

# Crypto\_Final\_1-3

## 复习计划

- 古典密码、对称密码
- 一些数学基础、非对称密码
- 非加密部分
- 往年题目和补充

## 题目对应知识点

- 2020
  - 重放攻击的防范 3
  - 古典密码密钥空间计算 1
  - 数学基础知识 2
  - Fiestel 结构分析 1
  - RSA 2
  - GF( $p^n$ )上的计算 2
  - MAC 分析 3
  - RSA签名伪造 2
  - 密钥分发过程分析 3
- 2021
  - 古典密码密钥空间计算 1
  - 数学基础知识 2
  - RSA 2
  - GF( $p^n$ )上的计算 2
  - 数字签名分析 3
  - 哈希函数分析 3
  - RSA签名伪造 2
  - RSA 累加器分析 4

## 古典密码

### 凯撒密码

- 加密过程:  $c = p + k \bmod 26$
- 解密过程:  $p = c - k \bmod 26$
- 明文空间  $[0, 26)$

- 密文空间  $[0, 26)$
- 密钥空间  $[0, 26)$

1	$p + k \bmod 26$
2	$= (p \bmod 26 + k \bmod 26) \bmod 26$
3	$= (p + k \bmod 26) \bmod 26$
4	
5	如果 $k_1 = k_2 \pmod{26}$

每次加密的单位是一个字符：

- $p_1$ (长度32) 使用  $k$  加密，得到  $c_1$ (长度32)
- $p_2$ (长度1) 使用  $k$  加密，得到  $c_2$ (长度1)

攻击方式：

- 已知一组明文-密文对即可反推密钥
  - $c = p + k \bmod 26$
  - $c - p = k \bmod 26$
  - $k = c_1 - p_1 \bmod 26$

1	tatbtc
2	-----d

- 密钥空间太小，穷举
- 频率分析

1	tatbtc
2	ubucud

## 仿射密码

- 加密过程： $c = a * p + b \bmod 26$
- 解密过程： $p = (c - b) * \text{inverse}(a) \bmod 26$
- 解密正确性：
  - $c = a * p + b \bmod 26$
  - $c - b = a * p \bmod 26$
  - $(c - b) * \text{inverse}(a) = a * \text{inverse}(a) * p \bmod 26$
  - $(c - b) * \text{inverse}(a) = p \bmod 26$
- 明文空间  $[0, 26)$
- 密文空间  $[0, 26)$
- 密钥空间：

- $Ek = (a, b)$
- $Dk = (a', b) \# a' = \text{inverse}(a, 26)$
- 因此需要  $a$  对 26 有逆
- 共有  $12 * 26$  种密钥
- 密钥恢复攻击：如果需要恢复仿射密码的密钥，需要收集两组  $p-c$  对。

```

1   c1 = a*p1 + b mod 26
2   c2 = a*p2 + b mod 26
3
4   (c2 - c1) = a*(p2 - p1) mod 26
5   (c2 - c1)*(inverse(p2 - p1)) = a mod 26
6
7
8   -----
9   如果 p2-p1 对 26 没有逆元：
10
11  a = 1, b = 0
12      p1 = 1 c1 = 1
13      p2 = 3 c2 = 3
14
15  a = 3, b = 3
16      p1 = 1 c1 = 6
17      p2 = 3 c2 = 12
18
19  6 = a*1 + b mod 26
20  12 = a*3 + b mod 26
21  6 = a*2 mod 26
22
23  注意，此时有多解：
24      6 = 3*2 mod 26
25      6 = 16*2 mod 26
26  a1,b1 = 3,3
27  a2,b2 = 16,16

```

- 穷举：虽然相比凯撒密码，密钥空间扩大，但仍旧太小。
- 频率分析攻击

## 维吉尼亚密码

- 类似凯撒，加上密钥
  - plain "aaabcbcdefefef"
  - key "abc" -> "0 1 2"

- cipher "abcbdd"
- 如果key和明文等长，类似一次一密，但是没有实际使用价值，保护和明文等长的key和直接保护明文代价是一样的
- 实际应用中，key会被循环利用，因此攻击者可以对密文进行分组，每一个分组是使用同样key的凯撒密码，那么，分析针对凯撒密码的攻击
  - 已知一对明文-密文：有效，整个分组的key泄露，整个分组都可被恢复
  - 穷举攻击：难以实现，虽然每个分组密钥理论只有26种，但是分组处理之后的明文不再具有实际意义，即使遍历所有可能明文，也没法识别出来
  - 频率分析：有效，虽然单词本身被打乱，但是字符的统计规律不会改变，依然可以频率分析
- 密钥空间：和长度有关，如果长度是 L，key是26个字母组成的单词，那么，密钥有  $26^L$  种。

## 简单的置换-栅栏

- 纵向写，横向读。
- key为栅栏深度
- 加密

```

1 | 明文:abcdefg
2 | 2层栅栏:
3 | a c e g
4 | b d f
5 | 密文:acegbdf
6 | -----
7 | 明文: ls rm clear
8 | 栅栏:
9 | l mcer
10 | sr la
11 | 密文: l mcersr la

```

## 复杂的置换

- 类比栅栏，可以认为是纵向写，横向按照特定顺序读
- key的长度为栅栏深度，key的内容为读的顺序
- 加密

```

1 | 明文: abcdefg
2 | key为: [2,1]

```

3	加密:
4	2 a c e g
5	1 b d f
6	密文: bdfaceg
7	对于一个密钥长度为n的置换加密, $n!$ 。

## 希尔密码

- key为一个可逆矩阵
- $c = A * p$
- $(A^{-1})c = (A^{-1}) * A * p$
- $p = A^{-1} * c$
- plain: [1, 3, 8, 5, 6, 7, 2, 4, 9]

1	[[1,5,2],
2	[3,6,4],
3	[8,7,9]]

- K(3x3)
- C(3x3) = K(3x3) \* P(3x3)
- $K'(3x3)^*C(3x3)=K'^*K * P(3x3)$

1	单位阵:
2	1 0 0
3	0 1 0
4	0 0 1
5	K:
6	0 1 0
7	1 0 0
8	0 0 1
9	P:
10	1 5 2
11	3 6 4
12	8 7 9
13	(加密)C = K*P:
14	3 6 4
15	1 5 2
16	8 7 9
17	(解密)P = (K <sup>-1</sup> )*C

- 密钥空间:

- $GL(n, Z_{29}) = (29^{6-1})*(29^{6-2}...*(29^{6-29})$
- 上面这个乘法的每一项代表一列，每一列理论最多有 $29^6$ 种可能，第一列要去除全0的场景，第二列去除第一列的线性组合29，第三列去掉前两列的线性组合 $29^2$ 。。。以此类推
- 例如作业1中，明文被编码到  $[0, 29)$ ，因为希尔密码使用可逆矩阵作为密钥，密钥空间的大小就是可逆矩阵的数量。
- 这和矩阵的大小以及矩阵内原始的数量都有关系。

## 题目示例

### 2020-Final-2

- 维吉尼亚密码
  - GF(29)
  - 长度在 $m_1-m_2$ 之间
    - $m_1-m_2$
    - 对于任意 $m_i$ ，密钥空间都有 $29^{m_i-1}$ 种
    - 从 $m_1$ 到 $m_2$ 的所有可能长度的所有可能密钥的和，我们假设长度包含 $m_1$  不包含  $m_2$
    - $29^{m_1+...+29^{(m_2-1)}} - (m_2-m_1)$
  - 只有密钥全0的情况下，才是平凡的
  - 所以假设密钥长度为 $m_4$ ，每位密钥都有29种可能，密钥为 $29^{m_4}$ 种，非平凡的有 $29^{m_4-1}$ 种
- 置换密码
  - GF(29)
  - 长度为 $m_3$
  - 长度 $m_3$ 的序列，每一个排列都是一种密钥
  - 只有顺序排列的情况下，并且明文长度不超过密钥长度，才是平凡的，不考虑这种情况。
  - 所以密钥的数量和长度为 $m_3$ 的序列的全排列数量相等，有  $m_3!$  种，非平凡的有 $m_3!$ 种
- 两者混合
  - 相继应用两种算法，等于同时拥有两个密钥，其中 $k_1$ 有 $29^{m_4}$ 种, $k_2$ 有 $m_3!$ 种
  - 只有两个 $k$ 都是平凡的，这个算法才会有一个平凡的key
  - 所以混合密码，密钥有  $(29^{m_4})^{(m_3!)}$  种

### 2021-Final-1

可以看到，两次的题目基本一致，仅数值变化。

## 对称密码

### 一次一密

选择和明文二进制等长的密钥，然后  $C = P \ xor \ K$ 。

#### 无条件安全

如果攻击者对一次一密暴力破解，当他遍历所有密钥的同时也遍历了等长度的所有可能信息，攻击者无法从中识别出明文。

#### 密钥配送问题

如果存在安全的方式交换和明文等长的密钥，为什么不直接交换明文？

#### 密钥不可复用

```
1 | C1 = P1 xor Key
2 | C2 = P2 xor Key
3 | C1 xor C2 = P1 xor Key xor P2 xor Key
4 | C1 xor C2 = P1 xor P2
5 | P2 = P1 xor C1 xor C2 (已知一对明文密文)
```

#### 比特翻转攻击

```
1 | A借B 1000 元
2 | 1 对应明文 '00110001'
3 | 假设此段key为 '10000111'
4 | 对应密文: '10110110'
5 | 攻击者篡改其中一个bit，新密文为: '10111110'
6 | 解密前没有检查是否被篡改，直接解密，得到: '0011100
7 | 1'，对应字符 9
8 | 明文变成了：A借B 9000 元
9 | 所以，需要保证密文的完整性。
```

## 流密码

- 延续一次一密的思路，如果可以使用较短的密钥生成较长的随机比特流，就可以使用这样的比特流进行加密

- 因为密钥长度远小于比特流长度，所以可以使用安全的方式交换

## feistel结构

- 代替和置换 - substiution&permutation
- 混淆和扩散 - confusion&diffusion (对于DES 分别由 S-box 和 P-box 提供)
- $L(i+1), R(i+1) = R(i), L(i) \text{ xor } f(R(i), k_i)$
- 最后一轮结束后再次交换（也可以认为最后一轮不交换）

## DES

- 数据加密标准
- 用到了16轮feistel结构
- 密钥长度是64，其中包含8个校验位

## 针对DES的攻击

- 有效密钥长度 56
- 线性分析  $2^{43}$
- 3DES
  - EDE 兼容 DES
  - 中间相遇攻击 安全强度

```

1 加密:
2 已知P
3 t1 = DESEnc(P, K1)
4
5 t2 = DESDec(t1, K2)
6 C = DESEnc(t2, K3)
7
8 解密:
9 已知C
10 t2 = DESDec(C, K3)
11 t1 = DESEnc(t2, K2)
12 P = DESDec(t1, K1)
13
14 兼容DES:
15 已知P
16 t1 = DESEnc(P, K1)
17 P = DESDec(t1, K1)
18 C = DESEnc(P, K1)

```

- 1 DES
  - 遍历所有 $2^{56}$ 种可能的密钥， $E(P, [K_i]) == C$
- 3 DES
  - 遍历 $2^{(56*3)}$ 种， $E(D(E(P, [K_{1i}]), [K_{2i}]), [K_{3i}]) == C$
  - 中间相遇攻击：
    - 遍历所有 $2^{56}$ 种可能的密钥， $C_1 = E(P, [K_{1i}])$ ，得到 $2^{56}$ 种 $C_1$ ,存进hash tbl
    - 遍历 $2^{(56*2)}$ 种可能的密钥， $D(E(C, [K_{3i}]), [K_{2i}])$  in hash tbl,  $K_1, K_2, K_3$

1	集合 A 存在若干数字，要求找出 $a, b$ 两个数字， $a + b = C$ 。
2	遍历 A 的同时，把 $c - a$ 存入哈希表，再次遍历 A，如果发现 $b$ 在哈希表中，则找到一对满足条件的 $a, b$ 。

## AES

- 高级加密标准
- 支持 128, 192, 256 三种长度的密钥
- AES 没有使用feistel结构，有轮的概念，每轮做下面操作：
  - 行替换
  - 列混淆
  - 字节代换
  - 轮密钥加

## 加密模式

- 用于块密码
- $P = P_1 | P_2 | P_3 \dots$
- $C = C_1 | C_2 | C_3 \dots$

## ECB

- 简单直观
- $C[i] = E(P[i], K)$
- $P[i] = D(C[i], K)$
- 密文的一个比特错误，导致一个明文分组错误，但是不会影响其他明文分组。
- 加解密都可以并行。
- 对于相同的明文，会被加密成相同的密文。

## CBC

- 借用上一个分组的加密结果，获取不同的明文
- $C[i] = E((P[i] \text{ xor } C[i-1]), k)$
- $P[i] = D(C[i], k) \text{ xor } C[i-1]$
- IV (不能重复使用)
  - 充当  $C[0]$
  - 避免第一个分组每次都一样，如果 IV 重复使用，则这一点无法保证。
- 密文的一个比特错误，对应明文分组错误，下一个明文分组的一个比特错误，后续分组不受影响。
  - 假设  $C[2]$  一个比特错误
  - $P[2] = D(C[2], k) \text{ xor } C[1]$  整个分组错误。
  - $P[3] = D(C[3], k) \text{ xor } C[2]$  一个比特错误。
  - $P[4] = D(C[4], k) \text{ xor } C[3]$  后续分组不受影响。
- 计算：
  - 加密：  $C[i] = E((P[i] \text{ xor } C[i-1]), k)$ , 计算一个分组的密文，依赖前一个分组的密文。不能并行。
  - 解密：  $P[i] = D(C[i], k) \text{ xor } C[i-1]$ , 需要两个分组的密文，都是已知的，所以可以并行解密。
- 比特翻转攻击
  - 和流密码或一次一密类似，攻击者可以翻转密文的某个比特，使得下一个分组对应明文的一个比特翻转。
    - xxxxxxxxxxxxxxxxxxxxxxxx username = xxx, isadmin = 0
    - doihfiljfo;sdjvifjoooooo1 username = xxx, isadmin = 1
    - 因为会影响前一个明文分组，所以比较容易识别

## Padding Oracle Attack \*

- 填充提示攻击
- 如果不满一个分组，怎么办 - padding?
- 利用 padding 错还原明文。

## CFB

- $C[i] = P[i] \text{ xor } E(C[i-1], k)$
- $P[i] = C[i] \text{ xor } E(C[i-1], k)$
- 密文的一个比特错误，对应明文分组一个比特错误，下一个明文分组错误，后续分组不受影响
  - 假设  $C[2]$  一个比特错误

- $P[2] = C[2] \text{ xor } E(C[1], k)$  一个比特错误
- $P[3] = C[3] \text{ xor } E(C[2], k)$  整个分组错误
- $P[4] = C[4] \text{ xor } E(C[3], k)$  后续分组不受影响。
- 计算：
  - 加密:  $C[i] = P[i] \text{ xor } E(C[i-1], k)$ , 计算一个分组的密文, 依赖前一个分组的密文。不能并行。
  - 解密:  $P[i] = C[i] \text{ xor } E(C[i-1], k)$ , 需要两个分组的密文, 都是已知的, 所以可以并行解密。
- **OFB**
  - $C[i] = P[i] \text{ xor } E^i(IV, k)$  #  $E^i$  指连续加密*i*次
  - $P[i] = C[i] \text{ xor } E^i(IV, k)$
  - 密文的一个比特错误, 对应明文分组一个比特错误, 不影响其他分组
    - 假设 $C[2]$ 一个比特错误
    - $P[2] = C[2] \text{ xor } E^2(IV, k)$ 一个比特错误
    - $P[3] = C[3] \text{ xor } E^3(IV, k)$ 后续分组均无影响
  - 计算:
    - 加密:  $C[i] = P[i] \text{ xor } E^i(IV, k)$ , 虽然异或很快且可以并行, 但是需要生成密钥流, 问题在于密钥流每生成一个分组, 都需要前一个密钥分组, 所以这一步不可并行
    - 解密: 同加密
    - 密钥流产生和明文无关, 所以加密方可预先计算
  - 比特翻转攻击: 流密码的特性
  - $IV$ 不能重复
- **CTR**
  - $C[i] = P[i] \text{ xor } E(\text{Counter}[i], k)$
  - $P[i] = C[i] \text{ xor } E(\text{Counter}[i], k)$
  - 密文的一个比特错误, 对应明文分组一个比特错误, 不影响其他分组
    - 假设 $C[2]$ 一个比特错误
    - $P[2] = C[2] \text{ xor } E(\text{Counter}[2], k)$ 一个比特错误
    - $P[3] = C[3] \text{ xor } E(\text{Counter}[3], k)$ 后续分组均无影响
  - 计算:
    - 加密:  $C[i] = P[i] \text{ xor } E(\text{Counter}[i], k)$ , 可以并行
    - 解密:  $P[i] = C[i] \text{ xor } E(\text{Counter}[i], k)$ , 可以并行
    - 密钥流产生和明文无关, 所以加密方可预先计算
  - 比特翻转攻击: 流密码的特性
  - $IV$ 不能重复, 否则出现密钥流重复

## 题目示例

### 2020-Final-4

- a: 安全(保证机密性)和可用(有应用场景, 效率高)
- b

```
1  Ln = Ro
2  Rn = Lo xor f(k, Ro)
3
4  已知Ln,Rn
5
6  因Ln = Ro: Ro = Ln, f(k, Ro) = f(k, Ln)
7  因Rn = Lo xor f(k, Ro)
8      Lo = Rn xor f(k, Ro)
9      Lo = Rn xor f(k, Ln)
10
11  Lo = Rn xor f(k, Ln)
12  Ro = Ln
```

```
1  绘制Fiestel结构解密图:
2  R3, L3 = 01, 02
3  R2, L2 = L3, R3 xor f(L3, k2)
4  R1, L1 = L2, R2 xor f(L2, k1)
5  I1, I2 = L1, R1
```

## 数学基础

### 整除相关计算

$a|b$  读作a整除b, 这时  $b = k^*a$ , 那么有以下结论

- 如果 $a|1$ 那么  $a = 1$  或  $a = -1$
- 任意a, 满足 $a|0$
- 如果 $a|b, b|c$  那么  $a|c$
- 如果 $a|b, a|c$  那么  $a|(sb+tc)$

$$a \equiv b \pmod{c}$$

- $n|(a-b) \rightarrow a \equiv b \pmod{n}$ 
  - $b = k1*n + c$   $a = (k1+(a-b)/n)*n + c$

- $a = b \bmod n, b=c \bmod n \rightarrow a=c \bmod n$
- $a+b \bmod n = ((a \bmod n) + (b \bmod n)) \bmod n$

$a = b \bmod n$

$a*c = b*c \bmod n$

上面这个性质对加、减、乘均成立，但是：

1	$18 = 48 \bmod 10$
2	$3 \neq 8 \bmod 10$

对于除法：

1	$ac = bc \bmod m$
2	$a = b \bmod (m/\gcd(c,m))$

例题：计算  $3^{123456789} \bmod 80$

1	$3^4 = 81$
2	$3^4 = 1 \pmod{80}$
3	$3^4 * 3^{123456785} \pmod{80}$
4	$1 * 3^{123456785} \pmod{80}$
5	$3^{(123456789 \% 4)} \pmod{80}$
6	---
7	关注点转移到: $123456789 \% 4$
8	$1234567 * 100 + 89 \% 4$
9	$20 * 4 + 9 \% 4$
10	$2 * 4 + 1 \% 4$
11	---
12	$3 ^ 1 \% 80 = 3$

## 算数基本定理

- 任意正整数可以表示成若干个 1 的加法，因此 1 是加法的基本单位。
- 素数则是乘法的基本单位，任意正整数可以分解成若干素数的乘积。

1	$m = p_1^{a_1} * p_2^{a_2} \dots p_n^{a_n}$
---	---------------------------------------------

任意正整数都可以得到这样的分解，并且已知正整数的分解后，恢复这个整数是容易的。

但是如果只知道整数，要得到它的分解，这是一个数学上的困难问题。

```

1 | def fac(n):
2 |     result = []
3 |     while n != 1:
4 |         for i in range(2, n + 1):
5 |             if n % i == 0:
6 |                 result.append(i)
7 |                 n //= i
8 |                 break
9 |     return result

```

## 欧拉定理

如果  $\gcd(a, n) == 1$ :

$$a^{\varphi(n)} \equiv 1 \pmod{n}$$

其中  $\varphi$  称为欧拉函数。

## 欧拉函数

- 定义欧拉函数  $\phi(n)$  为小于  $n$  且和  $n$  互素的整数数量
- 7: 1 2 3 4 5 6  $\phi(7) = 6$
- 6: 1 5  $\phi(6) = 2$
- 如果  $p$  是素数,  $\phi(p) = p-1$
- 如果  $n = p_1 * p_2$ ,  $\phi(n) = (p_1-1)(p_2-1)$
- 如果  $n = p_1 * p_2 \dots$ ,  $\phi(n) = (p_1-1) \dots$
- 如果存在相同的  $p$ ,  $n = (p_1^{a_1}) * p_1, \phi(n) = (p_1^{a_1}) * (p_1-1)$ 
  - 把不同的因子写到最前面, 如果出现相同的, 补到后面
  - $p_1^{a_1} * p_2^{a_2} * p_3^{a_3} \dots = (p_1-1) (p_2-1) (p_3-1) \dots p_1^{a_1} p_2^{a_2} p_3^{a_3} \dots$

也可以直接使用下面的计算方式:

$$\varphi(m) \equiv m \prod_{i=1}^n \left(1 - \frac{1}{p_i}\right)$$

其中  $p$  为  $m$  的分解:  $m = p_1^{a_1} p_2^{a_2} \dots p_n^{a_n}$

例题:  $n = 72283$  求  $\phi(n)$  ?

```

1 | 显然: 72283 = 41*41*43
2 | phi(72283) = (41-1)*41*(43-1) = 68880

```

## 证明

如果  $a$  和  $p_1 \dots p_n$  均和  $m$  互素，那么  $ap_1 \dots ap_n$  均和  $m$  互素。

如果  $p_1 \dots p_n$  恰好覆盖了  $0-m$  之间所有和  $m$  互素的数，则  $ap_1 \dots ap_n$  对  $m$  取余后同样完成覆盖。

反证法，如果  $api_1 = api_2 \pmod{m}$ ，那么  $pi_1 = pi_2 \pmod{m}$ ，这与前提矛盾。

1	$api_1 = k_1m+b$
2	$api_2 = k_2m+b$
3	$a(pi_1 - pi_2) = (k_2 - k_1)m$

那么

$$\prod_{i=1}^n pi \equiv \prod_{i=1}^n a p_i \pmod{m}$$
$$\prod_{i=1}^n p_i \equiv a^{\varphi(m)} \prod_{i=1}^n p_i \pmod{m}$$
$$a^{\varphi(m)} \equiv 1 \pmod{m}$$

## 费马小定理

- 费马小定理为欧拉定理特殊形式
  - 若  $m$  为素数，则  $\phi(m) = m-1$
  - $p = m$
  - $\text{pow}(a, p-1, p) = 1$

## 欧几里得算法

### 欧几里得算法

- 对于  $A, B$ ，要求  $\text{GCD}(A, B)$
- 考虑  $A = k*B + R$
- $\text{GCD}(A, B) = \text{GCD}(k*B + R, B) = \text{GCD}(B, R)$

### 扩展欧几里得算法

使用欧几里得算法可以求出  $\text{GCD}(A, B)$ ，扩展欧几里得算法给出一个构造性的证明，找到  $s$  和  $t$  满足  $sA + tB = \text{gcd}(A, B)$ 。

1	$\text{gcd}(a, b) == \text{gcd}(b, a \% b)$
2	
3	假设 $a, b$ 最大公因数为 $T$ , $\text{gcd}$ 递归下降的终点为 $(1*T + y*0 ==$

T)  $1 * T + y * 0 == T$ , 在gcd算法向上回溯的过程中总  
 有:  
 # a 和 b 是每一次调用 gcd 时的入参  
 # x 和 y 为对应系数  
 T = nx\*a + ny\*b  
 T = x\*b + y\*(a%b)  
 这里 a%b 写成 a-k\*b, 其中 k = a//b, 则  
 T = b\*x + a\*y - k\*b\*y  
 T = [y]\*a + [(x-k\*y)]\*b  
 那么 nx, ny = y, (x-a//b\*y), 由此递推计算 x, y

例题: a = 96 b = 35 , 应用 XGCD

a	b	x	y	等式
96	35			
35	26			
26	9			
9	8	1	-1	$9*1+8*(-1) == 1$
8	1	0	1	$8*0+1*1 == 1$
1	0	1	0	$1*1+0*0 == 1$

a	b	x	y
96	35	-4	11
35	26	3	-4
26	9	-1	3
9	8	1	-1
8	1	0	1
1	0	1	0

## 求逆元

我们的目标是对于 A 和 N 找到 A 的逆元 A', 使得  $A * A' = 1 \pmod{N}$ 。  
 扩展欧几里得算法对于给定 (a,b) 扩展欧几里得算法可以计算出  $a*x + b*y = gcd(a, b)$ , 那么, 只要  $gcd(A, N) == 1$ , 令 (a,b) 为 (A,N), 可以计算  $A*x + N*y = 1$ , 则  $A*x = 1 \pmod{N}$ , x 即为一个满足条件的 A'。

一种利用费马小定理求逆元的方法：

如果N是素数，那么由于费马小定理， $A^{(N-1)} \equiv 1 \pmod{N}$

$A * A^{(N-2)} \pmod{N}$

$A' = A^{(N-2)} \pmod{N}$

## 代码实现

```
1 def XGCD(a, b):
2     if (b == 0):
3         return a, 1, 0
4     gcd, x, y = XGCD(b, a%b)
5     return gcd, y, x-a//b*y
6
7
8 def inverse(a, n):
9     gcd, x, y = XGCD(a, n)
10    return x%n if gcd == 1 else None
```

## 快速幂

朴素实现：

```
1 def my_pow(a,b,c):
2     t = 1
3     for i in range(b):
4         t *= a
5         t %= c
6     return t % c
```

问题：如果我们要计算 1024 个 b 相乘，使用上面的函数就需要计算 1023 次乘法。

考虑到：

$$b^{1024} = b^{512} * b^{512}$$

$$b^{512} = b^{256} * b^{256}$$

256

128

64

32

16

8

4

2

这样只需算10次乘法。

```
1 1024 : 0b100000000000
2 65537 : 0b100000000000000000000001
```

```
1 # for bit in b:
2 #     if bit == 1:
3 #         t *= m
4 #     m *= m
5 #a^b
6 #b个a相乘
7 #....k2*2^2+k1*2^1+k0*2^0
8 #(a^(ki*2^i))*(a^(kj*2^j))...
9
10 def my_fast_pow(a,b,c):
11     t = 1
12     for bit in bin(b)[2:][::-1]:
13         if bit == '1':
14             t *= a
15             t %= c
16         a *= a
17         a %= c
18     return t % c
```

## 题目示例

### 2022-Practive-2

因为  $n$  是素数，所以  $\text{pow}(a, n-1, n) \equiv 1 \pmod{n}$

所以  $a * \text{pow}(a, n-2, n) \equiv 1 \pmod{n}$

所以  $x = \text{pow}(a, n-2, n)$

```
1 类似python代码:
2 def inverse(a, n):
3     assert n is prime
4     return pow(a, n-2, n)
5 ---
6 类似伪代码:
7     ret pow(a, n-2, n)
```

- None
- 8905
- 12
- -7 (4-6 补充看看能否根据上面内容推导出)

F3补充:

```

1 | 4 利用了  $29 \% 17 = 12$ :
2 |  $12 * 17 + (-7) * 29 = 1$ 
3 |
4 |  $12 * (-7) = 1 \bmod 17$ 
5 |
6 |
7 | 5 和 6 利用了上面的结论:
8 |  $\text{pow}(12, 30, 31) = 1 \rightarrow 12 * \text{pow}(12, 29, 31) =$ 
9 |     1  $\rightarrow$  逆为 13
    |  $\text{pow}(10, 30, 31) = 1 \rightarrow 10 * \text{pow}(10, 29, 31) =$ 
    |     1  $\rightarrow$  逆为 28

```

## 2020-Final-3

```

1 | a)
2 | 计算:  $a^{(p-1)} + (p-1)^a \bmod p$ 
3 |  $a$ 是和 $p$ 互素的奇数。
4 |  $a^{(p-1)} + (p-1)^a \bmod p$ 
5 |  $= (a^{(p-1)} \bmod p + (p-1)^a \bmod p) \bmod p$ 
6 |  $= 1 + (-1) \bmod p$ 
7 |  $= 0 \bmod p$ 
8 |  $= 0$ 
9 |
10|
11| b)
12| 此类题目一般将幂次降低之后使用带入法求解。
13| 对于  $f(x) = a_n * x^n + \dots = 0 \bmod p$  的同余式, 可以使
   | 用多项式因式分解的方式降幂, 保证最大次数不高于 $p$ , 方法如
   | 下:
14| 根据费马小定理:
15|  $x^p = x \bmod p$ 
16|  $x^p - x = 0 \bmod p$ 
17|
18| 令  $g(x) = x^p - x$ 
19| 则  $f(x) = g(x) * p(x) + t(x) = 0 \bmod p$ 

```

20 最终, 问题转化成  $t(x) = 0 \pmod{p}$

21 对照这个问题:

22  $f(x) = 3x^{14} + 4x^{10} + 6x - 18$

23  $g(x) = x^5 - x$

24 则  $t(x) = 7x^2 + 6x - 18$

25

26  $3x^9 + 7x^5 + 7x$

27

28  $\overline{x^5 - x / 3x^{14} + 4x^{10} + 6x - 18}$

29  $3x^{14} - 3x^{10}$

30

31  $7x^{10} + 6x - 18$

32  $7x^{10} - 7x^6$

33

34  $7x^6 + 6x - 18$

35  $7x^6 - 7x^2$

36

37  $7x^2 + 6x - 18$

38

39 手工带入得到:

40  $x - t(x)\%5$

41  $0 - 2$

42  $1 - 0$

43  $2 - 2$

44  $3 - 3$

45  $4 - 3$

46

47  $x = 1 \pmod{5}$

48

49

50 一次同余式, 有通用解法, 求逆再乘系数。

51 二次比较复杂。

## 2021-Final-2

可以看到两年题目差异非常小。

## 非对称

### 1. 密钥配送问题

- 2. 共享密钥数量爆炸
- 3. 需要签名机制

## RSA 参数计算

- 利用了大数分解的困难性
1. 生成  $p, q$
  2. 计算  $n = p \cdot q$
  3. 计算  $n$  的欧拉函数,  $\phi(n) = (p-1) \cdot (q-1)$
  4. 生成公钥  $e$ , 和  $\phi(n)$  互素即可
  5. 计算私钥  $d$ , 满足  $e \cdot d \% \phi(n) = 1$ , 例如  $d = \text{inverse}(e, \phi(n))$
  6. 丢弃  $\phi(n), p, q$
  7. 保留  $e, n$  作为公钥,  $d$  作为私钥

## RSA 应用

### 加解密

- 加密:  $C = \text{pow}(P, e, n)$
  - 解密:  $P = \text{pow}(C, d, n)$
  - $P_2 = \text{pow}(\text{pow}(P, e, n), d, n)$
  - $P_2 = \text{pow}(P, e \cdot d, n)$
  - $P_2 = \text{pow}(P, k \cdot \phi(n) + 1, n)$
  - $P_2 = \text{pow}(\text{pow}(P, \phi(n), n), k, n) \cdot \text{pow}(P, 1, n)$
  - $P_2 = 1 \cdot P$
- $(P \wedge e \% n) \wedge d \% n$
- $(P \wedge e) \wedge d \% n$

### 数字签名

- 签名:  $S = \text{pow}(M, d, n)$
- 验证:  $M == \text{pow}(S, e, n)$

## RSA 攻击

### 直接分解

- 是一个困难问题
- 目前最高768bit
- <http://www.factordb.com/>

### 参数选取不当导致的分解

- $p, q$  过近 -- 开方寻找
  - 利用  $p, q$  两个数字都在根号  $n$  附近
- 重用因子 -- gcd 分解
  - $n_1 = p_1 * p_2$
  - $n_2 = p_1 * p_3$
  - $\gcd(n_1, n_2) = p_1$
  - $p_1 | n_1, p_2 = n_1 // p_1$

## 明文枚举攻击

- 明文空间较小，枚举明文
  - $e, d, n$  安全，但是  $m$  空间较小
  - 攻击者具有加密所需所有内容，除了明文
  - 攻击者枚举所有可能的明文，尝试加密，如果与已知密文相同，则可以确定明文

例：明文是 "ABC"  $\rightarrow 0 + 1 * 100 + 2 * 10000 = 20100$

相对不安全的编码方式："ABC"  $\rightarrow [0, 1, 2]$

## 共模攻击

一种对 RSA 的错误使用方式。

```

1 1. 生成  $p, q$ 
2 需要生成大素数  $p, q$ 。这个过程是很耗时间的。
3 2. 计算  $n = p * q$ 
4 3. 计算  $n$  的欧拉函数， $\phi(n) = (p-1)*(q-1)$ 
5  $\phi(n)$ 
6 4. 生成公钥  $e$ ，和  $\phi(n)$  互素即可
7 这个  $e$  会用于加密，计算时作为指数，因此  $e$  的二进制长度
8 和其中比特 1 的个数会影响加密时间。
9 5. 计算私钥  $d$ ，满足  $e * d \% \phi(n) = 1$ ，例如  $d = \text{inverse}(e, \phi(n))$ 
10  $d$  用于解密，计算时作为指数，因此  $d$  的二进制长度和其中
11 比特 1 的个数会影响解密时间。
    6. 丢弃  $\phi(n), p, q$ 
    7. 保留  $e, n$  作为公钥， $d$  作为私钥

```

$m$  相同， $N$  相同，使用互素的  $e$  发给不同用户，则不需要分解  $N$  即可还原。

- $e_1, e_2$  互素
- 用到 广义欧几里得算法

XGCD: 如果  $e_1$  和  $e_2$  互素, 那么我们可以找到  $s_1$  和  $s_2$ , 令  $e_1*s_1 + e_2*s_2 == 1$

```
1 c1 = pow(m, e1, n)
2 c2 = pow(m, e2, n)
3
4 pow(c1, s1, n) == pow(m, e1*s1, n)
5 pow(c2, s2, n) == pow(m, e2*s2, n)
6
7 pow(c1, s1, n)*pow(c2, s2, n) = pow(m, e1*s1 +
e2*s2, n) = m
```

如果  $e_1, e_2$  不互素, 那么不能应用共模攻击。

但是, 假定  $e_1 = a*b$ ,  $e_2 = a*c$

可以对  $b$  和  $c$  应用 XGCD, 计算出  $\text{pow}(m, a, n)$ 。

## 利用计算的性质, 侧信道攻击

```
1 pow(m, d, n)
2 t = 1
3 for bit in d:
4     if bit == 1:
5         t *= m
6     m *= m
```

- 从时间上: 如果计算时间较长, 是不是就说明  $d$  中有更多的1?
- 侧信道能量分析: 如果比特为1, CPU 能量信息会比较高。

## 签名构造

RSA 运算具有乘法性质:

```
1 pow(m1, d, n) * pow(m2, d, n)
2 = pow(m1*m2, d, n)
3
4 如果 m3 = m1 * m2
5 S1 = pow(m1, d, n)
6 S2 = pow(m2, d, n)
7 我们就可以计算 S1 * S2
8 = pow(m1, d, n) * pow(m2, d, n)
9 = pow(m1 * m2, d, n)
```

```
10 = pow(m3, d, n)
11 = S3
```

## 盲签名

攻击者误导受害者签名看上去随机的内容，得到针对指定内容的签名。

```
1 令 M' = FastModExp(x, e, n) * M % n
2 计算 T = Sign(M', d, n)
3   = Sign(FastModExp(x, e, n) * M, d, n)
4   = FastModExp(FastModExp(x, e, n)*M, d, n)
5   = FastModExp(FastModExp(x, e, n), d, n) * FastM
6   odExp(M, d, n) % n
7   = x*Sign(M, d, n) % n
8   = x*S % n
9 计算 S' = T * inverse(x, n) % n == 1*S % n = S
```

## DH 密钥交换

```
1 没有减法的世界：
2 1. 人人都会算加法，但是没人会减法
3 2. t + a + b = t + b + a
4
5 Alice Bob 通信
6 A 公开一个公共参数： 10
7 --- 攻击者 (10)
8 A 选择自己的私钥： 3 并计算公钥： 10+3 = 13
9 B 选择自己的私钥： 5 并计算公钥： 10+5 = 15
10 A B 交换公钥
11 --- 攻击者 (10, 13, 15)
12 A 计算共享密钥 15 + 3 = 18
13 B 计算共享密钥 13 + 5 = 18
14 共享密钥为 18
```

```
1 实际选取的运算：
2 A = g ^ a % p
3 正向计算：已知a，计算A是容易的；
4 反向计算：已知A，计算a是数学上的困难问题；
5 交换律：
6 (g ^ a % p) ^ b % p
```

```

7   = (g ^ a) ^ b % p
8   = (g ^ a*b) % p
9   = (g ^ b*a) % p
10  = (g ^ b % p) ^ a % p
11
12  公开信息: g, p
13  Alice
14    私钥: a
15    公钥: A = pow(g, a, p)
16  Bob
17    私钥: b
18    公钥: B = pow(g, b, p)
19
20  共享密钥: K = pow(B, a, p) = pow(g, a*b, p) = pow(A, b, p)
21
22  g p A B
23
24  K

```

## 针对 DH 密钥交换的中间人攻击

```

1  公开信息: g, p
2  Alice
3    私钥: a
4    公钥: A = pow(g, a, p)
5  Bob
6    私钥: b
7    公钥: B = pow(g, b, p)
8  A --> B : A, g, p (a, A, g, p)
9  B --> A : B (b, B, g, p, A) => [K = pow(A,
b, p)]
10 A : (a, A, g, p, B) => [K = pow(B, a, p)]
11
12 =====
13 ===
14  user C
15    私钥: c
16  A --X B --> C: A, g, p
17    A: (a, A, g, p)
18    B: (b)

```

```

18      C: (A, g, p, c) => [C = pow(g, c, p), KA =
19          pow(A, c, p)]
20          C --> B: C, g, p
21          A: (a, A, g, p)
22          B: (b, g, p, C) => [B = pow(g, b, p), KB =
23          pow(C, b, p)]
24          C: (A, g, p, c, C)
25          B --X A --> C: B
26          A: (a, A, g, p)
27          B: (b, g, p, C, B)
28          C: (A, g, p, c, C, B) => [KB = pow(B, c,
29          p)]
30          C --> A : C
31          A: (a, A, g, p, C) => [KA = pow(C, a, p)]
32
33          A-C : KA
34          B-C : KB
35          =====
36          m = (A,g,p)
37          h = H(k||m)
38          m1 = (C,g,p)
39          h1 = H(k||m1) k是攻击者不知道的
40          -----
41          m1 || h
42          -----
43          m1
44          h1 = H(k||m1)
45          h1 != h

```

## ElGamal

- 基于DH原理的公钥密码
- 下面使用DH密钥交换的变量描述ElGamal

```

1 和 DH 密钥交换一样, Alice 可以公开信息: p, g, A 作
2 为公钥, a 为私钥。 A = pow(g, a, p)
3
4 当 Bob 准备给 Alice 发消息时, 进行下列操作:
5     选择随机数 b(DH中的私钥), 计算公钥 B = pow(g, b
6     , p)
7     计算共享密钥 K_s = pow(A, b, p)
8     ---

```

```

7   计算密文  $C = K_s * M \bmod p$ 
8   这是一个类似仿射密码的过程，并且这才是实际的加密过
9   程。
10  -----
11      发送 B 和 C 给 Alice
12
13  Alice 收到消息后：
14      使用  $a$  和  $B$  恢复  $K_s = \text{pow}(B, a, p)$ 
15      解密  $M = \text{inverse}(K_s, p) * C \bmod p$ 
16
17  正确性：
18       $K_s = \text{pow}(B, a, p) = \text{pow}(g, a*b, p) = \text{pow}($ 
19       $A, b, p)$ 
20      -  $C = K_s * M \bmod p$ 
21      -  $M = \text{inverse}(K_s, p) * C \bmod p = \text{inverse}($ 
22       $K_s, p) * K_s * M \bmod p$ 
23
24      =  $1 * M \bmod p$ 
25
26      = M

```

## 题目示例

### 2020-Final-5

```

1 sage: p = 29
2 sage: q = 41
3 sage: fn = (p-1)*(q-1)
4 sage: fn
5 1120
6 sage: d_lst = []
7 sage: for d in range(3, fn):
8 .....    if GCD(d, fn) == 1:
9 .....        d_lst.append(d)
10 .....    if len(d_lst) == 3:
11 .....        break
12 .....
13 sage: d_lst
14 [3, 9, 11]

```

### 2020-Final-8

```

1 已知签名对(m1, s1), (m2, s2), (m3, s3), 能否伪造m
2 4的签名s4, 其中m4 = m1*m2^2*m3^3?
3 s4 = Sign(m4, d, n)
4 = pow(m4, d, n)
5 = pow(m1*m2*m2*m3*m3*m3, d, n)
6 # 考虑到: pow(m1*m2, d, n)
7 # = (m1*m2)^d % n
8 # = (m1)^d * (m2)^d % n
9 # = ((m1)^d % n) * ((m2)^d % n)) % n
10 # = pow(m1, d, n) * pow(m2, d, n) % n
11 # = s1 * s2 % n
12 = pow(m1, d, n)*pow(m2, d, n)*pow(m2, d, n)*pow(m3, d
13 ,n)*pow(m3, d, n)*pow(m3, d, n) % n
14 = Sign(m1, d, n) .... % n
15 = s1*s2^2*s3^3 % n
16
17
18
19 如果RSA签名使用:
20 def sign(M, n, d):
21     return pow(M, d, n)
22
23 对于乘法具有线性的性质。
24 因为:
25     pow(M1, d, n)*pow(M2, d, n) % n
26     = ((M1^d % n) * (M2^d % n)) % n
27     = ((M1*M2)^d) % n
28     = pow(M1*M2, d, n)
29 但是对于加法:
30     pow(M1+M2, d, n)
31     = (M1+M2)^d % n
32     = M1^d+M2^d+d*M1*M2^(d-1)...
33 所以 2 可以被伪造。
34 补充系数不为1, 不能伪造:
35 假设存在 m1 对应签名为 s1:
36 m2 = 2*m1 能否伪造 s2?
37 pow(m2, d, n)
38 = pow(2*m1, d, n)
39 = pow(2, d, n)*pow(m1, d, n)
40 = pow(2, d, n)*s1
41 对于攻击者 pow(2, d, n) 是未知的, 主要原因是d不知道, 所
42 以没法伪造。

```

## 2021-Final-7

## 2022-Practive-6

F3补充: 这个题目和前两年没有区别。

### 盲签名

补充21年A2的对应题目看看。

F3补充:

#### a 实现 RSA 签名

```
1 # 1a
2 def sign(M, n, d):
3     return pow(M, d, n)
```

#### b 实现 RSA 签名验证

```
1 # 1b
2 def verify(S, n, e, M):
3     return sign(S, n, e) == M
```

#### c 实现 RSA 盲签名

```
1 M_b = pow(x, e, n) * M % n
2 S = sign(M, d, n)
3 S_b = sign(M_b, d, n)
4     = sign(pow(x, e, n) * M, d, n)
5     = pow(pow(x, e, n) * M, d, n)
6     = pow(pow(x, e, n), d, n) * pow(M, d, n) % n
7     = x * sign(M, d, n) % n
8     = x * S % n
9 S_b * inverse(x, n) % n == S
```

## GF(P<sup>N</sup>) 计算

### 计算方式

- 1 Consider  $\text{GF}(2^3) = \text{GF}(2)[x] \bmod (x^3 + x^2 + 1)$ , a field with 8 elements.  
 2 Express all the elements of  $\text{GF}(2^3) = \text{GF}(2)[x] \bmod (x^3 + x^2 + 1)$  as polynomials.

和多项式运算类似，需要注意两点：

- 系数对 p 取模，这里是 2
- 如果多项式最高次数大于等于 n，对不可约多项式取模

i	$x^i$	as polynomials
0	1	1
1	x	x
2	$x^2$	$x^2$
3	$x^3$	$x^2 + 1$
4	$x^4$	$x^2 + x + 1$
5	$x^5$	
6	$x^6$	
7	$x^7$	

$$\begin{array}{r}
 1 \\
 2 \\
 3 \quad x^3 + x^2 + 1 / x^3 \\
 4 \qquad x^3 + x^2 + 1 \\
 5 \\
 6 \qquad x^2 + 1 \\
 7 \\
 8 \qquad x + 1 \\
 9 \\
 10 \quad x^3 + x^2 + 1 / x^4 \\
 11 \qquad x^4 + x^3 + x \\
 12 \\
 13 \qquad x^3 + x \\
 14 \qquad x^3 + x^2 + 1 \\
 15 \\
 16 \qquad x^2 + x + 1 \\
 17
 \end{array}$$

可以用sage计算：

```
1 | F.<x> = GF(2^3, modulus=[1,0,1,1])
```

注意，系数的运算是对p模运算，这在p=2时恰好加减法就是异或运算，但是p为其他值时并不是这样，需要先正常使用加减法再对结果做模运算。

```
1 sage: F.<x> = GF(3^3) # 补充查不可约多项式方法
2 sage: for i in range(8):
3 ....:     print(x^i)
4 ....:
5 1
6 x
7 x^2
8 x + 2
9 x^2 + 2*x
10 2*x^2 + x + 2
11 x^2 + x + 1
12 x^2 + 2*x + 2
```

```
1 求逆:
2
3 GF(2^4), x^4+x+1
4
5 x^3+x^2
6
7 和整数一样:
8
9 x^4+x+1 / x^3+x^2 = x + 1 ... x^2 + x + 1
10
11
12
13 x^3 + x^2   /x^4 +      x + 1
14
15
16
17
18
19
20
21 x^3 + x^2 / x^2 + x + 1 = x ... x
```

```

22
23
24
25      x
26      -----
27      x^2 + x + 1   /x^3 + x^2
28      x^3 + x^2 + x
29
30      -----
31      x
32
33      -----
34      x   /x^2 + x + 1
35      x^2
36
37      -----
38      x + 1
39
40      -----
41      x
42      -----
43      1
44
45      1*x + 1*(x+1) = 1
46
47  那么 nx, ny = y, (x-a//b*y), 由此递推计算x, y

```

a	b	x	y
$x^4+x+1$	$x^3+x^2$	$x^2+x+1$	$(x+1)-(x+1)(x^2+x+1) = (x+1)-$ $(x^3+x^2+x+x^2+x+1) = x^3$ $+x$
$x^3+x^2$	$x^2+x+1$	$x+1$	$x^2+x+1$
$x^2+x+1$	x	1	$x+1$
x	$x + 1$	1	1

$$1*(x^2+x+1) + (1-0)*1 = 1$$

如果除数次数较高怎么处理?

求逆：

$GF(2^4), x^4+x+1$

$x^5$

在  $GF(2^4)$

$$x^5 = x^2+x$$

1		x
2		-----
3	$x^4+x+1$	/ $x^5$
4		$x^5+x^2+x$
5		-----
6		$x^2+x$

所以只要求  $x^2+x$  对  $x^4+x+1$  的逆就可以了。

---

这部分的运算在 AES 算法的列混淆部分有应用。

用的比较多的是  $GF(p)$  和  $GF(2^n)$  这两种场景。

## 题目示例

### 2022-Practive-4

### 2020-Final-6

```
1 可以利用sage:  
2 F.<x> = GF(2^3, modulus=[1,0,1,1])  
3 # ai [a_i*x^i] i 属于[0, n]  
4  
5 for i in range(8):  
6     print(i, x^i)  
7  
8 (0, 1)  
9 (1, x)  
10 (2, x^2)  
11 (3, x^2 + 1)  
12 (4, x^2 + x + 1)  
13 (5, x + 1)  
14 (6, x^2 + x)  
15 (7, 1)  
16
```

```

17 手工, 和之前的多项式计算没有区别, 注意加减实际为异或:
18 1 - 1
19 x - x
20 x^2 - x^2
21 x^3 - x^2+1
22
23 1
24 -----
25 x^3+x^2+1/x^3
26 x^3 + x^2 +1
27 -----
28 x^2 + 1
29
30 x^6 - x^2 + x
31
32 x^3+x^2+x
33 -----
34 x^3+x^2+1/x^6
35 x^6 + x^5 + x^3
36 -----
37 x^5 + x^3
38 x^5 + x^4 + x^2
39 -----
40 x^4 + x^3 + x^2
41 x^4 + x^3 + x
42 -----
43 x^2 + x

```

```

1 sage: (x^2+x)^(-1)
2 x
3
4 手工:
5 inverse(x^2+x, x^3+x^2+1)
6
7 xgcd(x^2+x, x^3+x^2+1)

```

和之前是一样的(mid):

<b>a</b>	<b>b</b>	<b>x</b>	<b>y</b>
a	b	nx	ny

<b>a</b>	<b>b</b>	<b>x</b>	<b>y</b>
<b>a</b>	$a \% b$	x	y

- $1 = b*x + (a - k*b)*y$
- $1 = b*x + a*y - k*b*y$
- $1 = a*[y] + b*[(x - k*y)]$
- $nx = y$
- $ny = x - y*(a/b)$
- ps.  $k = a/b$

<b>a</b>	<b>b</b>	<b>x</b>	<b>y</b>
$x^{3+x}2+1$	$x^2+x$	1	x
$x^2+x$	1	0	1
1	0	1	0

1	$x$
2	-----
3	$x^2+x/x^3 + x^2 + 1$
4	$x^3 + x^2$
5	-----
6	1
7	

## 2021-Final-4

# 完整性保护

- 对于被动攻击，我们通常关注机密性，也就是防止消息被窃听，被第三方知道
- 除了机密性外，也有一些场景，我们可能不关心消息是否会被其他人知道，而是比较关注消息本身是否被篡改
  - 下载文件被劫持，文件内容被篡改成病毒
  - 重要文件被修改内容，导致利益纠纷

# HASH

要求对任意大小的数据，生成一个定长的值，这个值相当于数据的指纹，如果数据发生改变，HASH值必定改变（我们认为的）。

- 输入可变长
- 输出定长
- 效率高
- 抗原像攻击  
// 给定  $\text{Hash}(A)$  很难找到  $A$   
给定一个哈希值  $h$ , 很难找到原文  $m$ , 使得  $\text{Hash}(m) = h$
- 抗第二原像攻击  
给定  $A$  很难找到  $B$ , 满足  $\text{Hash}(A) == \text{Hash}(B)$
- 抗碰撞  
给定  $\text{Hash}$ , 很难找到  $A$  和  $B$ , 满足  $\text{Hash}(A) == \text{Hash}(B)$

常用哈希函数: SHA-3 SHA-256 SHA-1 MD5 . . .

哈希应用流程:

- $M||H(M)$ : 简单, 但是主动攻击者修改消息后可以替换哈希, 导致校验无效
- $E(K, M||H(M))$ : 因为被加密, 攻击者无法篡改并替换哈希, 但是需要加密, 在无需保密性的场景代价较高
- $M||E(K, H(M))$ : 相对方案1, 对哈希加密保护, 攻击者无法篡改, 并且消息本身不加密, 代价较低
- $M||H(M||K1)$ : 带KEY的哈希, 一种简单的MAC, 可以认证对端身份
- $E(K, M||H(M||K1))$ : 增加保密性支持

## 题目示例

### 2022-A2-4

消息  $A$  -- 有意义的情报

计算结果  $\text{Hash}(A)$

可以根据结果, 构造一个输入, 使得  $\text{HASH}(\text{输入}) == \text{结果}$ 。

因此对于这样的哈希函数, 如果有一个哈希值  $h$ 。

$$M_0^3 + M_1^3 + M_2^3 + \dots + M_n^3 \% n$$

这样的话, 在已知  $h$  的情况下, 可以构造一个原始消息  $m$ , 此消息具有  $h$  个分组, 每一个分组的内容都对应数字 1。

$\text{Hash}(m)$

$$= 1^{3+1} 3 \dots + 1^3 (\text{一共 } h \text{ 个}) \% n$$

$$= h$$

因此对于  $h$  找到了  $m$  使得  $\text{Hash}(m) = h$ 。

## 认证

- HASH可以防止消息篡改，但是无法提供认证
  - 因为接收者无法确认对方的身份，攻击者C可以声称自己是A，发送消息给B，B无法确认对方身份，即使消息中带有HASH
  - 因为HASH算法是公开的，并且输入中没有可以证明自己身份的东西
- 

## 加密中的认证

- 对称加密除了提供保密性外，其实还起到了认证对方身份的作用，因为对方只有知道共享密钥才能发送加密消息给你
- 但是非对称加密因为使用公钥加密，公钥是公开的，所以任何人都可以做这个操作
- 为了实现认证，需要使用私钥进行签名，因为私钥只有一个人持有，所以可以认证对方身份
- 同时实现完整性和保密性保护的方法： $E(PU_b, E(PR_a, M))$

```
1 | 发送者:  
2 |   E(PR_a, M)  
3 |   E(PU_b, E(PR_a, M))  
4 |  
5 | 接收者:  
6 |   E(PR_b, E(PU_b, E(PR_a, M))) = E(PR_a, M)  
7 |   E(PU_a, E(PR_a, M)) = M
```

## MAC

- MAC提供了较低代价的认证，不需要对整个消息进行加解密操作
- 利用HASH实现的MAC： $\text{HASH}(\text{KEY xor OPAD} \parallel \text{HASH}(\text{KEY xor IPAD} \parallel M))$ 
  - $H1 = \text{HASH}(\text{KEY xor IPAD} \parallel M)$
  - $\text{HASH}(\text{KEY xor OPAD} \parallel H1)$
  - $H(K \parallel M)$  方式存在哈希长度扩展攻击

## 题目示例

### 2020-Final-7

## 防抵赖

- MAC虽然可以提供认证，但是不能防止抵赖
- 当A发送给B一条消息，事后如果A发现此行为对自己不利，他可以否认曾经发送过这样的消息，即使消息中带有MAC

- 因为MAC的密钥是双方共享的，所以A可以声称发送给B的消息是B自己伪造的
- B也确实有能力伪造一条消息，声称其来自A，消息中可以附上MAC

## RSA签名与盲签名

RSA 章节已包含。

## EIGamal签名

需要用到的性质：

```

1  如果 g 是 p 的原根(pow(g, i, p) 两两不相同), 那么:
2       $g^i \equiv g^j \pmod{p}$ , 当且仅当  $i \equiv j \pmod{p-1}$  同
余。
3
4   $g^i = g^j \pmod{p}$ 
5   $g^{(j+k(p-1))} = g^j \pmod{p}$ 
6   $g^{(p-1)} = 1 \pmod{p}$ 
```

```

1  Alice 公开信息: p, g, A 作为公钥, a 为私钥。  $A = p$ 
ow(g, a, p)
2
3  签名:
4      选择随机数K, 要求与p-1互素
5      计算 $S1 = \text{pow}(g, K, p)$ 
6      计算 $S2 = \text{inverse}(K, p-1) * (m - a*S1) \pmod{p}$ 
-1
7       $(S1, S2)$ 作为签名
8
9  验证:
10      $\text{pow}(g, m, p) \equiv \text{pow}(A, S1, p) * \text{pow}(S1, S2,$ 
     $p) \pmod{p}$ 
11  证明:
12      $\text{pow}(g, m, p) \equiv \text{pow}(A, S1, p) * \text{pow}(S1, S2,$ 
     $p) \pmod{p}$ 
13      $= \text{pow}(g, a*S1, p) * \text{pow}(g, K*i$ 
     $\text{inverse}(K, p-1) * (m-a*S1), p) \pmod{p}$ 
14      $= \text{pow}(g, a*S1 + K*\text{inverse}(K,$ 
     $p-1) * (m-a*S1), p) \pmod{p}$ 
15      $m = a*S1 + K*\text{inverse}(K, p-1)*(m$ 
```

```
16 | -a*S1) mod p-1  
     m = m mod p-1
```

## DSA

```
1 | 全局公钥:  
2 | 大素数p  
3 | 素数q, 满足 $q \mid (p-1)$   
4 | g, 满足 $g = h^{(p-1)/q} \pmod{p}$ , 其中h满足 $\text{pow}(h, (p-1)/q, p) > 1$   
5 | 私钥:  
6 | x 为[1, q]之间的随机数  
7 | 公钥:  
8 | y =  $\text{pow}(g, x, p)$   
9 |  
10 | 签名:  
11 | 取 k, 为[1, q]之间的随机数  
12 | r =  $\text{pow}(g, k, p) \pmod{q}$   
13 | s =  $[\text{inverse}(k, q) * (\text{H}(M) + x * r)] \pmod{q}$   
14 | 验证:  
15 | w =  $\text{inverse}(s, q)$   
16 | u1 =  $(\text{H}(M) * w) \pmod{q}$   
17 | u2 =  $r * w \pmod{q}$   
18 | v =  $\text{pow}(g, u1, p) * \text{pow}(y, u2, p) \pmod{q}$   
19 | 验证 v == r  
20 | 证明:  
21 | r == v  
22 |  $\text{pow}(g, k, p) == \text{pow}(g, \text{H}(M) * w + x * r * w, p) \pmod{q}$   
23 |  $= \text{pow}(g, (\text{H}(M) + x * r) * (\text{inverse}([\text{inverse}(k, q) * (\text{H}(M) + x * r)]))) \pmod{q}$   
24 |  $= \text{pow}(g, (\text{H}(M) + x * r) * (\text{inverse}(\text{inverse}(k, q)) * \text{inverse}(\text{H}(M) + x * r, q))) \pmod{q}$   
25 |  $= \text{pow}(g, k, p) \pmod{q}$ 
```

相比RSA中n的分解pq都是不公开的，DSA只有私钥是不公开的。

## 题目示例

先跳过。。。

全局参数: p,q,g

私钥: x

公钥:  $y = \text{pow}(g, x, p)$

签名过程:

$r = \text{pow}(g, k, p)$

$e = H(r||M)$

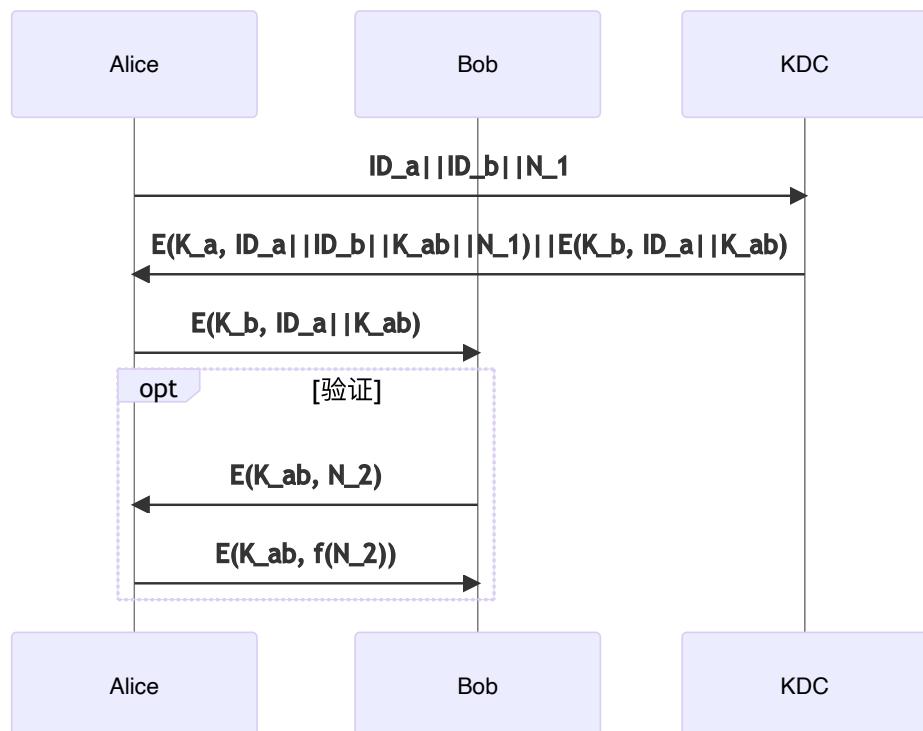
$s = k - e^*x \bmod q$

公开(e,s)对。

## 密钥分发

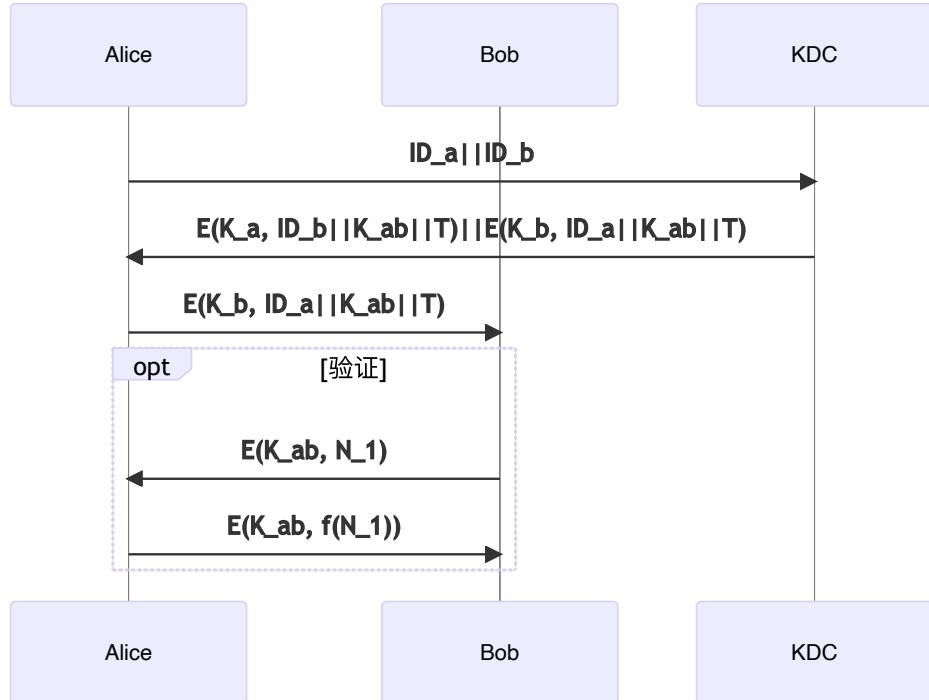
### 1 基于对称加密的共享密钥分发

#### 方案1



- 攻击者无法通过被动攻击获取密钥
- 验证部分用于防止部分重放攻击
- 如果历史 $K_{ab}$ 泄露，这种方式是否安全？
  - 不安全，如果历史 $K_{ab}$ 泄露，攻击者有能力对加密后的 $N_2$ 解密，并构造响应消息

## 方案2

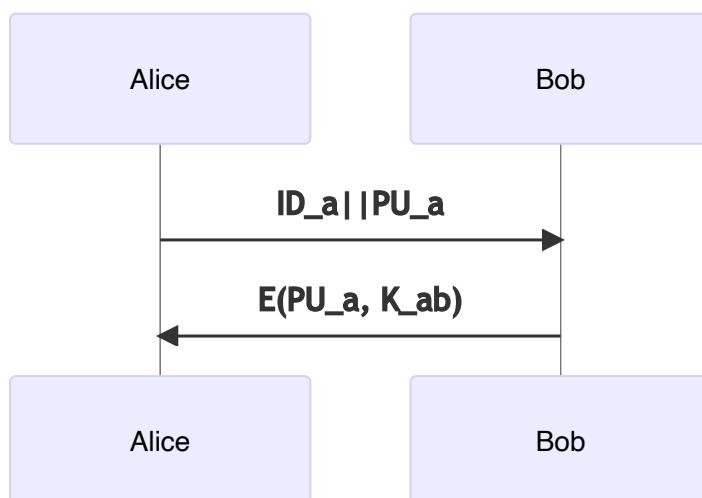


- 需要一个全局同步的时钟

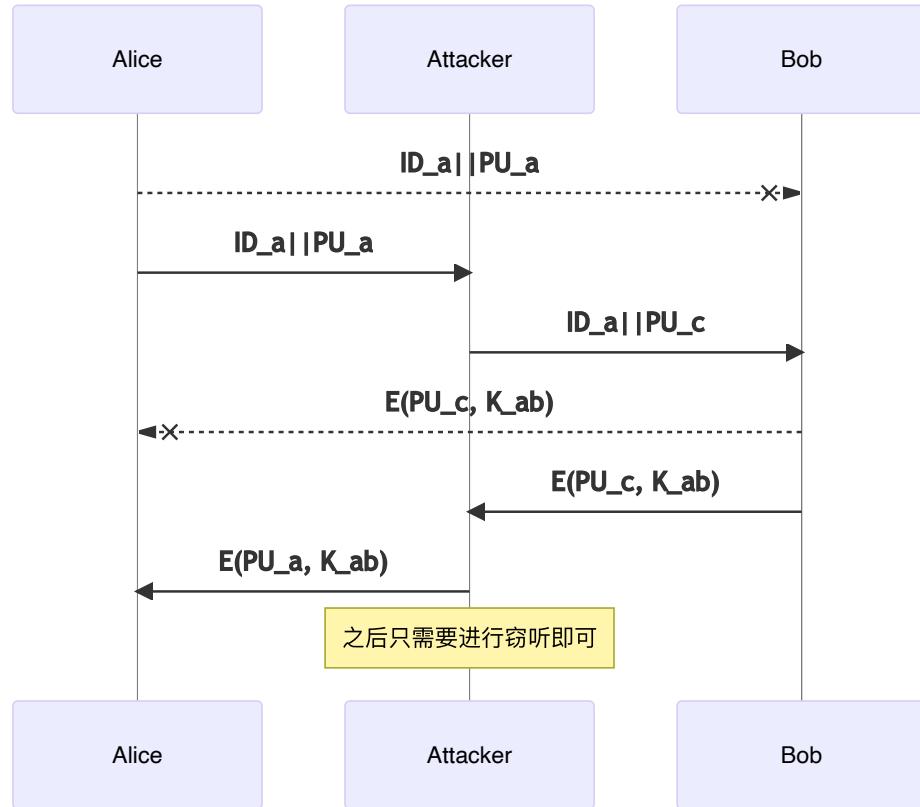
可能的题目出法：标准的场景中去除nonce(请求-应答)或是时间戳，那么就可能存在重放攻击。

## 2 基于非对称加密的共享密钥分发

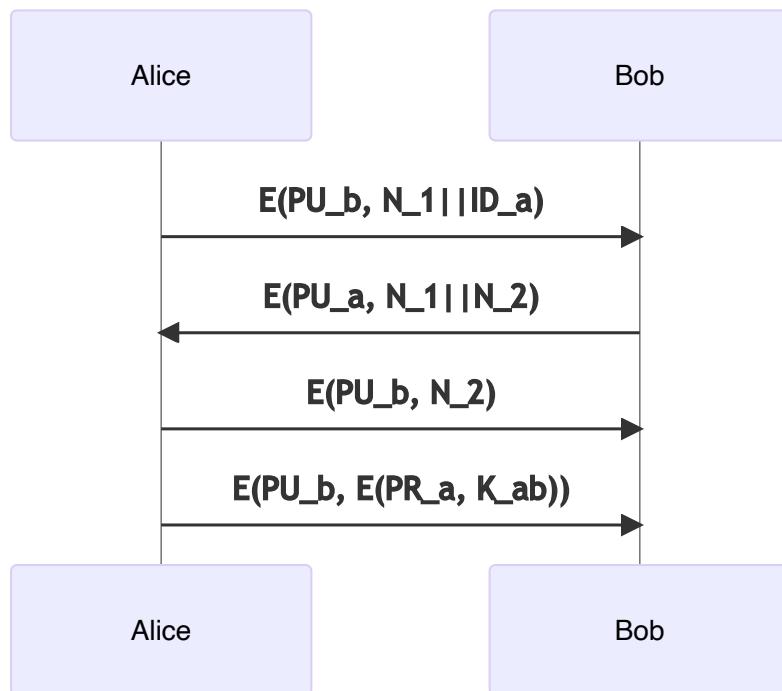
### 方案1



这种方案可能遭受中间人攻击：



## 方案2



- 这种方案可以提供保密性和认证

- 需要提前安全的交换公钥

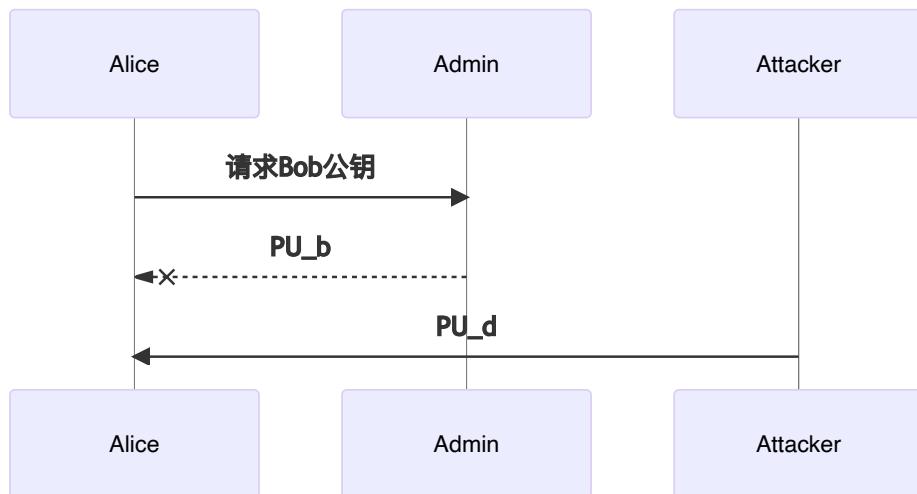
### 3 公钥分发

#### 方案1 广播分发

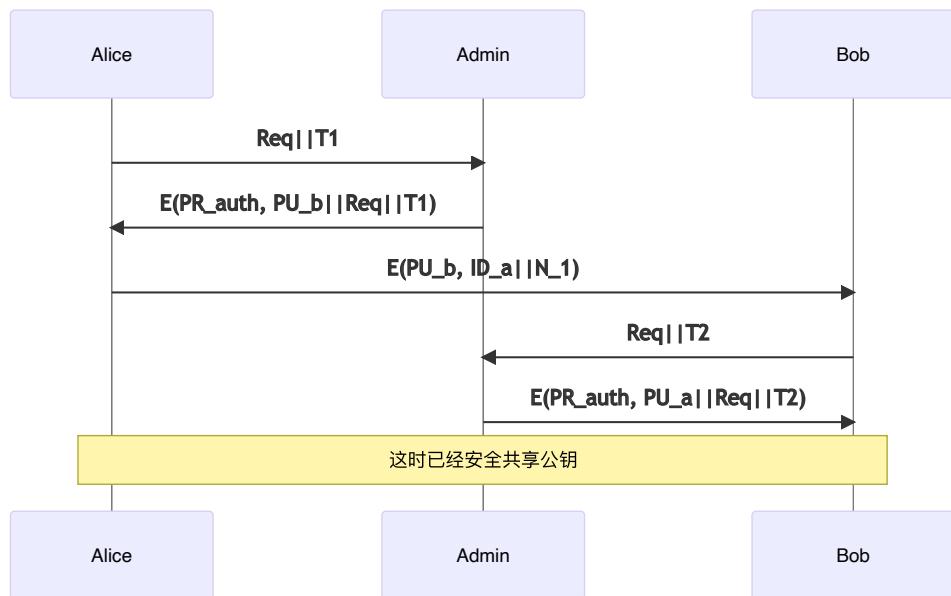
- 没有提供认证的情况下，任何人都可以以其他人名义分发公钥
- 如果消息发送者使用错误的公钥加密，机密性被破坏

#### 方案2 受管理的公开目录

- 任何人都要通过管理员注册公钥，这时需要认证
- 公钥放在一个公开目录用于查询
- 需要保证公开目录文件的完整性
- 如果系统被攻破，则公钥信息不可信任
- 即使系统中文件完整性没有被破坏，系统对查询的响应也可能被篡改



#### 方案3 公钥授权



- 每次通信都依赖Admin
- Admin需要保证其管理信息的完整性

## 用户认证

### 用户认证定义

向系统证明一个用户就是它所声称的那个用户。

认证分为下面两步：

- 鉴定阶段：向系统提供身份标识（如用户名、id）
- 核实阶段：提供或产生可以证实实体与标识对应关系的认证信息（如口令）

### 认证方式

- 知道什么：口令
- 拥有什么：智能卡
- 生物特征：指纹、面容、笔迹等信息
- 多因素认证：结合多种方式实现认证

### 重放攻击

用户认证过程中需要考虑重放攻击，例如一个使用口令认证的系统中，如果使用了相对不安全的方式加密。攻击者可以保存核实阶段发送的流量信息并在后续过程中发送，从而伪装成特定用户。

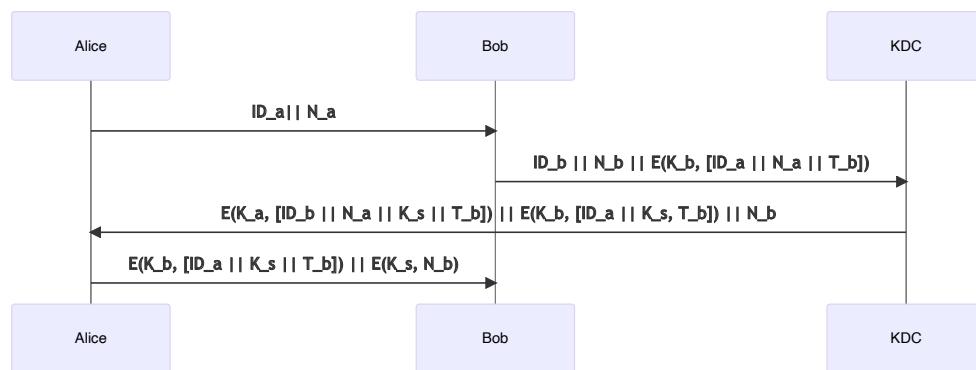
为了防止重放攻击，有三种常见方法：

1. 为认证交互过程中每一条消息都附加序列号，只有满足时序才认为是合法消息。
2. 消息中包含时间戳，接收者判断时间戳不能距离当前时间过久，这就需要全局同步的时钟。
3. 挑战/应答，当A接受B消息前会向B发送一个挑战，并要求B的消息中带有响应。

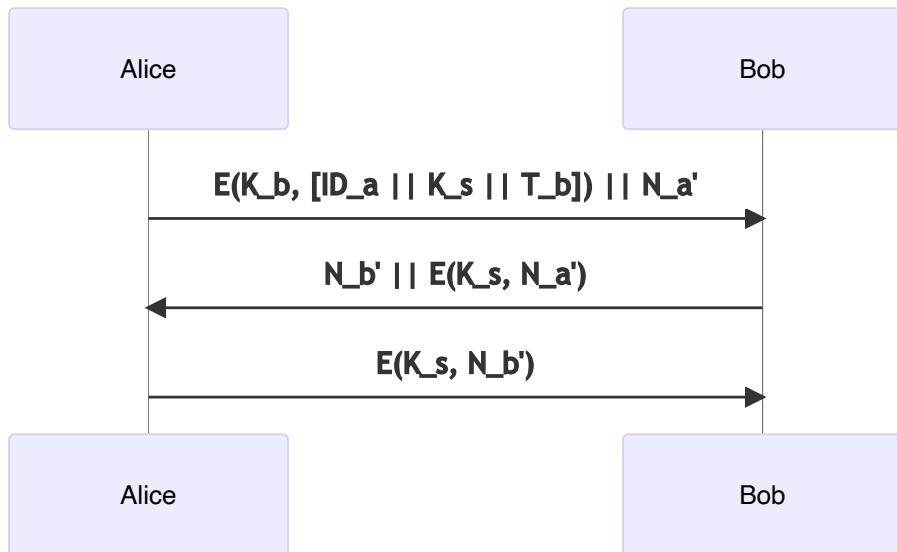
## 基于对称加密双向认证

N-S 方案及 Denning 的改进在密钥分发章节已经提到。因为 Denning 的方案需要同步的时间戳，如果因为意外或攻击者刻意控制导致发送方时钟比接收方时钟快，依然可能受到重放攻击。

进一步的改进方案如下：



上面过程中，A和B收到的消息分别包含 $N_a$ 和 $N_b$ ，来确保消息不是重放的。此方案的另一个优点在于A可以在一段时间内重新建立会话而不需要KDC参与，步骤如下：



B 通过检查  $T_b$  来判断这次交互是否过期，然后用  $N_a'$  和  $N_b'$  保证消息不被重放。

因为整个过程中  $T_b$  仅由 B 产生和检查，所以不存在时钟同步问题。

## 题目示例

**2020-Final-1**

**2020-Final-9**

提示：第二类重放。

**2020-Final-10**