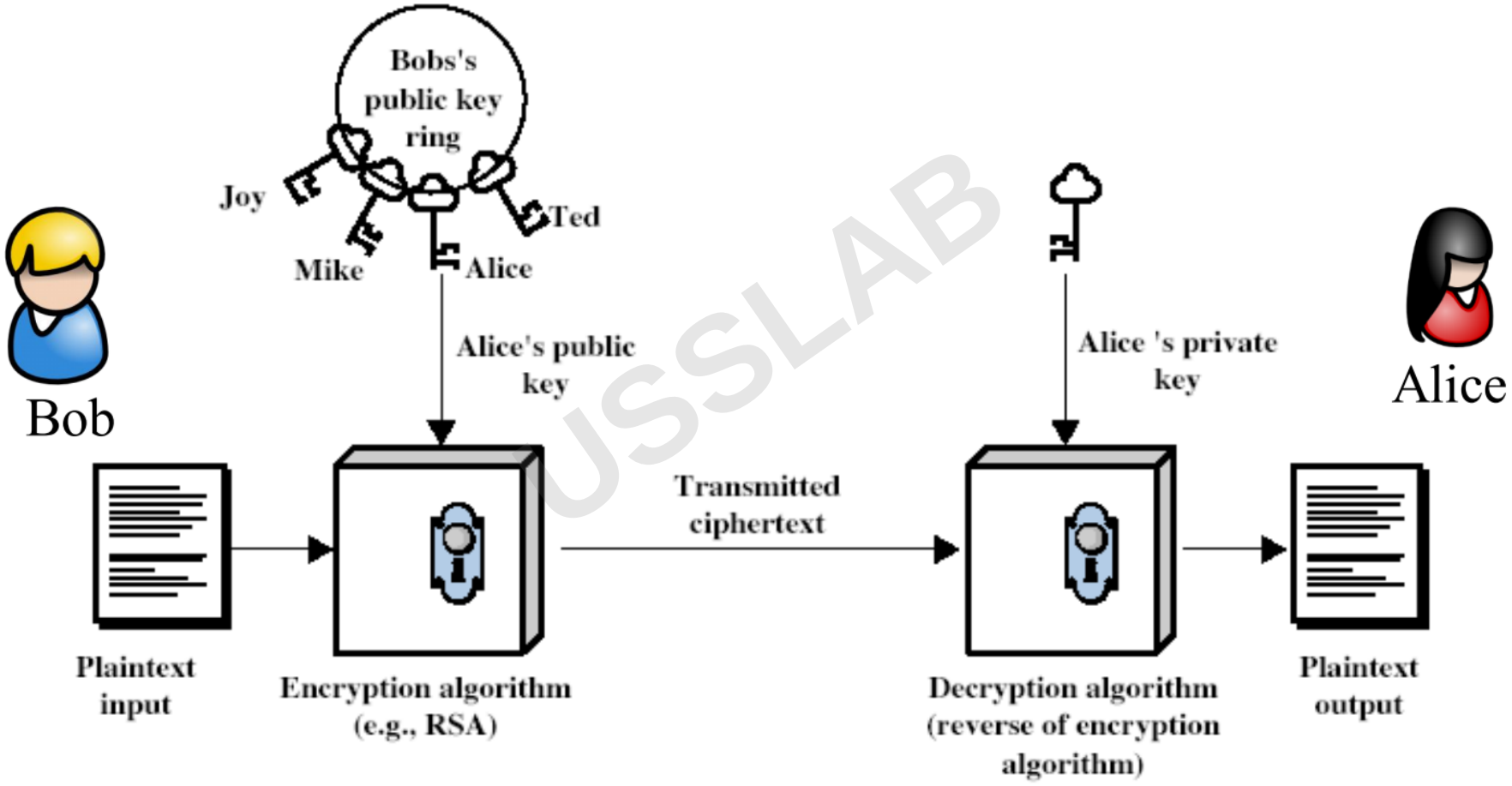
密钥管理成本高

公钥密码体制历史

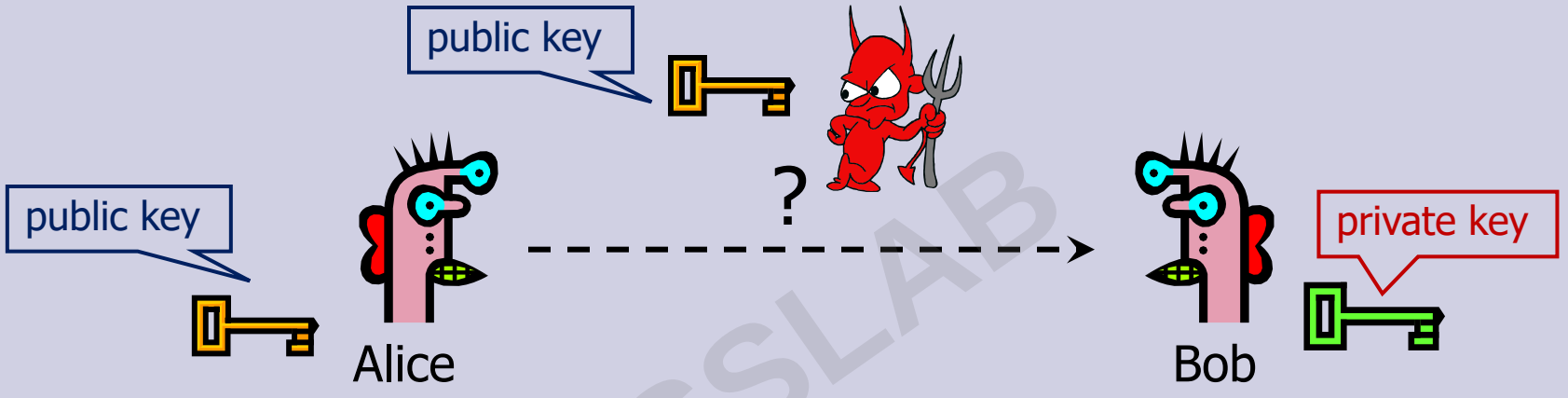
- 1976年 Diffie和Hellman提出“公钥密码”这一概念，2015年获得ACM图灵奖
- 1977年Ronald Rivest, Adi Shamir, Leonard Adleman提出RSA算法，首次实现了公钥密码体制

Diffie　　Hellman

公钥密码体制基本思想



公钥密码体制简介



公钥p_k：在网络上公开，由其他用户使用

私钥s_k：由用户本人使用

如何保证安全：一些问题呈现“非对称性”，即从一个方向计算非常容易，而从另一个方向计算则很困难。即**加密容易解密难！**

例如：计算任意给定整数的乘积很容易，而计算给定大整数的因子分解则非常困难。

RSA-768表示如下：

```
1230186684530117755130494958384962720772853569595334792197322452151726400507263655187452021997864693899564749427740638459251925573263034537315482685079170261221429134616704292143116022212404792747377940806653514195974598569021434133

= 33478071698956898786044169848212690817704794983713768568912431388982883793878002287614711652531743087737814467999489×3674604366679959042824463379962795263227915816434308764267603228381573966651127923337341714339681027009279873630891
```

公钥密码体制简介

密钥生成：通过相对容易的计算过程生成一对公钥p_k与私钥s_k。

- 如果仅获得公钥 p_k，得到私钥s_k的操作在计算上是不可行的。

加密：给定明文M与公钥 p_k，很容易计算得到密文 $C=Ep_k(M)$

解密：给定密文 $C=Ep_k(M)$ 和私钥 s_k，很容易计算得到明文 M

- 如果没有私钥s_k，从密文C中是**不可以**计算得到明文M的。

PS：“不可以”是指在当前的计算能力条件下，计算时间和消息长度呈指数增长关系。

优势：

开放系统

没有预先建立关系的用户也能通过对方的公钥建立安全通信

密钥分发

公钥能够采用公开(认证的)信道进行传输

密钥管理

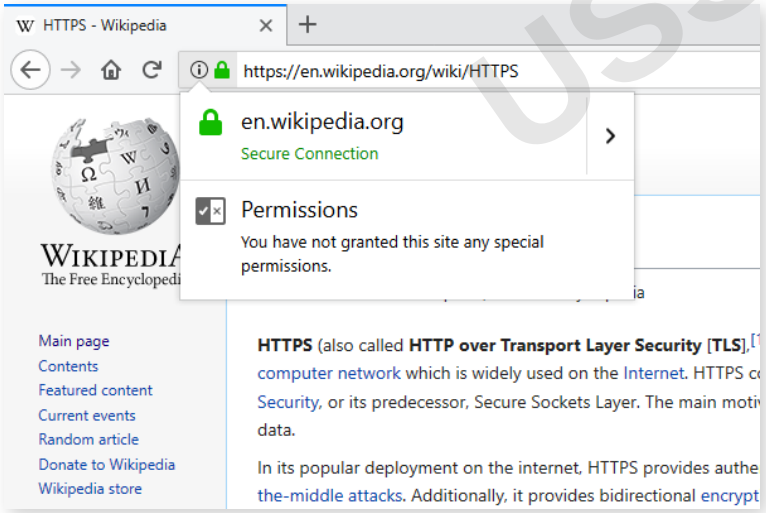
N人系统中，整个系统仅仅需要维护N个公钥

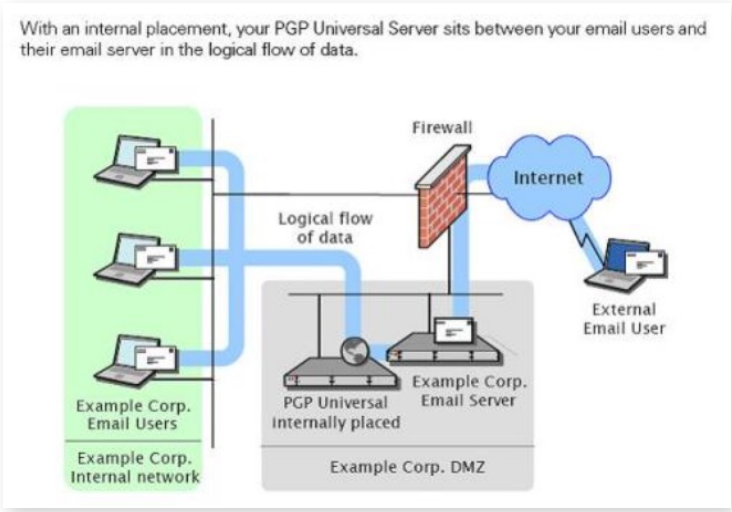
公钥密码体制应用

- **HTTPs**：Hyper Text Transfer Protocol over Secure Socket Layer
  在HTTP协议下加入SSL，提升安全性，可用于敏感信息的通讯。

- **PGP**：Pretty Good Privacy：secure E-mail

PS：公钥加密通常可以用于私钥加密中密钥传输，想想为什么?





3.2 RSA Encryption Algorithm

RSA加密算法

RSA公钥密码体制历史

- 1977年，麻省理工学院Ron Rivest、Adi Shamir和Leonard Adleman提出RSA加密算法。
- 2002年，获得图灵奖。

RSA公钥加密体制原理

密钥生成:

1. 选择两个大素数p, q (例如：每个数字1024位)
2. 选择 n = pq，z = φ(n) = (p-1)(q-1)
3. 随机选取 e（其中e<n），e与z没有公约数。 //gcd(e,z)=1
4. 选取 d 使得 ed-1 能够被z完全整除。 //ed = 1 mod z
5. **公钥是 (n, e)，私钥是（d）。**

RSA举例

1. Select primes: p=61 & q=53
2. Compute $n = pq$ = 3233
3. Compute $\varphi(n)=(p-1)(q-1)$ = 3120
4. Select e : gcd(e, $\varphi(n)$)=1; choose e = 17
5. Determine d: $de = 1 \bmod \varphi(n)$ and $d < \varphi(n)$.

   17*d - 3120*k = 1

   私钥：**d = 2753, (k=15)**

   公钥：**n = 3233, e=17**

(PS: without p and q, calculating d is computationally infeasible.)

加解密举例：

6. Suppose m=32
7. Ciphertext $c=m^e \bmod n = (32)^{17} \bmod 3233=1992$
8. Reconstruct plaintext: $m=c^d \bmod n = 32$

RSA公钥加密体制原理

加密/解密算法:

如上所述给出（n,e）和（d）

加密：由$c = m^e \bmod n$将明文m转变为密文c（即：当m^e模n所得的余数）。

注意：m<n（如果需要，则分块）

解密：$m = c^d \bmod n$（即：c^d模n所得的余数）。

核心思想： $m = (\underbrace{m^e \bmod n}_{c})^d \bmod n$

RSA原理证明

- 由**欧拉定理**得出

  当gcd$(a,n)=1$时（a, n互素），$a^{\varphi(n)} \bmod n = 1$.

- 在**RSA**中：

  1. $n = p \cdot q$

  2. $\varphi(n) = (p-1)\cdot(q-1)$

  3. 选择整数 e 和 d，d 为 e 关于模 $\varphi(n)$ 的逆元/模反元素

  4. 因此，$e \cdot d = 1 + k \cdot \varphi(n)\ (k > 0, k \epsilon Z)$

  对于密文C，$C^d = (M^e)^d \bmod n = M^{1+k\varphi(n)} \bmod n = M*(M^{\varphi(n)})^k \bmod n$

  $= M*(1)^k \bmod n = M \bmod n$

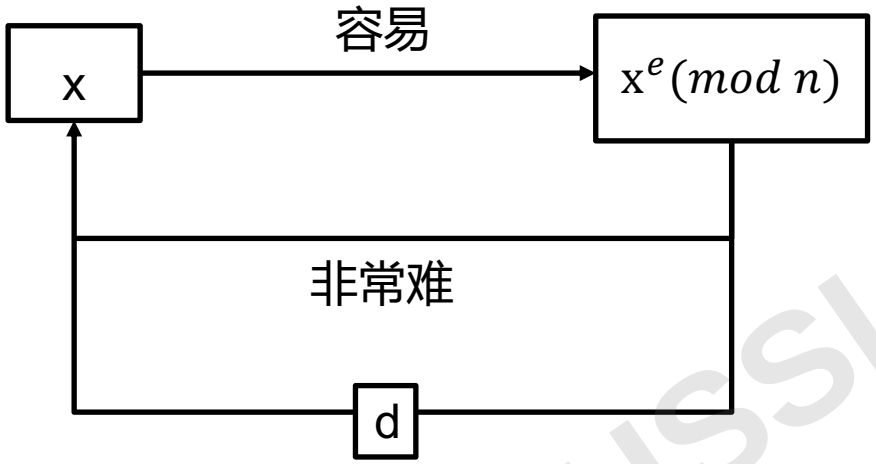
  Q：为什么?

如何证明 $(M^{\varphi(n)})^k \bmod n = 1$

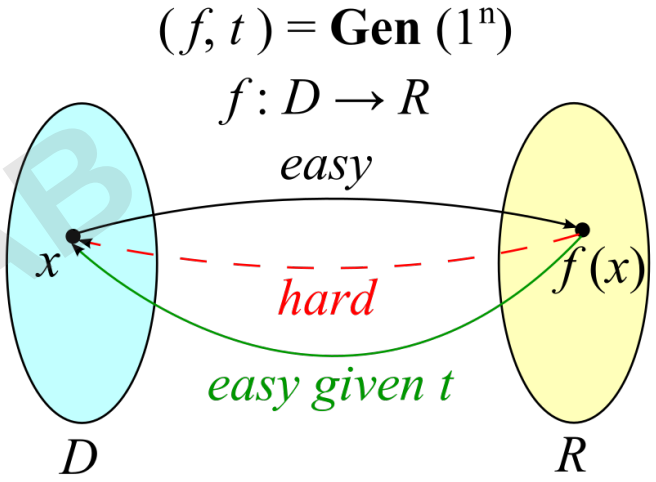
- 情况1：gcd(M,n)=1
- 情况2：gcd(M,n)≠1

1. 由于p、q均为素数，n=p*q，因此，M必须满足如下一种：
2. M = rp, 或者 M = sq
3. 假设 M= rp（意味着gcd(M,q)=1），因此 $(M^{\varphi(q)})^k \bmod q = 1$
4. $(M^{\varphi(n)})^k \bmod q = (M^{(p-1)(q-1)})^k \bmod q$
5. $= (M^{(p-1)\varphi(q)})^k \bmod q = ((M^{\varphi(q)})^k)^{p-1} = 1 \bmod q$
6. 因此，
7. $(M^{\varphi(n)})^k = 1 \bmod q \rightarrow (M^{\varphi(n)})^k = 1+uq$，所以
8. $M* (M^{\varphi(n)})^k = M+M*u*q = M+r*p*u*q = M+r*u*n$
9. $M* (M^{\varphi(n)})^k = M \bmod n$
10. 证毕！

RSA安全性本质



d 是陷门函数



陷门函数示意

问题本质：已知n，求$n = pq$，即数的**素分解问题**。

这一问题的时间复杂度是$e^{\sqrt{\ln(n)ln(ln(n))}}$

一般情况下，破解RSA密钥成为计算上的不可解问题。

RSA安全性分析

穷举法攻击

- 攻击者设计一个M，$C=M^e \bmod n$。对d进行暴力破解，因为$d < \varphi(n)$，所以d的个数至多有n-1个
- 设p，q分别为100位（十进制），则n-1约200位（十进制），所以 $n=10^{200}$
- 假定每秒可以做一亿次搜索（10^8），每年可以搜索 $10^8*60*60*24*365=3*10^{15}$

搜索10^{200}个密钥的时间为$10^{200}/(3*10^{15})=3*10^{185}$**年!**

计算上不可行!

RSA安全性

- 2009年12月12日，编号为RSA-768（768 bits, 232 digits）数也被成功分解（目前被破解最长的）
- 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

```
RSA-768 = 123018668453011775513049495838496272077285356959533479219732245215172640050726365751874520219978646938995647494277406384592519255732630345373154826850791702612214291346167042921431160222124047927473779408066535141959745985690214
          3413
```

```
RSA-768 = 33478071698956898786044169848212690817704794983713768568912431388982883793878002287614711652531743087737814467999489
        × 36746043666799590428244633799627952632279158164343087642676032283815739666511279233373417143396810270092798736308917
```

https://en.wikipedia.org/wiki/RSA_numbers

RSA安全性

- RSA-2,048 有 617 个十进制数字（2048 位）
- 如果出现大规模可用的量子计算机（比如运行Shor算法分解大整数），RSA-2048将不再安全。但截至2025年，还没有达到能够威胁2048位RSA的量子计算机诞生。
- 建议使用RSA-3072。

```
RSA-2048 = 25195908475657893494027183240048398571429282126204032027777137836043662020707
           75955562640185258807844069182906412495150821892985591491761845028084891200728
           44499268739280728777673597141834727026189637501497182469116507761337985909570
           00097330459748808428401797429100642458691817195118746121515172654632282216869
           99875491824224336372590851418654620435767984233871847744479207399342365848238
           24281198163815010674810451660377306056201619676256133844143603833904414952634
           44321901146575444541784240209246165157233507787077498171257724679629263863563
           73289912154831438167899885040445364023527381951378636564391212010397122822120720357
```

如何建立密钥?

- 通信过程中如何建立安全的会话密钥是一个难题。
- 如果通信双方物理上不在一起，如何建立一个共享密钥?
- 密钥建立的两种方法:
  - 密钥约定（Key Agreement）
  - 密钥分发（Key Distribution）

密钥约定—DH密钥交换协议

Diffie-Hellman算法是第一个公开密钥算法，于1976年提出。该算法可以使两个用户之间安全地交换一个密钥，但不能用于加密或解密信息

- DH实现了双方在不安全通信信道下交流，如何得到一个共同密钥。
- 例如：商用加密算法AES和DES需要在安全通信之前，实现通信双方的对称密钥共享。

New Directions in Cryptography

Invited Paper

Whitfield Diffie and Martin E. Hellman

Abstract Two kinds of contemporary developments in cryptography are examined. Widening applications of teleprocessing have given rise to a need for new types of cryptographic systems, which minimize the need for secure key distribution channels and supply the equivalent of a written signature. This paper suggests ways to solve these currently open problems. communications over an insecure channel order to use cryptography to insure privacy, however, it currently necessary for the communicating parties to share a key which is known to no one else. This is done by sending the key in advance over some secure channel such a private courier or registered mail. A private conversation between two people with no prior acquain-

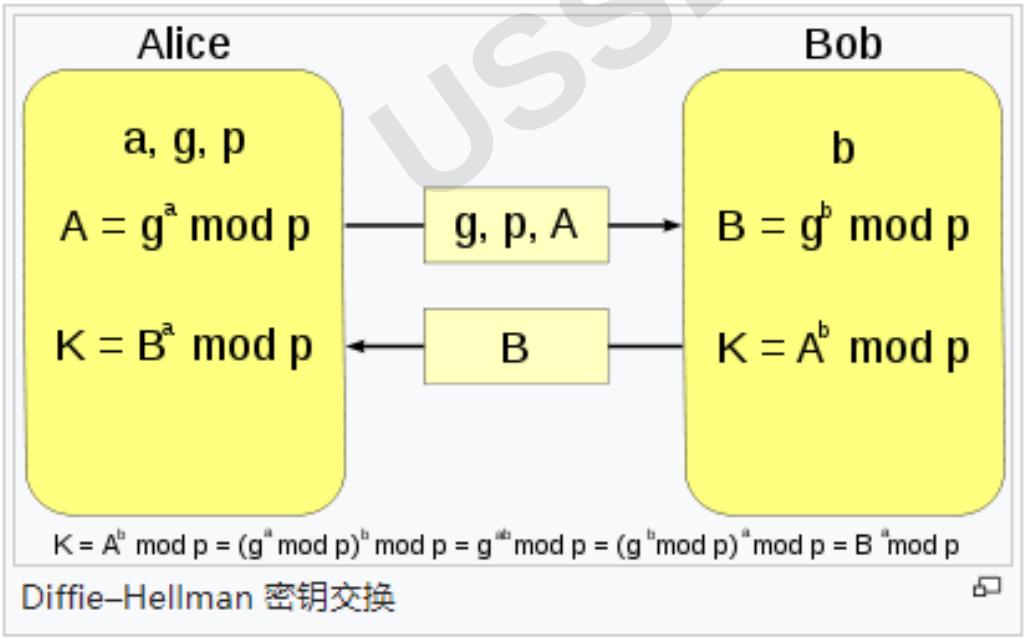
DH协议基本原理

- 回顾RSA加密，你能想到什么方法吗？
- 给定双方A,B，则A,B通过如下过程建立共享密钥：

$$A \to B: g^a \bmod p \qquad B \to A: g^b \bmod p$$

$$A: \left(g^b\right)^a \bmod p = g^{ab} \bmod p \qquad B: \left(g^a\right)^b \bmod p = g^{ab} \bmod p$$

g^{ab}为共享密钥

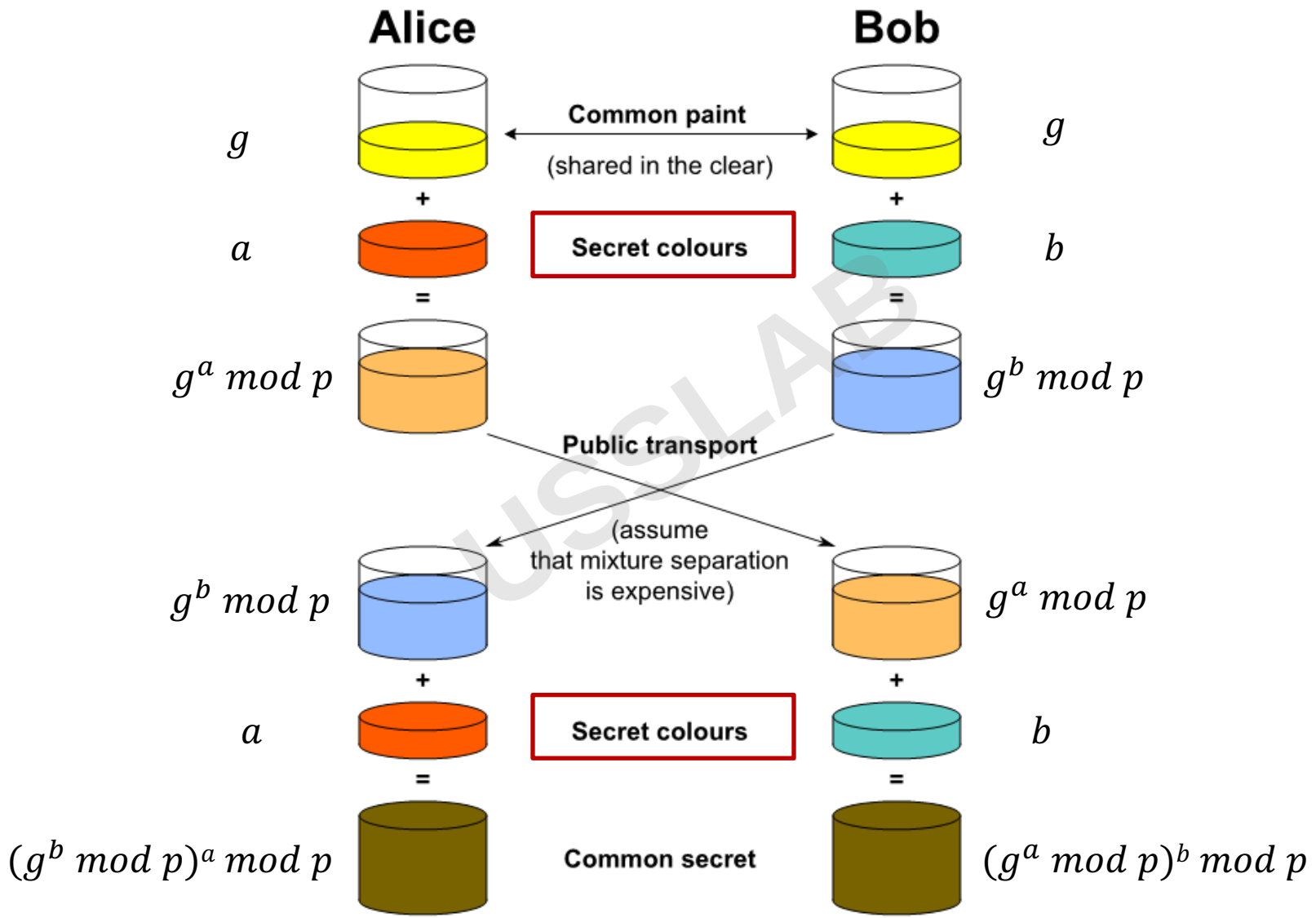


Q：左边过程中，哪些是保密的，哪些是公开的？

DH密钥交换协议示例

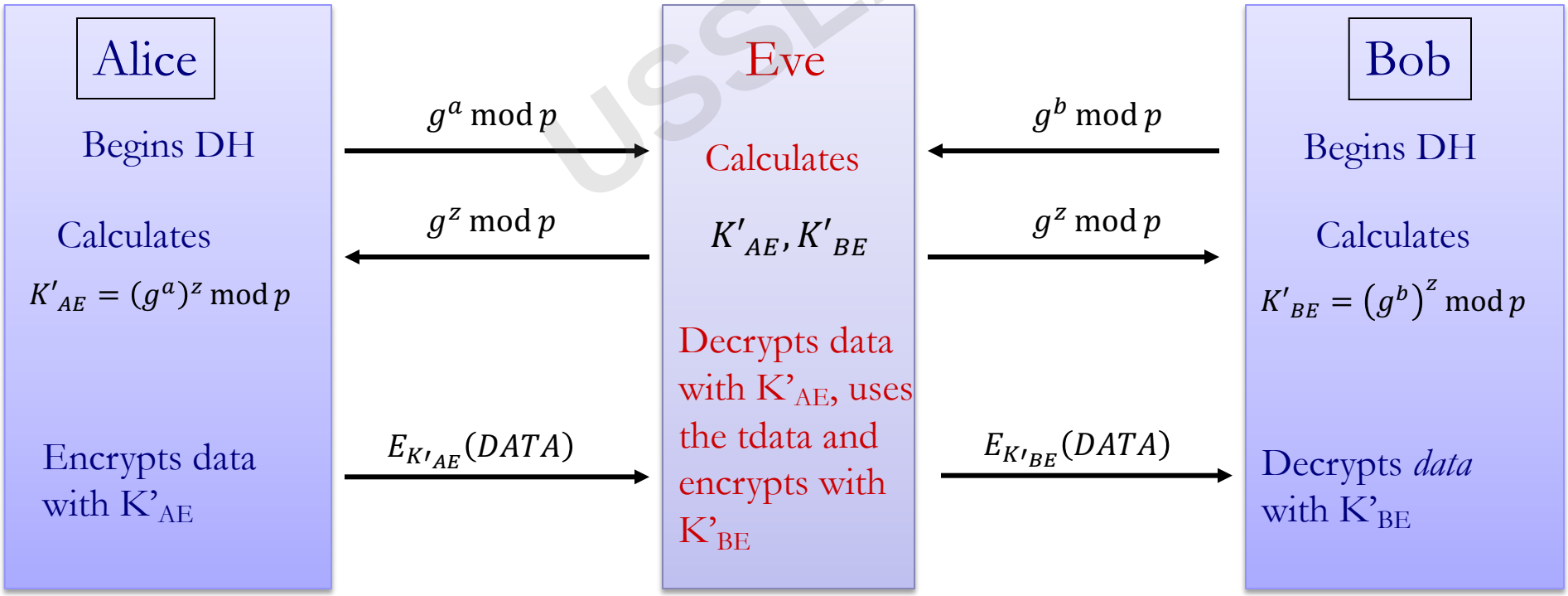
1. Alice和Bob协定使用素数$p = 23$，base $g = 5$.
2. Alice选择要给秘密整数：$a = 6$; 计算$A = g^a \ mod \ p$并发送给Bob
   - $A = 5^6 \ mod \ 23 = 8$
3. Bob选择要给秘密整数：$b = 15$; 计算$B = g^b \ mod \ p$并发送给Alice
   - $B = 5^{15} \ mod \ 23 = 19$
4. Alice计算$s = B^a \ mod \ p$
   - $19^6 \ mod \ 23 = \mathbf{2}$.
5. Bob计算$s = A^b \ mod \ p$
   - $8^{15} \ mod \ 23 = \mathbf{2}$.

爱丽丝				鲍伯		
秘密	非秘密	计算		计算	非秘密	秘密
	p, g		→		**p, g**	
a						**b**
		$\mathbf{g^a}$ mod p			...	
	...		←	$\mathbf{g^b}$ mod p		
	$(\mathbf{g^b}$ mod p$)^{\mathbf{a}}$ mod p		=		$(\mathbf{g^a}$ mod p$)^{\mathbf{b}}$ mod p	


Alice
Bob
g
Common paint
(shared in the clear)
g
+
a
Secret colours
b
+
=
=
$g^a \bmod p$
$g^b \bmod p$
Public transport
(assume
that mixture separation
is expensive)
$g^b \bmod p$
$g^a \bmod p$
+
+
a
Secret colours
b
=
=
$(g^b \bmod p)^a \bmod p$
Common secret
$(g^a \bmod p)^b \bmod p$

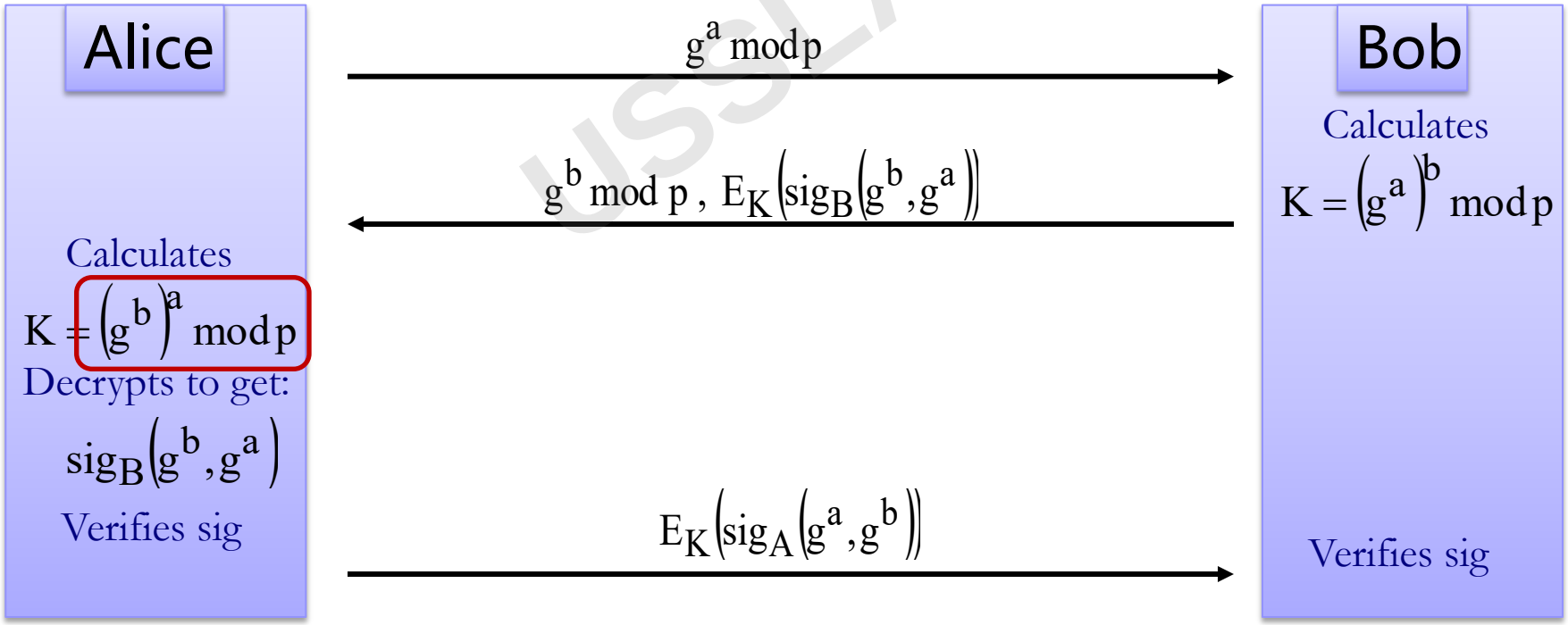
DH密钥交换中的安全问题

- 容易遭受**阻塞攻击**：因为幂运算是计算密集性的，当攻击者发起大量的密钥请求，受攻击者将花费较大计算资源来做幂运算；
- 容易遭受**中间人攻击**：攻击者可分别冒充用户A和B中的一方，与另一方交换密钥（可以监听和传递A和B的秘密信息而不被发现）。



防御DH中间人攻击

- 数字签名可以用来阻止DH中间人攻击
- Private sig( ) 函数：数字签名
- Public ver( ) 函数：身份验证



Diffie加盟浙大

3.3 Domestic encryption algorithm

国密算法

3.3.1 国密算法

- **定义**：国家密码局认定的国产密码算法。主要有SM1，SM2，SM4等，密钥长度和分组长度均为128位。

- **算法分类**：
  - SM1为**对称加密**，加密强度与AES相当，算法不公开，使用时调用加密芯片的接口，对标AES。
  - SM2为**非对称加密**，是基于椭圆曲线的加密算法，对标RSA，但是签名和密钥生成速度都快于RSA。
  - SM4为**对称加密**，对标3DES，是无线局域网标准的分组数据加密算法。

3.3.2 国密算法与国际算法

- **SM1算法**： SM1是我国自主研制的对称加密算法，算法不公开仅存在于加密芯片中。SM1安全性和软硬件实现性能与AES相当。
- **SM1与AES**：

	SM1	AES
计算结构	基于椭圆曲线	代换置换网络
存储空间长度	128位	128位
加密解密速度	约为n*10M/s	约为n*10M/s

3.3.2 国密算法与国际算法

- **SM2算法**：SM2是我国自主研制的公钥密码算法，包括SM2-1**数字签名算法**、SM2-2**密钥交换协议**、SM2-3**加密算法。**
- SM2与RSA：

	SM2	RSA
计算结构	基于椭圆曲线	基于可逆模幂运算
复杂度	指数级	亚指数级
加密解密速度	约为M/s	约为0.1M/s
相同安全性所需公钥长度	较少（160位SM2与1024位RSA安全性接近）	较多

3.3.2 国密算法与国际算法

- **SM4算法**：SM4是我国自主设计的分组对称密码算法，是无线局域网标准的分组数据算法，密钥长度和分组长度位128位，SM4在计算过程中加入了非线性变换，安全性更高，安全性高于3DES。
- **SM4与DES**：

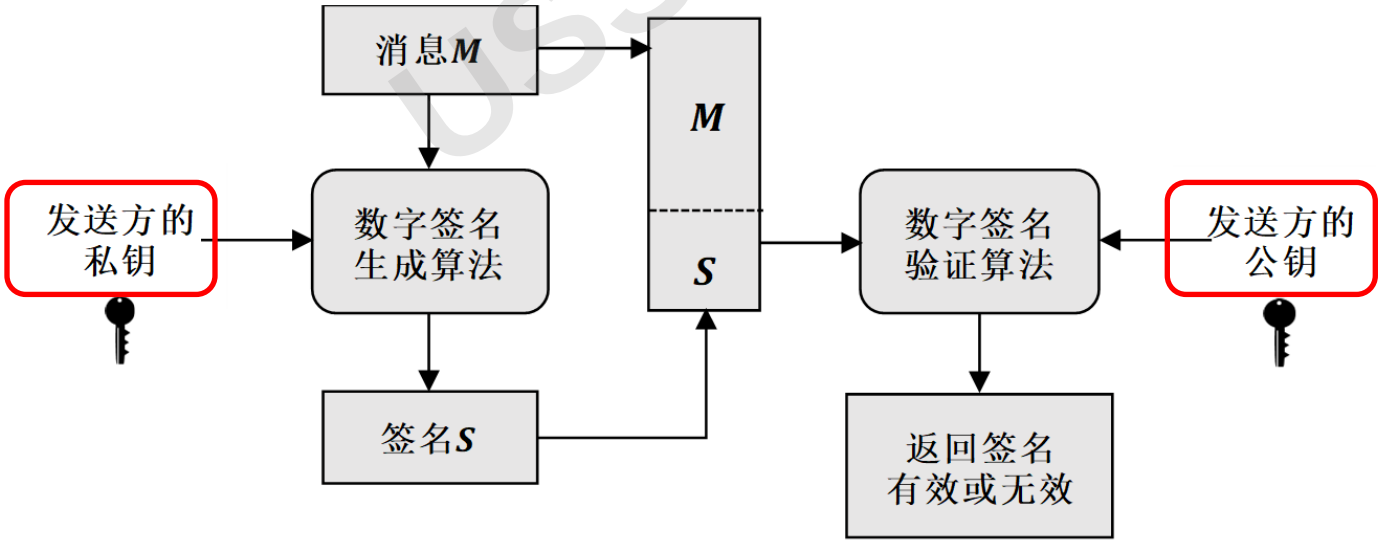
	SM4	DES(3DES)
计算轮数	32轮	16轮 （3DES48轮）
分组长度	128位	64位
密钥长度	128位	64位（3DES为128位, $k_1=k_3$）
实现性能	软件硬件实现性能都好	软件实现较慢，硬件实现较快
安全性	较高	较低（3DES较高）

3.4 Digital signature and message authentication

数字签名与消息认证

3.4.1 数字签名

- **信息安全目标之一：如何保证消息真实性、不可抵赖性?**
- 数字签名（又称公钥数字签名）：只有信息发送者才能产生的别人无法伪造的一段数字串，类似于手写签名，可保证每个用户都可验证信息来源，是对信息的发送者身份真实性的一个有效证明，保证消息的真实性和不可抵赖性。



数字签名流程图

基于RSA算法的数字签名

- **RSA公钥算法数字签名：**
- **加密过程**

  1. 参数选择和密钥生成
  2. 签名过程：用户A对信息M进行签名，采用**私钥 d 加密**，计算
  3. S=Sig(M)=$M^d mod\ n$
  4. M+S

和RSA加密中用公钥e加密相反

- **验证过程：**用户B验证用户A对信息M的数字签名，计算

$$M' = \text{Ver(S)} = S^e \bmod n$$

M'=M?

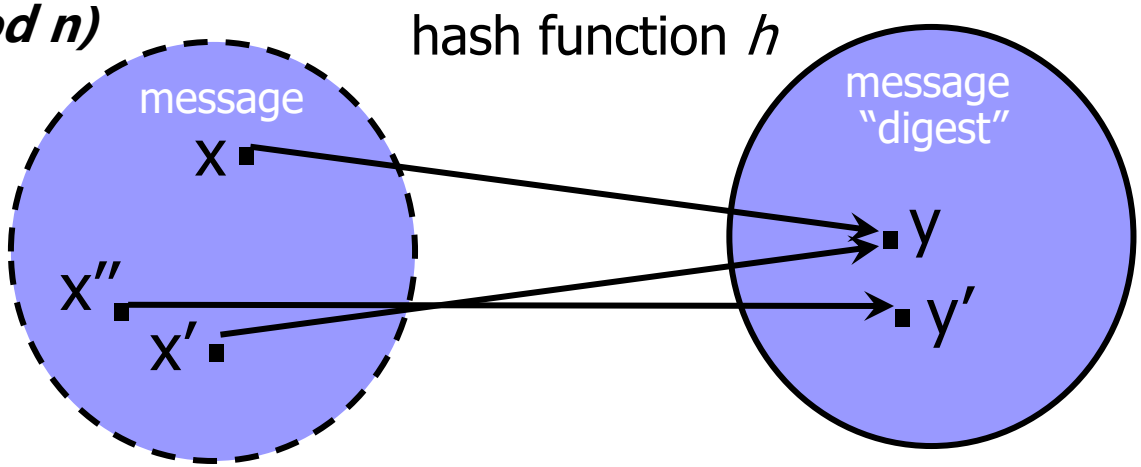
相等则身份正确

- 除了RSA数字签名算法之外，还有DSA算法，Elgamal等
- 相比于其他数字签名算法，RSA数字签名安全等级低，计算速度慢，因此已经逐渐被替代。

3.4.2 哈希/散列函数

- **信息安全另一目标：如何保证消息的完整性?**
- **哈希函数定义**
  - **哈希函数**（Hash function）又称**散列算法、散列函数**，是一种从任何一种数据中创建**小的数字“指纹”**的方法。
  - 散列函数把消息或数据压缩成摘要digest，使得数据量变小，将数据的格式固定下来。

举例: *h(m) = m (mod n)*

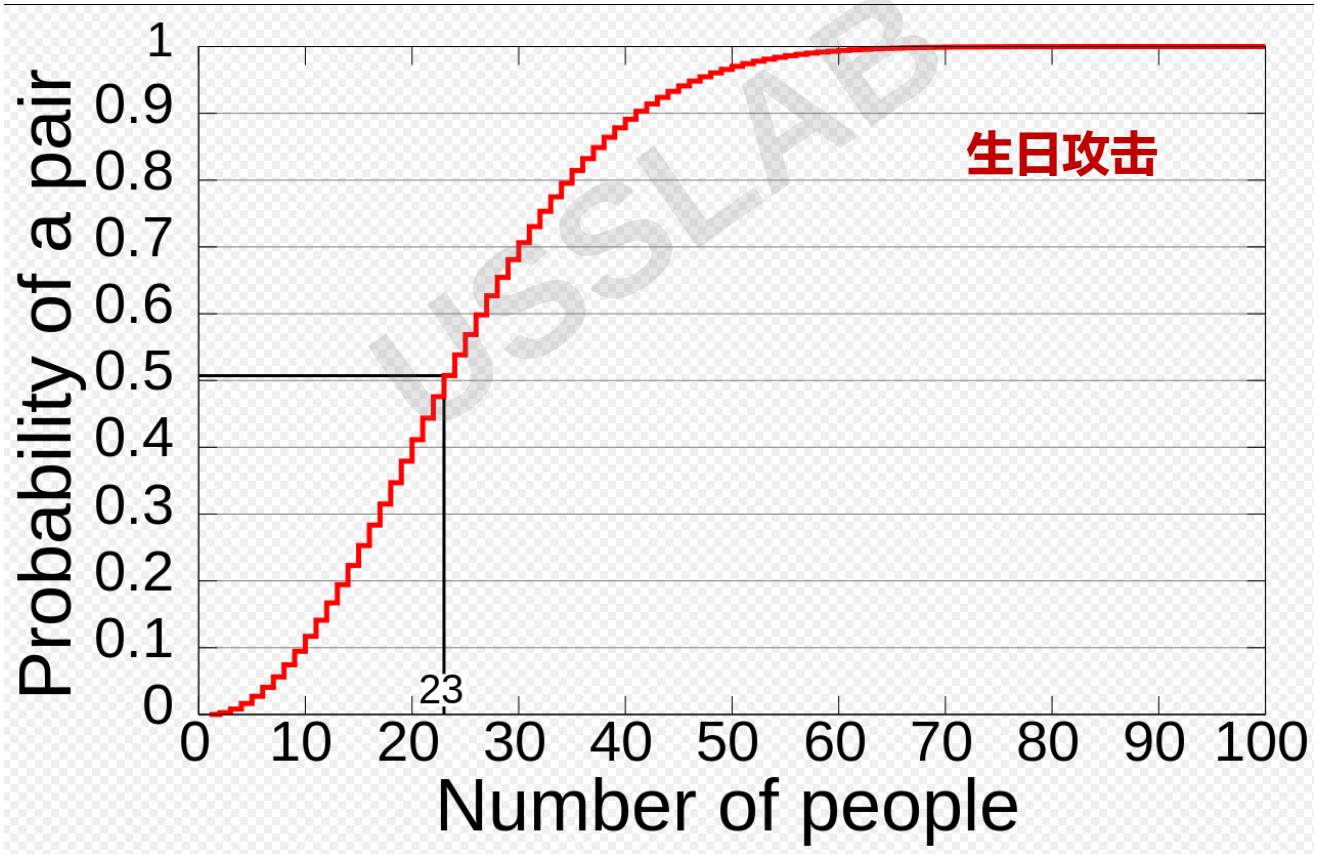


3.4.2 哈希/散列函数

- **哈希函数需满足的特性：**
  - **输入长度可变**：对任何长度的输入x都适用
  - **输出长度固定**：对任何长度的输入x，输出z都是固定长度
  - **效率**：计算复杂度低
  - **抗原像攻击（单向性）**：对于给定输出y，不可能找到对应的输入x
  - **抗第二原像攻击**：对于给定的x_1，找到满足$h(x_1)=h(x_2)$的x_2是不可能

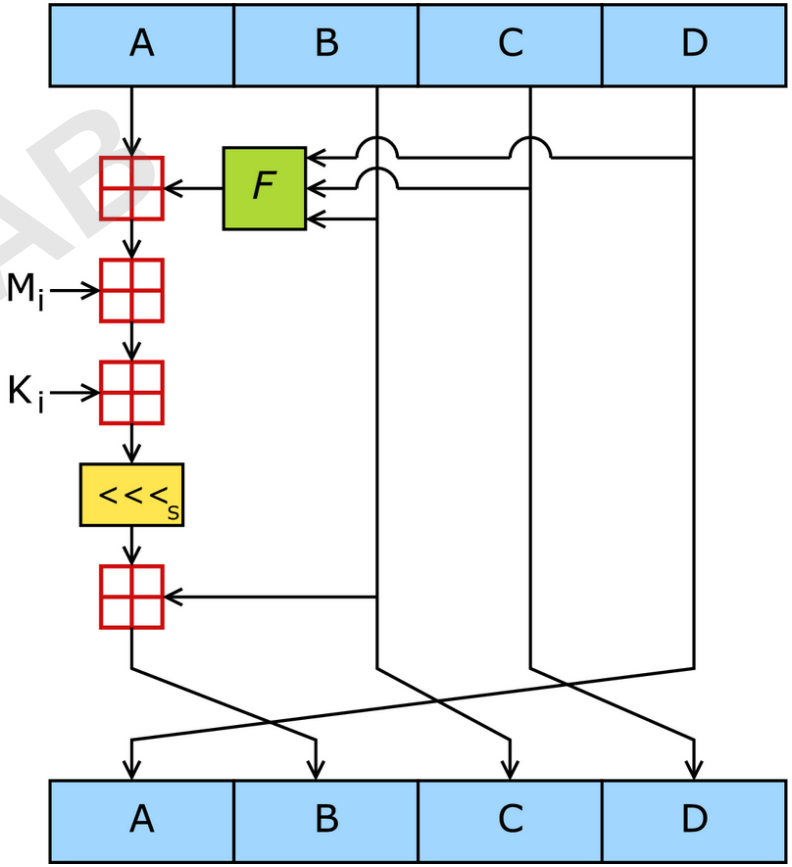
生日攻击

- **哈希算法中的碰撞（第二原像攻击）**
  - 会出现$\mathrm{h}(x_1)=\mathrm{h}(x_2)$



3.4.2 哈希/散列函数——具体算法

- **哈希函数MD5**
  - **MD5** Message Digest
    消息摘要算法
  - MD5是由密码学家Rivest在1992年公布的单向散列函数
  - 2004年夏天发现碰撞
- **MD5特点：**
  - 输入消息可以是任意长
  - 每一次迭代处理512bits的消息分组
  - 最终输出散列值为**128bits**



举例: BitTorrent下载片段完整性

3.4.2 哈希/散列函数

- **MD5的碰撞案例**

In [1]:
```
import hashlib

# 两段HEX字节串，注意它们有细微差别
a = bytearray.fromhex("0e306561559aa787d00bc6f70bbdfe3404cf03659e704f8534c00ffb659c4c8740c

b = bytearray.fromhex("0e306561559aa787d00bc6f70bbdfe3404cf03659e744f8534c00ffb659c4c8740c

# 输出MD5，它们的结果一致
print(hashlib.md5(a).hexdigest())
print(hashlib.md5(b).hexdigest())
```

```
cee9a457e790cf20d4bdaa6d69f01e41
cee9a457e790cf20d4bdaa6d69f01e41
```


- 事实上我们的智能手机中每过数秒就可以找到一个MD5碰撞案例
- MD5不被推荐作为应用中的算法方案，取代它的是SHA（Secure Hash Algorithm）家族算法

3.4.2 哈希/散列函数——具体算法

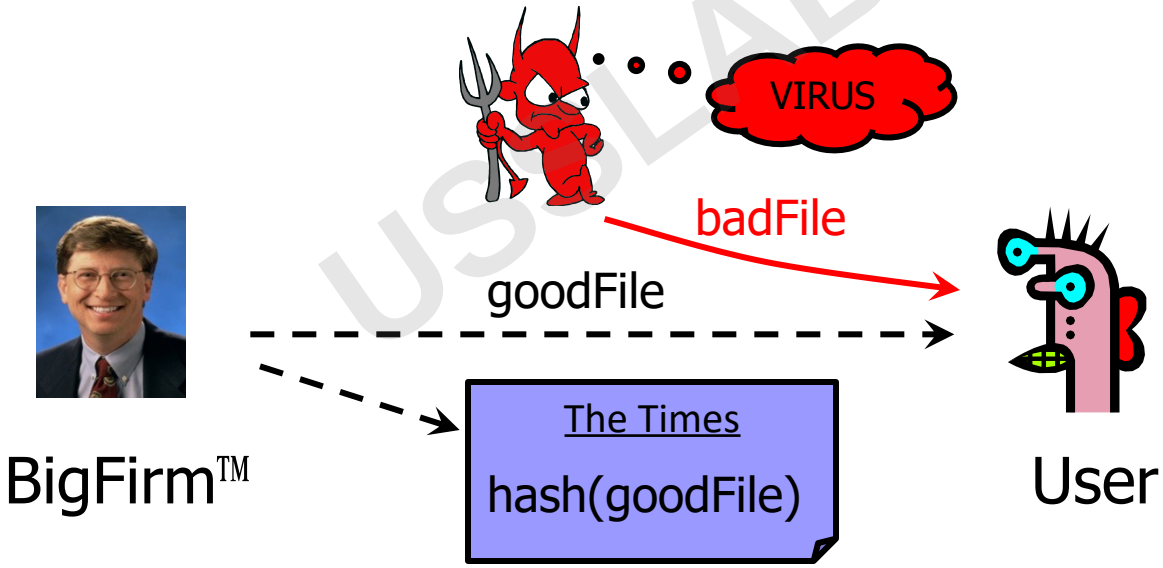
- **SHA(Security Hash Algorithm)**
  - 安全哈希算法
  - 最大消息长度$<2^{64}$
  - 消息分组长度512bits
  - 输出散列长度**160bits**
  - **2005年夏天发现碰撞。**

SHA家族与MD5的对比

算法和变体		输出散列值长度(bits)	中继散列值长度(bits)	数据区块长度(bits)	最大输入消息长度(bits)	循环次数	碰撞攻击(bits)	性能实例(MiB/s)
MD5		128	128 (4*32)	512	无限	64	<64 (发现碰撞)	335
SHA-0		160	160 (5*32)	512	2^{64}-1	80	<80 (发现碰撞)	-
SHA-1		160	160 (5*32)	512	2^{64}-1	80	<80 (发现碰撞)	192
SHA-2	SHA-224 SHA-256	224 256	256 (8*32)	512	2^{64}-1	64	112 128	139
	SHA-384 SHA-512 SHA-512/224 SHA-512/256	384 512 224 256	512 (8*64)	1024	2^{128}-1	80	192 256 112 128	154
SHA-3	SHA3-224 SHA3-256 SHA3-384 SHA3-512	224 256 384 512	1600 (5*5*64)	1152 1088 832 576	无限	24	112 128 192 256	-
	SHAKE128 SHAKE256	d(arbitrary) d(arbitrary)		1344 1088			Min(d/2, 128) Min(d/2, 256)	-

3.4.3 消息认证码

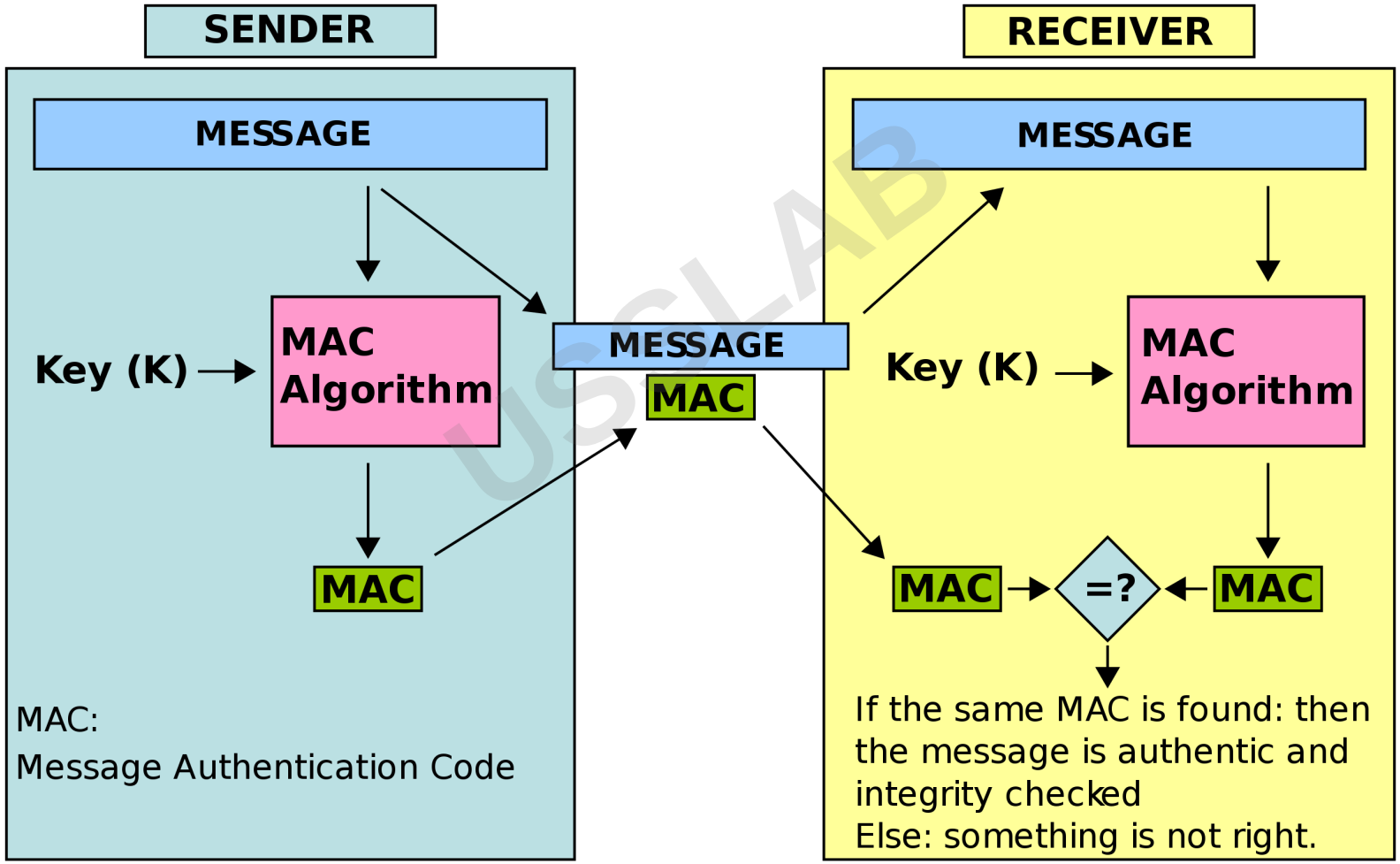
- **完整性保证保护方法：消息认证码（MAC，message authentication code）**
- 例如：同时对消息和密钥进行哈希，防止对消息进行篡改



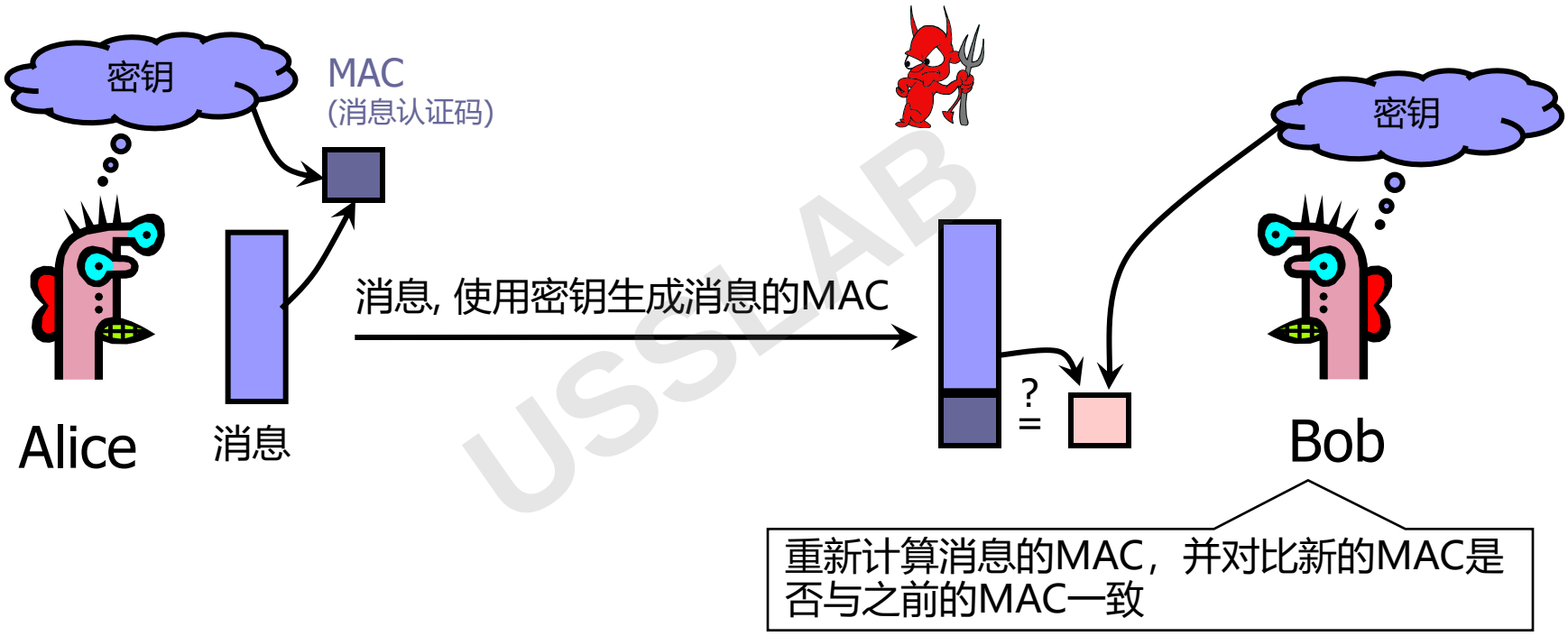
$$MAC = H(message + key)$$

H是一个哈希函数，‘+’代表连接运算

3.4.3 消息认证码



3.4.3 消息认证码



完整性：只有知道密钥KEY的人才能计算出给定消息的MAC。
认证：通过验证密钥KEY是否有效，来判断发送方真实身份

3.5 Key management and distribution

密钥管理和分发

3.5.1 密钥管理

- **密钥管理（Key management）：**包括密钥的**生成、分发、验证、更新、存储、备份、销毁**等处理，涉及到密码学协议设计、密钥服务器、用户程序，以及其他相关协议。

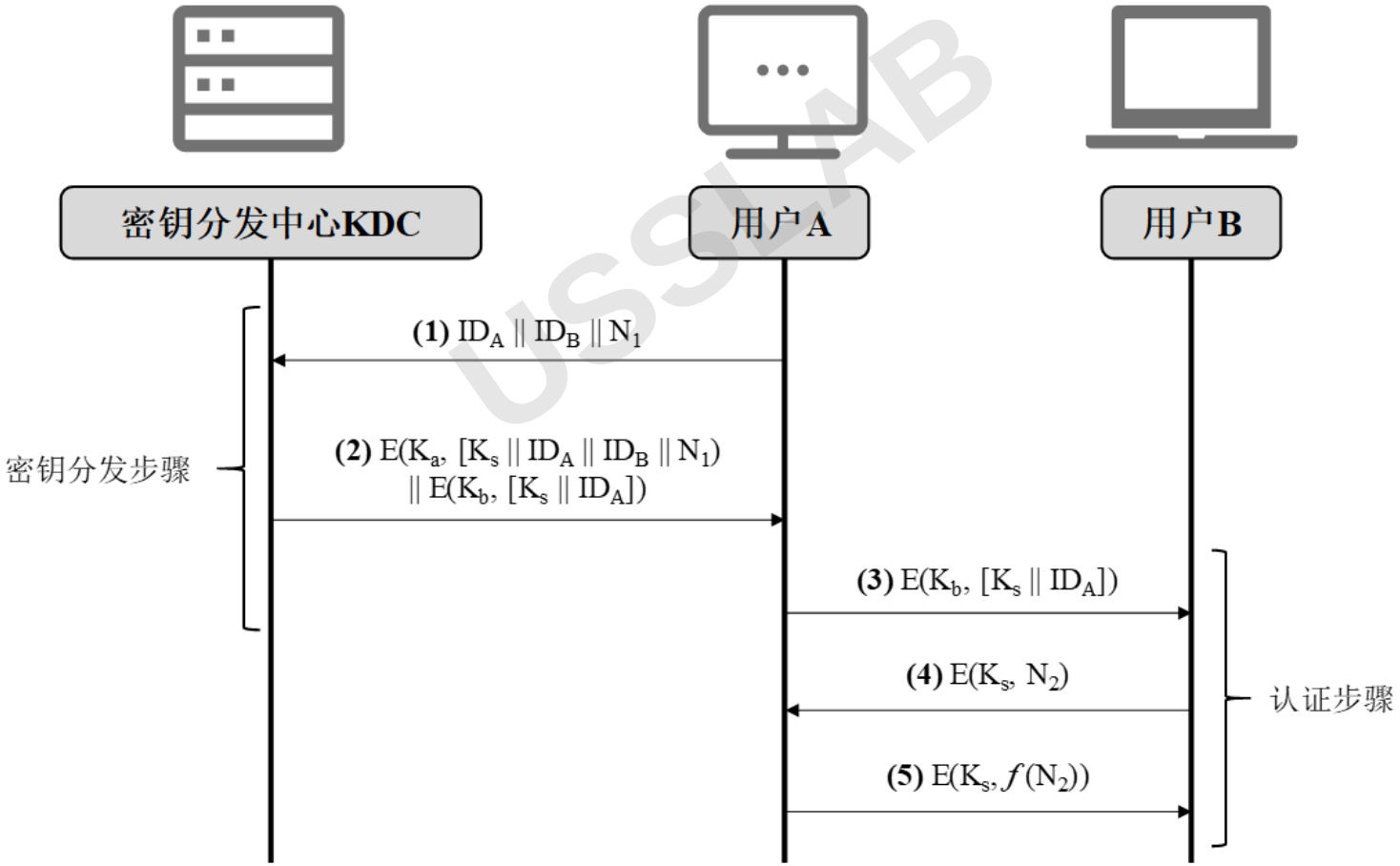
- **密钥保护的基本原则：**
  - 密钥永远不可以以明文的形式出现在密码装置，包括软件硬件之外！

3.5.2 密钥分发方案

- **密钥分发方式：**
  - 方法1：A选择密钥并手工传递给B
  - 方法2：第三方C选择密钥分别手工传递给A,B
  - 方法3：用A,B原有的共享密钥传送新密钥
  - 方法4：与A,B分别有共享密钥的第三方C传送新密钥给A和/或B
- **方案优劣：**
  - 方法1, 2：手动交付密钥，需支持的通信对密钥数量极高。
  - 方法3：若攻击者成功获得一个密钥，则后续密钥将全部泄露
  - 方法4成本较低，已被广泛使用

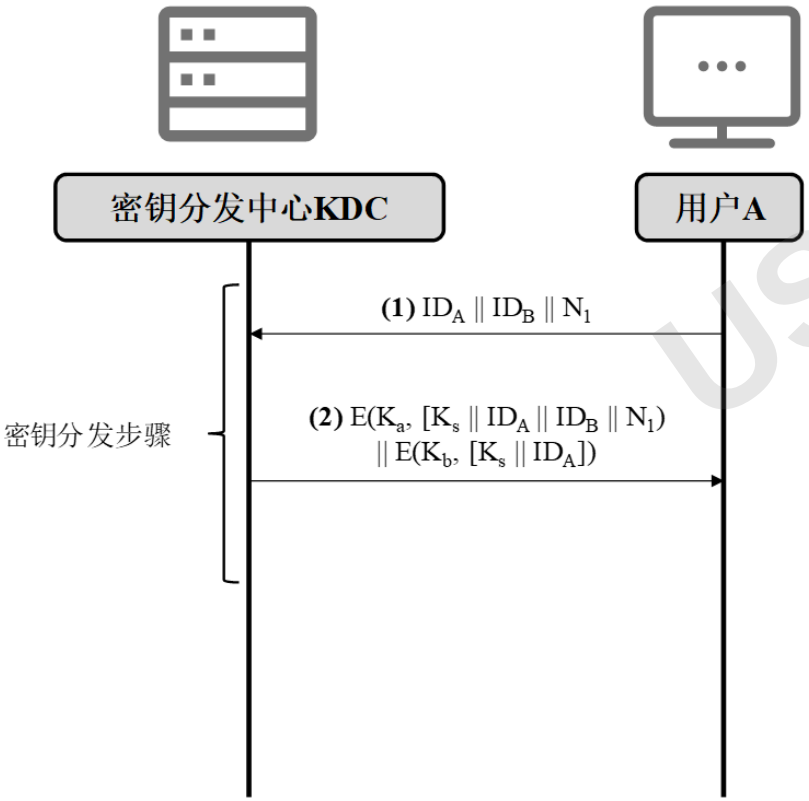
3.5.2 密钥分发方案

- （1）基于对称加密的对称密钥分发



3.5.2 密钥分发方案

- （1）基于对称加密的对称密钥分发



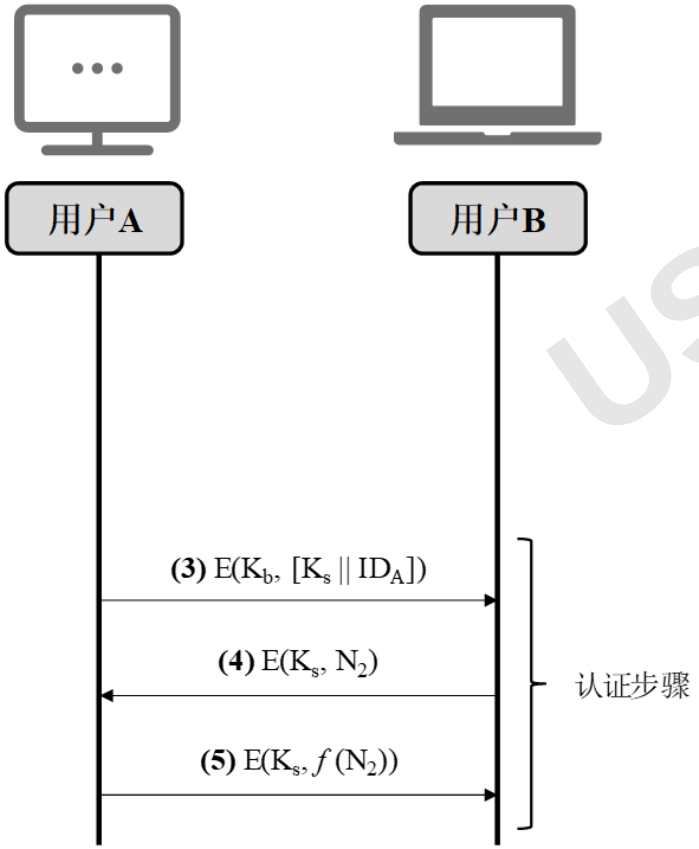
① **A向KDC**发送一个包括了A、B身份信息及本次传输的临时交互号N_1(作为此次传输的标志)的数据包。

② **KDC将返回给A**以下被K_a加密过的信息：

- 用于A、B通信的一次性会话密钥K_s
- 步骤①中的请求信息
- 密钥K_b加密过的会话密钥K_s与A的身份信息，这部分将发送给B，将用于B验证A的身份与通信连接建立。

3.5.2 密钥分发方案

- （1）基于对称加密的对称密钥分发



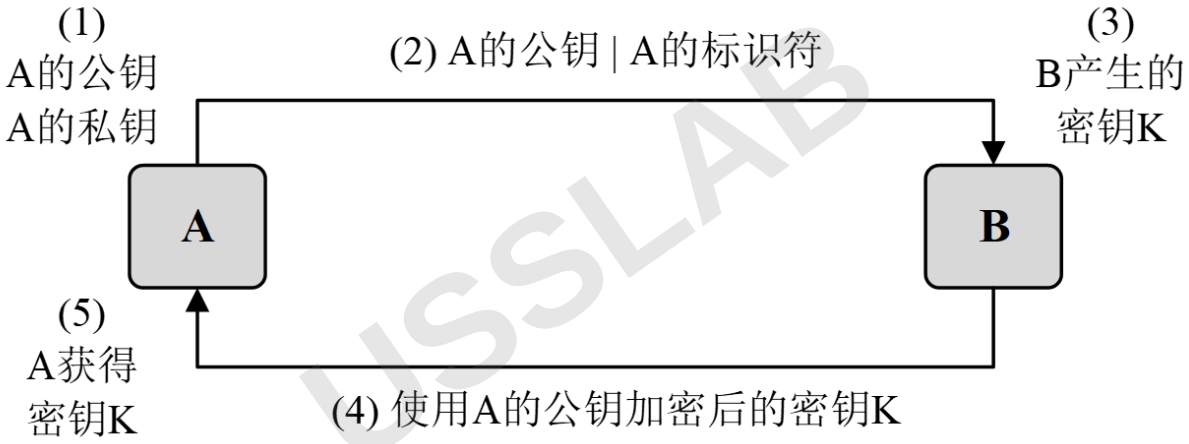
③ **A**获得步骤②中的所有信息，并**将属于B的信息转发给B**。B收到信息后可用其主密钥K_b解密信息，获得会话密钥与A的身份信息。由于仅有KDC与B掌握此密钥，故该信息可保证安全性。

④ 为防止B在步骤③收到的信息受到重放攻击，**B需要验证A的身份**。B使用会话密钥K_s加密新的临时交互号N_2，将结果发送给A；

⑤ A获得信息，解密得到N_2，并对N_2进行函数变换。最后，对变换值进行加密，发送给B，**完成认证过程。**

3.5.2 密钥分发方案

- （2）基于非对称加密的对称密钥分发



① 用户A产生一个公私钥对；

② 用户A将自己的公钥与标识符发送给想要构建通信联系的用户B；

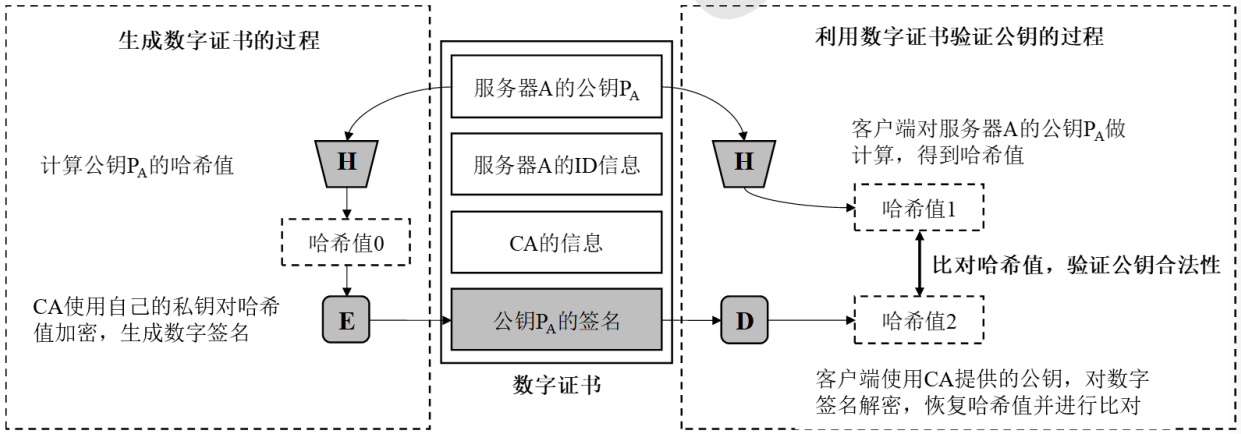
③ 用户B产生对称密钥K；

④ 用户B使用用户A的公钥对密钥K进行加密，发送给用户A

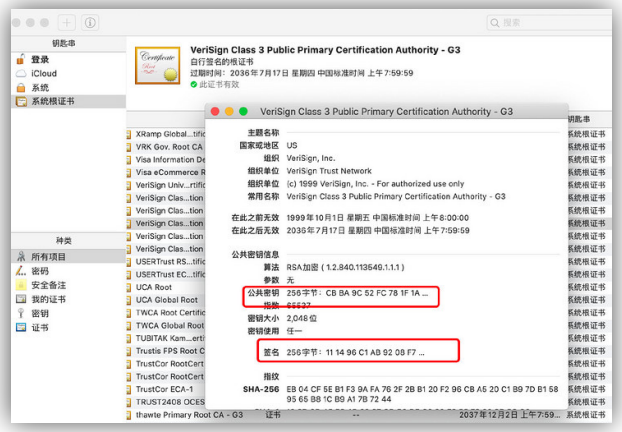
⑤ 只有用户A掌握公钥加密算法中的私钥，故用户A将解密获得密钥K。

3.5.2 密钥分发方案：公钥数字证书

- **公开数字证书** (Public key certificate)：又称“**数字证书**”(digital certificate)，本质上是公钥的数字签名。需要一个可信的第三方CA。
- **CA(Certificate Authority)**：证书颁发机构。核心作用是用其私钥对请求方的公钥进行签名，生成公钥数字证书
- **问题举例**：公钥被冒充，例如有人冒充“招商银行的公钥”
- 假设CA为中国人民银行：
  - CA用其私钥对真实的招商银行的公钥进行数字签名得到招商银行的数字证书；
  - 用户使用CA公钥验证所有声称招商银行公钥的合法性（需要使用U-key等方法，提前将中国人民银行公钥安装在电脑上）



数字证书生成和验证流程



数字证书举例

3.6 Cryptography in IoT scenarios

物联网场景中的密码学

3.6.1 密码学面临的挑战

- **物联网系统的限制：**

- **硬件资源受限，已有加密算法难以部署**

  处理能力、电池寿命、通信带宽和内存限制

- **系统节点数多，安全威胁危害性大且隐蔽**

  设备数远超传统IT环境，安全漏洞繁多

- **业务场景复杂，传统安全体系结构适用性不足**

  具体业务场景与安全需求多变

3.6.1 应对措施

加密轻量化

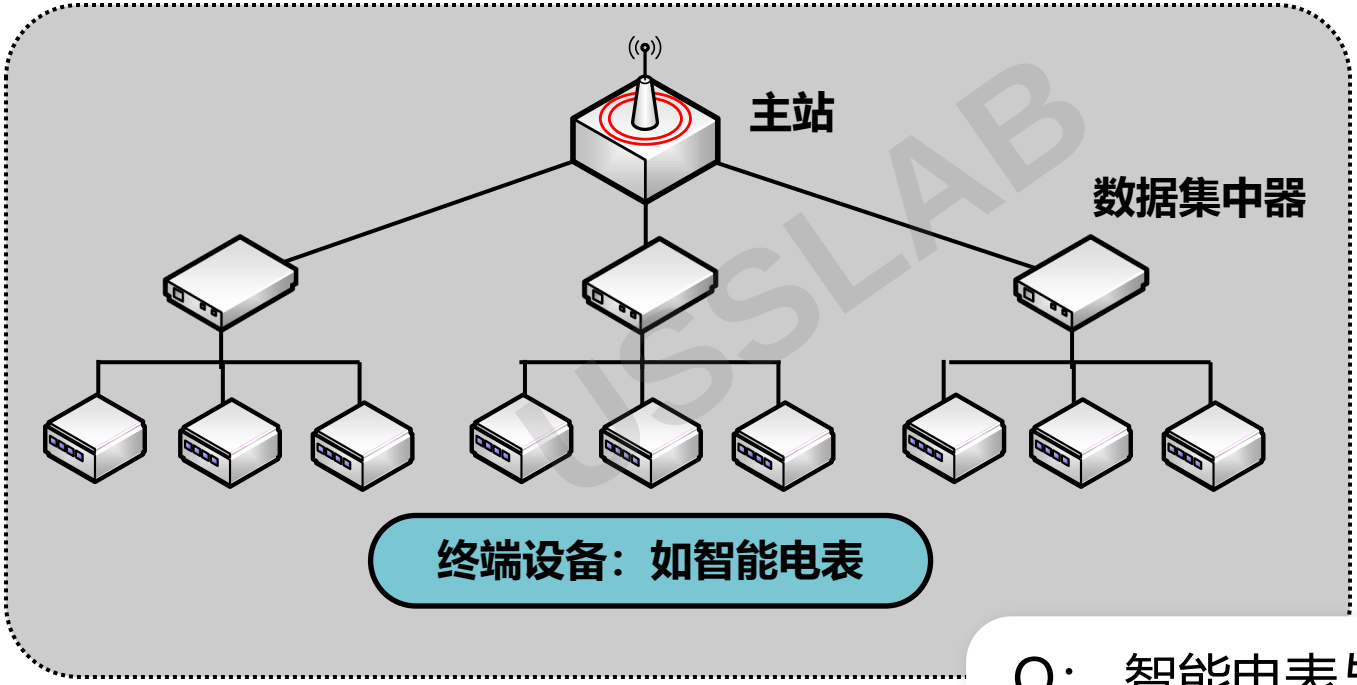
1. **减小密钥长度**：如PRESENT、Twine两类常用算法，物联网场景中均使用80位密钥长度
2. **减小加密轮数**：如Hummingbird算法，仅4轮加密，降低运算时间
3. **改进或重新设计结构**：如DES的变体DESL使用新的S盒代替之前结构的所有S盒，可降低算法实现所需的门电路数量

算法	密钥长度	加密轮数	结构
AES	128/192/256	10/12/14	SPN
HEIGHT	128	32	GFS
PRESENT	80/128	31	SPN
RC5	0-2040	1-255	Feistel
TEA	128	64	Feistel
XTEA	128	64	Feistel
LEA	128/192/256	24/28/32	Feistel
DES	54	16	Feistel
Seed	128	16	Feistel
Twine	80/128	32	Feistel
DESL	54	16	Feistel
3DES	56/112/168	48	Feistel
Hummingbird	256	4	SPN
Iceberg	128	16	SPN
Pride	128	20	SPN

轻量级加密算法一览

3.6.2 应用案例

- 智能电网中的终端通信安全机制设计

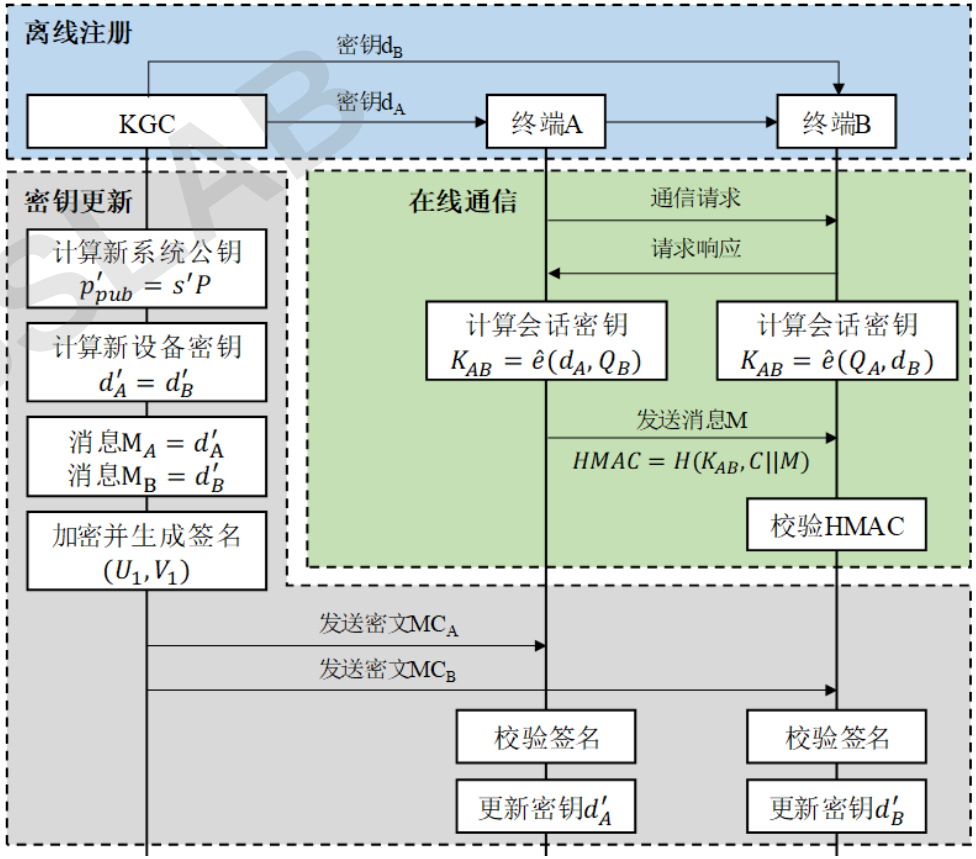


Q： 智能电表与数据集中器之间的通信过程需要加密，如何实现？

3.6.2 应用案例

- 智能电网中的终端通信安全机制设计

1. **离线注册**：密钥生成中心为终端设备生成私钥。
2. **在线通信**：终端之间进行密钥协商，获得会话密钥进行加密。（此处的密钥协商机制可使用更具安全意义的国密算法，如SM2）
3. **密钥更新**：通过数字签名实现安全的密钥更新。



本章总结

- 私钥加密原理，及常见算法DES, AES；
- 公钥加密思想，及常见的算法如RSA；
- 私钥加密和公钥加密的不同点及原因；
- 密钥交换协议DH的流程及其安全问题；
- 常见国密算法名称及对标算法；
- 数字签名、消息认证、密钥管理等；

练习题

- 用RSA解密[167, 1915, 130, 591, 1988, 1235, 1032, 2088, 825]（每个数字对应一个字母）
- 公钥(n,e)=(2449,7)
- 求破解私钥d，并解出明文。

示例程序参加课程主页

- **程序框架：**

```
def decrypt(n,e,ciphertext):
    (x, y) = crack(n)
    fai = (x - 1) * (y - 1)
    plaintext = []
    temp = []
    d = int(GetD(e,fai))
    for num in ciphertext:
        num = pow(num, d, n)
        temp.append(num)
        plaintext.append(chr(num))
    return "".join(plaintext)
```


crack(n)对n进行素数分解
GetD(e,fai)计算d值

本章结束

RSA数学基础——剩余类

剩余类定义：给定正整数m，全体整数可按照模m是否同余分为若干两两不相交的集合，使得每一个集合中的任意两个正整数对模m一定同余，而属于不同集合的任意两个整数对模m不同余，每一个这样的集合称为模m的剩余类。

示例：取$m=7$，则模m的剩余类为：

$$[0]=\{\cdots,-14,-7,0,7,14,\cdots\}$$
$$[1]=\{\cdots,-13,-6,1,8,15,\cdots\}$$
$$[2]=\{\cdots,-12,-5,2,9,16,\cdots\}$$
$$[3]=\{\cdots,-11,-4,3,10,\cdots\}$$
$$[4]=\{\cdots,-10,-5,4,11,\cdots\}$$
$$[5]=\{\cdots,-9,-2,5,12,\cdots\}$$
$$[6]=\{\cdots,-8,-1,6,13,\cdots\}$$

RSA数学基础——欧拉函数

剩余类**互素/互质定义**：在模m的一个剩余类当中，如果有一个数与m互素，则该剩余类中所有的数均与m互素，这时称该剩余类与m互素。

欧拉函数定义：与m互素的剩余类的个数称为欧拉函数，记为$\varphi(m)$。
$\varphi(m)$ 等于 Z_m（最大值为 m 的整数集合）当中与 m 互素的数的个数。

示例：取m=12，则与m互素的剩余类有四个

$$[1] = \{\dots, -24, -11, 1, 13, 25, \dots\}$$

$$[5] = \{\dots, -19, -7, 5, 17, 29, \dots\}$$

$$[7] = \{\dots, -17, -5, 7, 19, 31, \dots\}$$

$$[11] = \{\dots, -13, -1, 11, 23, 35, \dots\}$$

$$\boldsymbol{\varphi(12) = 4}$$

特例1：对于任意一个素数m，$\boldsymbol{\varphi(m) = m - 1}$

特例2：如果m可以分解成两个素数相乘，$m = p1 * p2$，则：

$$\boldsymbol{\varphi(m) = \varphi(p1 * p2) = \varphi(p1) * \varphi(p2)}$$

RSA数学基础——欧拉定理

欧拉定理：设m是正整数，$r \epsilon Z_m$，若$\gcd(r,m)=1$，则$r^{\varphi(m)} = 1(mod\ m)$.

说明：

Z_m：最大值为 m 的整数集合；$\gcd(r,m)=1$：r, m两者互素。

$r^{\varphi(m)} = 1(mod\ m)$：即$r^{\varphi(m)}$模m的余数为1。

费马小定理：如果m也是素数，$r \epsilon Z_m$,且$\gcd(r,m)=1$，则$r^{m-1} = 1(mod\ m)$.

欧拉/费马定理举例

- Quiz: 欧拉/费马定理举例: 计算 $2^{43210} \pmod{101}$.
- 解法：通过费马小定理，我们可知$2^{100} = 1 \pmod{101}$.
  - Why?
  - 101是素数，且gcd(2,101)=1, 所以 $2^{100} = 1 \bmod 101$
- 因此, $2^{43210} \equiv (2^{100})^{432}2^{10} \equiv 1^{432}2^{10} \equiv 1024 \equiv 14 \pmod{101}$.