# Crypto_p1

## 开始

### 内容分布

- 总体介绍 - p1
- 古典密码 - p2
- 对称密码 - p3
- 数学基础 - p4 p5 p6 p7
- 非对称密码 - p8 p9 p10 p14 p18
- 哈希函数 - p11
- MAC - p12
- 数字签名 - p9 p10 p17 p18 p19(ElGamal签名)
- 密钥管理 - p13 p15 p16
- 认证 - p20 p21 p22

### 常见概念

- CIA：机密性、完整性和可用性，Confidentiality, Integrity, and Availability
- 密码编码学：设计密码系统，Cryptography - Study of encryption principles/methods。
- 密码分析学：破解密码系统，Cryptanalysis (codebreaking) - Study of principles/ methods of deciphering Ciphertext without knowing key。
- Kerckhoffs 原则：一个密码系统的安全性应该只取决于密钥的保密性，而不是算法的隐秘性。
- 无条件安全：基于信息论的安全，即使攻击者的资源是无限的，依然无法破解，一次一密。
- 计算安全：使得有限资源的攻击者无法破解，或者攻击者破解密码所需资源远大于资源本身。

### 工具

#### Python

- 安装方式参考：https://www.python.org/
- 第三方推荐：ipython，pycryptodome，gmpy2

#### sage

- 安装方式参考：https://www.sagemath.org/
- 临时使用：https://sagecell.sagemath.org/

## 古典密码

如何安全的传递信息，从公元前至今都是经典问题。

补充一下公开课没有详细讲的爱情故事：

```
密文：****-/*----/----*/****-/****-/*----/---**/*----/****-/*----/-
****/***--/****-/*----/----*/**---/-****/**---/**---/***--/--***/****-/
```

```
第一层：4194418141634192622374

第二层：41 94 41 81 41 63 41 92 62 23 74

第三层：gzgtgogxncs

第四层：otoeoiouyvl

第五层：i love you too
```

## 常见非加密技术

### 编码

- 编码通常没有密钥
- 编码的目的不是为了保护机密性
- 常见的编码：摩斯密码、hex编码、base64编码

### 隐写

- 用于隐藏秘密信息，但是和加密算法的思想不同
- 加密算法将信息加密成攻击者读不懂的密文
- 隐写则是为了让攻击者察觉不到信息的存在
- 常见隐写：图片低比特隐写、盲水印技术

## 代替和置换

- 代替：位置不变，内容变
- 置换：内容不变，位置变

## 凯撒密码

- 其实不应该算密码，因为没有密钥
- 但是老师ppt上的算，因为有密钥： )
- p c 属于[0, 26)
- c = p + k mod 26
- p = c - k mod 26
- p1(长度32) 使用 k 加密，得到c1(长度32)
- p2(长度1) 使用 k 加密，得到c2(长度1)

攻击者已知 p2, c2，c1

- c - p = k mod 26
- k = c1 - p1 mod 26

- 根据 k 和 c1 解密得到 p1
- 已知明文-密文对可以快速分析
- 频率分析

### 仿射密码*

- key为(a, b)
- p c 属于[0, 26)
- c = a*p + b mod 26
- p = (c - b) * inverse(a) mod 26
- 需要一点点数学知识
- c = a*p + b mod 26
    - c - b = a*p mod 26
    - (c-b)*inverse(a) = a*inverse(a)*p mod 26
    - (c-b)*inverse(a) = p mod 26
- Ek = (a,b)
- Dk = (a' , b) # a' = inverse(a, 26)

```
假设 b = 0, a = 17
c = 17 * p mod 26
p = c * 23 mod 26
  = (17 * p mod 26) * 23 mod 26
  = 17 * p * 23 mod 26
  = 391 * p mod 26
  = 1 * p mod 26

在此基础上，已知 p2 = 1, c2 = 17
17 = a + b mod 26
不足以分析出key
额外的，知道 p3 = 2, c3 = 8
17 = a + b mod 26
8 = 2*a + b mod 26
```

### 代替表

- 使用替换表作为 key，创建加密表和解密表（类似密码本）
- 加密和解密都是查表操作

例子：

- 简化的语言仅有ABC三个字母 key = 'CBA'（其实解密表也是 'CBA'）

- p1 = "AACBBA" c1 = "CCABBC"

## 维吉尼亚密码

- 类似凯撒，加上密钥
  - plain "aaabcbcdefefef"
  - key "abc" -> "0 1 2"
  - cipher "abcbdd"
- key会被循环利用
- 如果key和明文等长，类似一次一密

## 简单的置换-栅栏

- 纵向写，横向读。
- key为栅栏深度
- 加密

```
明文:abcdefg
2层栅栏:
a c e g
b d f
密文:acegbdf
----------
明文: ls rm clear
栅栏:
l mcer
sr la
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

```
明文: abcdefg
key为: [2,1]
加密:
2 a c e g
1 b d f
密文: bdfaceg
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]

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

- K(3x3)
- C(3x3) = K(3x3) * P(3x3)
- K'(3x3)*C(3x3)=K'*K * P(3x3)

```
单位阵:
1 0 0
0 1 0
0 0 1
K:
0 1 0
1 0 0
0 0 1
P:
1 5 2
3 6 4
8 7 9
(加密)C = K*P:
3 6 4
1 5 2
8 7 9
(解密)P = (K^(-1))*C
```

# 补充内容

如何求逆元?

我们的目标是对于 A 和 N 找到 A 的逆元 B，使得 `A*B = 1 (mod 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 +b*N = 1`，则 `A*x = 1 (mod N)`，x 即为一个满足条件的 B。

扩展欧几里得算法是怎样求出 x, y的?

```
gcd(a, b) == gcd(b, a%b)
```

```
假设a,b最大公因数为X,gcd递归下降的终点为1*X+0*y == X, 在gcd算法向上回溯的过程中总有:
X = nx*a + ny*b
X = x*b + y*(a%b)
这里 a%b 写成 a-k*b, 其中 k = a//b,则
X = b*x + a*y - k*b*y
X = b*x +(a-k*b)*y
X = b*x + a*y - k*b*y
X = a*[y] + b*[(x-k*y)]
那么 nx, ny = y, (x-a//b*y),由此递推计算x, y
```

下面是 Python 代码实现:

```
def XGCD(a, b):
    if (b == 0):
        return a, 1, 0
    gcd, x, y = XGCD(b, a%b)
    return gcd, y, x-a//b*y


def inverse(a, n):
    gcd, x, y = XGCD(a, n)
    return x%n if gcd == 1 else None
```