2022

浙江大学 ZHEJIANG UNIVERSITY

密码学基础

戴勤明

浙江大学

什么是密码学

密码学是**研究编制密码和破译密码的技术科学**。

1. 设计加解密算法

2. 破解加解密算法

为什么需要密码学

- 存储：信息的存储可能是不安全的，会被窃取
- 传输：信息的传输过程可能也不是隐秘的，会被窃听

不能直接使用明文进行存储和传输！

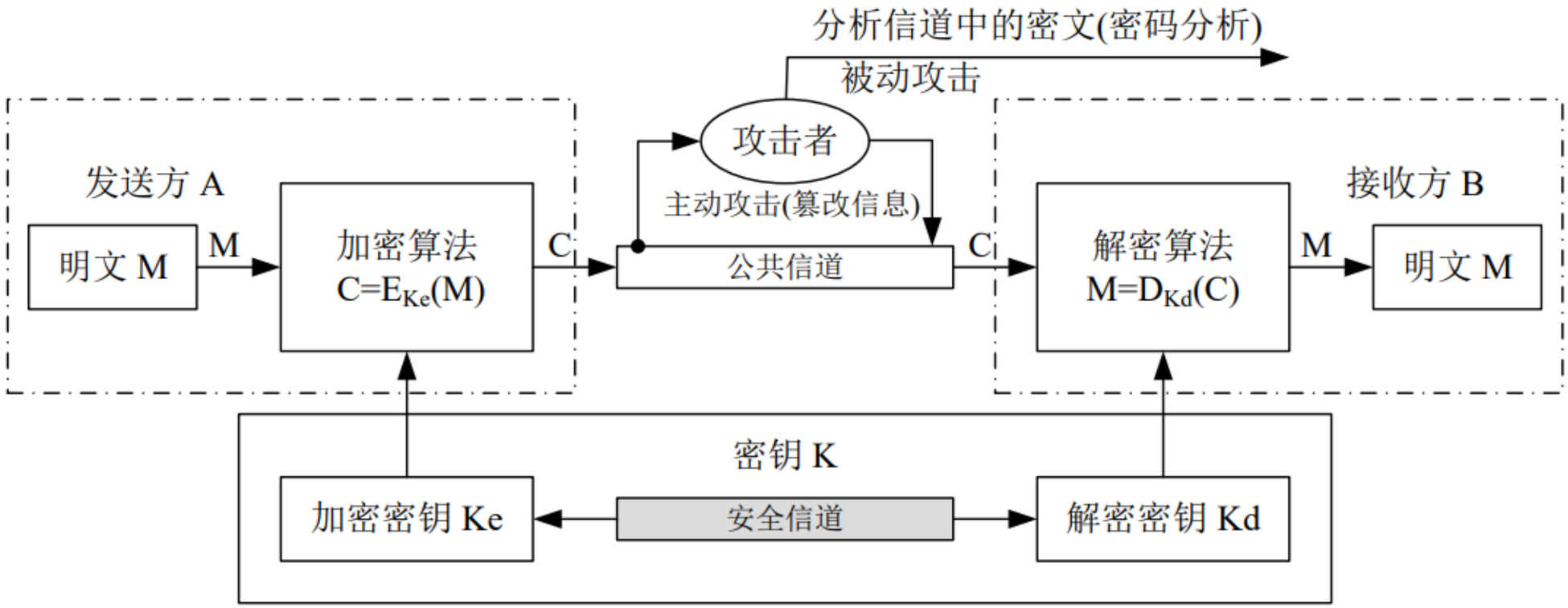
基本术语

- 消息被称为明文（Plaintext）。用某种方法伪装消息以隐藏它的内容的过程称为加密（Encryption），被加密的消息称为密文（Ciphertext），把密文恢复为明文的过程称为解密（Decryption）。

- 密码算法（Cryptography Algorithm）：是用于加密和解密的数学函数。

- 密钥（Key）：加密或解密所需要的除密码算法之外的关键信息。

基于密码学的保密通信系统



纲要

1. 数学基础
2. Python基础
3. 古典密码
4. 流密码
5. 对称密码
6. 非对称密码

数学基础

1. 整除
2. 素数与互素
3. 最大公约数
4. 模（mod）运算和同余
5. 逆元

以上材料来自http://10.71.45.100/bhh/crypto.doc

整除

整除的定义: 设a、b均为整数,且a≠0, 若存在整数k使得b=a*k，则称a整除b，记作a|b。

整除相关的3个命题:

①对于任意整数a，都有1|a；若a≠0，则有a|0且a|a。

②若a|b且b|c，则a|c。

③若a|b且a|c，则a|(s*b+t*c)，其中s、t为任意整数。

以上材料来自http://10.71.45.100/bhh/crypto.doc

素数与互素

素数的定义: 若整数p只有因子±1及±p, 则称p为素数。

互素(relatively prime)的定义:对于整数a、b，若gcd(a,b)=1，则称a、b互素。

（gcd: Greatest Common Divisor）

a=3, b=5 gcd(a,b)=1

a=3, b=4 gcd(a,b)=1

a=4, b=9 gcd(a,b)=1

素数相关的定理: 任一整数a(a>0)都能唯一分解成以下形式:

a = p1 * p2 * p3 * ... * pt

其中p1、p2、p3、...、pt是素数。

以上材料来自http://10.71.45.100/bhh/crypto.doc

最大公约数（gcd）

最大公因数是指能够整除多个整数的最大正整数。

gcd相关的定理: 设a、b为整数，且a、b中至少有一个不等于0，令d=gcd(a,b)，则一定存在整数x、y使得下式成立:

a*x + b*y = d

特别地，当a、b互素时，则一定存在整数x、y使得a*x+b*y=1成立。

思考：如何快速计算gcd？

以上材料来自http://10.71.45.100/bhh/crypto.doc

模(mod)运算和同余

同余的定义: 设a、b、n均为整数，且n≠0，当a-b是n的倍数时即a=b+n*k(k为整数)，我们称a、b对于模n同余(a is congruent to b mod n)，记作:

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

可以理解为: a%n == b %n

例如: 1 ≡ 4 mod 3

例如: 5 ≡ 8 mod 3

以上材料来自http://10.71.45.100/bhh/crypto.doc

模(mod)运算和同余

同余相关的命题: 设a,b,c,d,n均为整数，且n≠0，则有

①当且仅当n|a时，有a≡0 (mod n)

②a≡a (mod n)

③当且仅当 b≡a (mod n)时, 有a≡b (mod n)

④若a≡b 且 b≡c (mod n)，则一定有a≡c (mod n)

⑤若a≡b (mod n)且c≡d (mod n)，则有

a+c≡b+d, a-c≡b-d, a*c≡b*d (mod n)

以上材料来自http://10.71.45.100/bhh/crypto.doc

逆元（inverse）

- 加法模逆元

定义: 若a+b≡0 (mod n), 则称a是b的加法模n逆元，b是a的加法模n逆元。

- 乘法模逆元

定义: 若a*b≡1 (mod n), 则称a是b的乘法模n逆元，b是a的乘法模n逆元。a的乘法逆元记作a^{-1}。

例如：求13模35的乘法逆元

设13模35的乘法逆元为x，则

$$13*x \equiv 1 \pmod{35}$$

上述等式成立的充要条件为gcd(13,35)=1

思考：如何求解inverse？

Python基础

不同于C，Python是一种脚本语言，不需要编译便可以直接执行。

脚本语言通常设计的比较简洁，方便写一些具有小功能的程序。

```
print("Hello, world!")
```

执行： python3 test.py

Python环境

在Ubuntu里面，python3是预装的。

在命令行中键入python3然后回车就能看到：

```
➜ basic python3
Python 3.8.10 (default, Mar 15 2022, 12:22:08)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>>
>>>
>>> print("hello world")
hello world
>>>
```

Python变量类型

- Numbers
- String
- List
- Tuple（元组）
- Dictionary（字典）

Python运算符

- +-*/
- %
- **
- ^
- //

Python格式化输出

假设a,b是两个变量（number,string,list...）

```
print("output:",a,b)
```

```
print(f"output: {a} {b}")
```

```
print("output: {} {}".format(a, b))
```

Python条件语句if

if 判断条件1:

　　执行语句1……

elif 判断条件2:

　　执行语句2……

elif 判断条件3:

　　执行语句3……

else:

　　执行语句4……

```
flag = False
name = 'aaa'
if name == 'aaa':
    flag = True
    print('It\'s aaa')
else:
    print("Not aaa")
```

Python循环语句for

```
for i in range(10):
    print(i)


for letter in 'Python':     # 第一个实例
   print("当前字母:", letter)


fruits = ['banana', 'apple',  'mango']
for fruit in fruits:        # 第二个实例
   print ('当前水果:', fruit)
```

Python循环语句while

```
count = 0

while (count < 9):

   print('The count is:', count)

   count += 1
```

Python嵌套循环

```
for iterating_var in sequence:
    for iterating_var in sequence:
        statements(s)
```

Python break&continue

```
for letter in 'Python':
   if letter == 'h':
      break
   elif letter == 't':
      continue
print('letter:', letter)
```

Python List

```
list1 = [1, 2, 3, 4, 5, 6, 7]
list1[-1] # 下标范围[-7,6]
# 7
list1[1:4]
# [2, 3, 4]
list1[1:]
# [2, 3, 4, 5, 6, 7]
list1[:4]
# [1, 2, 3, 4]
list1[:-1]
# [1, 2, 3, 4, 5, 6]
list1[::-1]
# [7, 6, 5, 4, 3, 2, 1]
list1[0]=100
```

Python List

```
list1 = [1, 2, 3, 4, 5, 6, 7]
list2 = ['physics', 'chemistry', 1997, 2000]

list1+list2
# [1, 2, 3, 4, 5, 6, 7, 'physics', 'chemistry', 1997, 2000]
list1.append(8)
# [1, 2, 3, 4, 5, 6, 7, 8]
list1.append(list2)
# [1, 2, 3, 4, 5, 6, 7, 8, ['physics', 'chemistry', 1997, 2000]]
```

Python Tuple

```
tuple1 = (1,2,3)
tuple2 = ("a", "bb")
tuple1+tuple2
# (1, 2, 3, 'a', 'bb')
tuple1[1:]
# (2, 3)
tuple1[0]=100  非法操作
```

Python Dictionary

由一系列的键值对构成

```
dict1 = {'a': 1, 'b': 2, 'b': '3'}
dict1['b']
# '3'
dict1
# {'a': 1, 'b': '3'}
dict1['a'] = 'aaa'
# {'a': 'aaa', 'b': '3'}
```

Python 函数

```
def foo(params):
    #.......
    #.......
    print(params)
    return 1,2
```

Python 库

和C一样，python中也能导入外部库然后调用其中的函数

```python
import random

print(random.getrandbits(32))
```

```python
from random import getrandbits

print(getrandbits(32))
```

```python
from random import *

print(getrandbits(32))
print(randint(10, 100))
```

Python 库

安装库

pip3 install pycryptodome

古典密码

- 代换（substitution）密码——用新的替换原先的内容
- 置换（permutation）密码——打乱原先的顺序
- Hill密码
- 其他编码算法

代换密码

代换是古典密码中用到的最基本的处理技巧。所谓代换，就是将明文中的一个字母由其他字母、数字或符号替代的一种方法。

- 凯撒密码
- 仿射密码
- 单表代换
- 多表代换

凯撒（caesar）密码

已知最早的代换密码，又称移位密码。

“它是一种替换加密的技术，明文中的所有字母都在字母表上向后（或向前）按照一个固定数目进行偏移后被替换成密文。例如，当偏移量是3的时候，所有的字母A将被替换成D，B变成E，以此类推。这个加密方法是以罗马共和时期恺撒的名字命名的，当年恺撒曾用此方法与其将军们进行联系。”—— 维基百科

- 代换表（密钥）：

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

- 数学描述：

C = E(P) = (P + K) mod 26 (C: Cipher, E: Encrypt, P: Plain, K: Key)
P = D(C) = (C – K) mod 26 (D: Decrypt)

- 攻击

穷举攻击，对Key进行爆破（即穷举）。

仿射（Affine）密码

- 数学形式

$y = E(x) = a*x+b \bmod n \quad (\gcd(a,n) = 1)$

$x = D(y) = a^{-1}*(y-b) \bmod n$

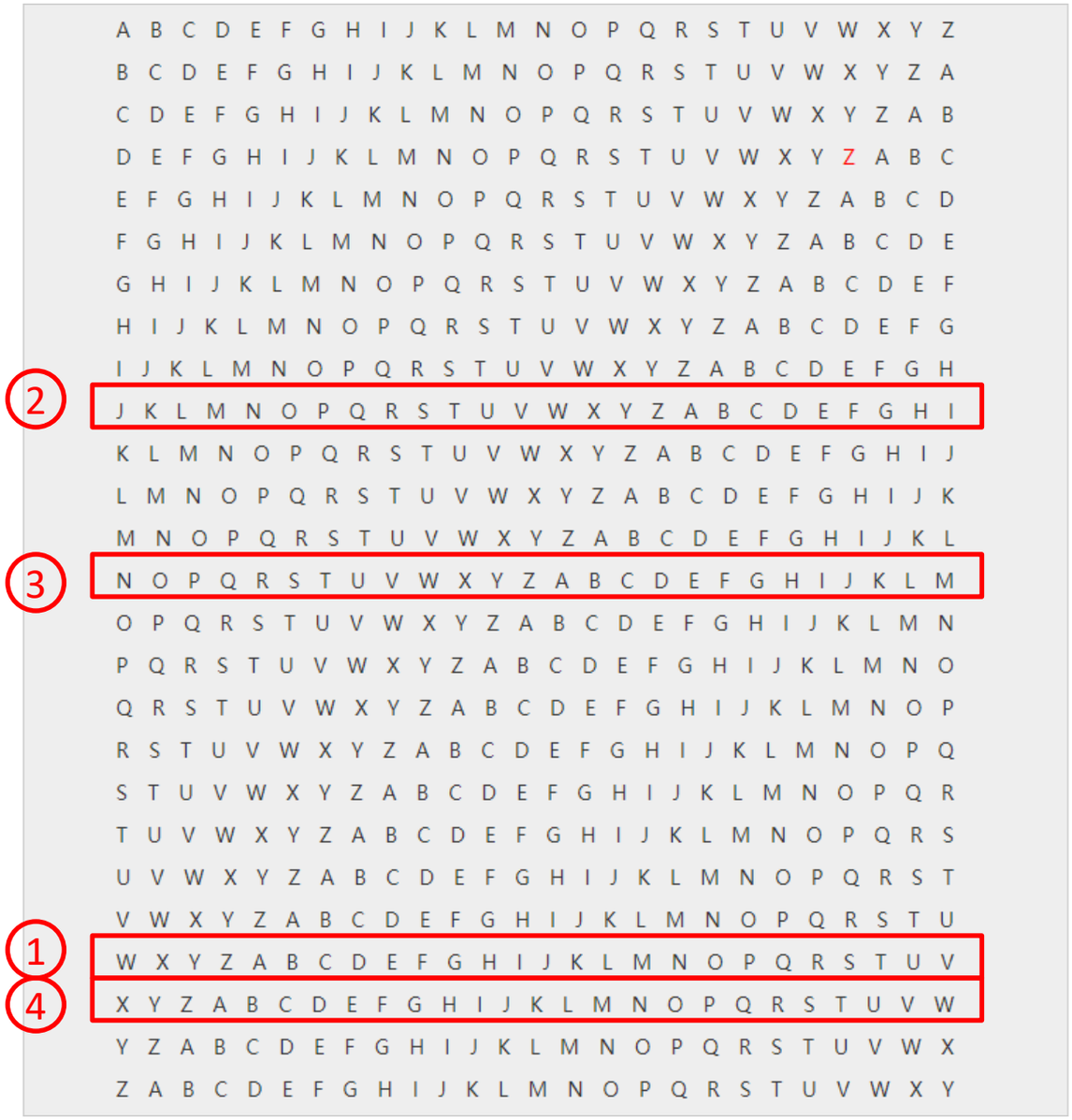
凯撒密码就是仿射密码的一种特殊格式：a=1，n=26

单表代换密码

类似于凯撒密码，但是密码表是打乱的，一一对应的。

多表代换密码

利用多个表进行加解密，典型代表是维吉尼亚密码



维吉尼亚密码表

每一行相当于一个（凯撒）密码表，例如第二行就是key=1的凯撒密码表。

而在维吉尼亚密码中，key是一个代表偏移的字符串，例如key=wjny，对应到凯撒中就是22，9，13，24。

假设目前需要加密明文P=crypto，那么加密过程如下：

（1）针对第一个明文字符c，利用key=22进行凯撒加密，得到y

（2）针对第一个明文字符r，利用key=9进行凯撒加密，得到a

（3）y -> l

（4）p -> m

（5）t，使用key=22（循环），得到p

（6）o -> x

因此最终密文C=yalmpx

维吉尼亚（vigenere）密码

简单尝试一下：密钥cat，明文thisissecret，密文？

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

维吉尼亚密码表

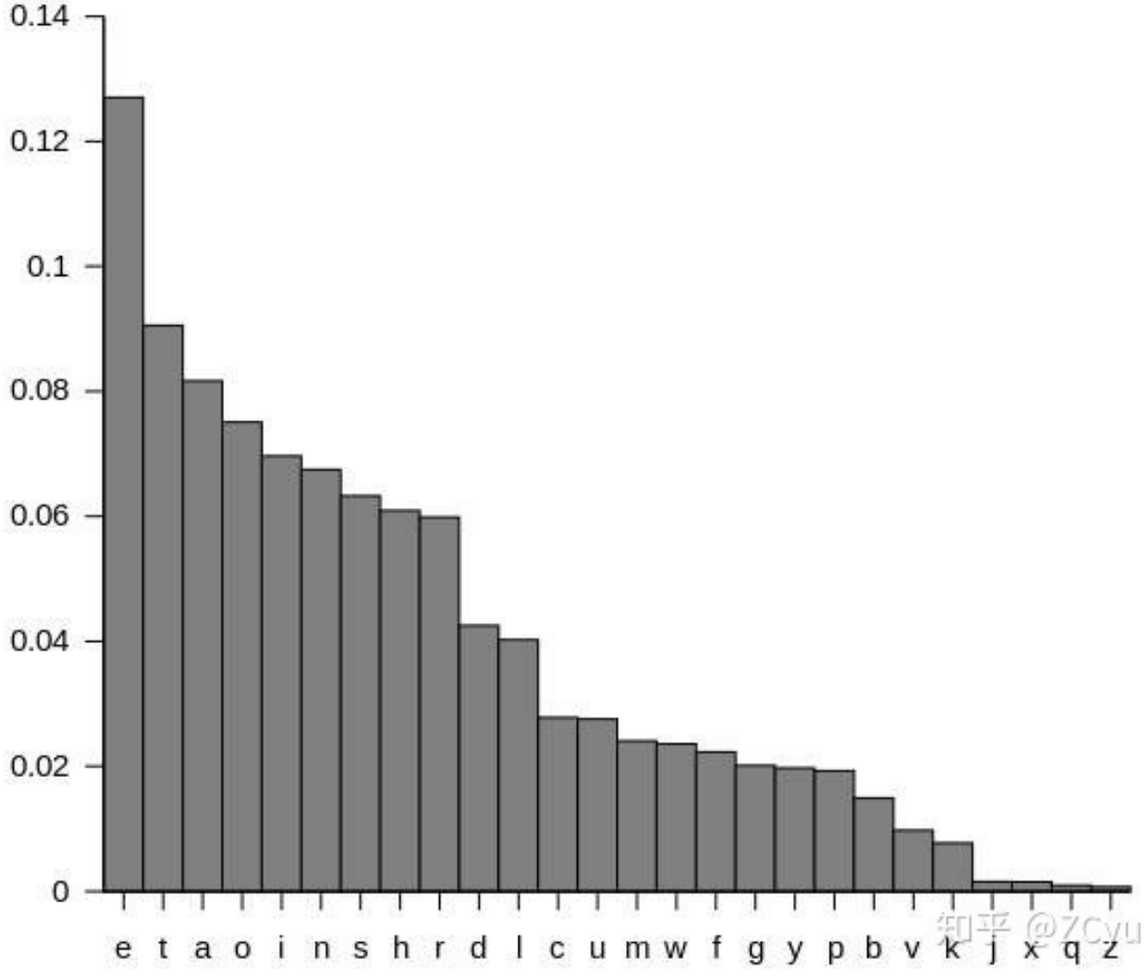
vhbuiluevtem

维吉尼亚密码破解

$$C = (P_1 + K_1, P_2 + K_2, \ldots, P_m + K_m) mod 26$$

$$P = (C_1 - K_1, C_2 - K_2, \ldots, C_m - K_m) mod 26$$

移位密码、仿射密码和单表代换密码都没有破坏统计规律，所以我们可以直接根据字母的频率分析进行破解。



维吉尼亚密码破解

频率排序：

两个字母： th, he, in, er, re, on, an, en

三个字母： the, and, tio, ati, for, tha, ter, res

维吉尼亚密码破解

步骤：

1. 确定密钥的长度
2. 确定密钥的内容
3. 根据密钥恢复明文。

维吉尼亚密码破解

确定密钥的长度

破解维吉尼亚密码的关键在于它的密钥是循环重复的。如果我们知道了密钥的长度m，破解难度就会降低。

将相同字母组找出来，然后求距离差的最大公约数gcd()，由此猜测m。

维吉尼亚密码破解

Fhovq abi mn ypp hyvee krp a vmftvi tobwq ox e rabq. Ano hmy dlq ovh tobwq acoe tri xidxxe rsdso xa sorp tri ihoef ty xte wmxl. Dlq lsxflo larci us fidy rebpi. Lq ckvdiow fho atekx mnn vgnc xawkvp tri yivp. Nud xtebi us k vuvov un pvand sr tri xidxxe rsdso. Lq sdsbs krp dyie nyx wnya ihkx fo ns zehx. Vucx fhor Muxx Oog me pkweixk ny.Dlq lsxflo larci msuw, "Muxx Oog, txekwq topx mo. Gmn S gdocw fho vuvov ".Muxx Oog ezsgids, "Sx us xsf doib, yyy oax gdocw ut."Glqn dlq lsxflo larci neqmzs ds orywe tri difid, a vmftvi eqemdrop ehyyfs kx tiw, "Putdpqhyvee, nszt mvasc mf, yyy iivp ne nvawxip. Yowfebhmy yrq op qk fbmqnnw iac hdogrqd sr fhsw difid."Tri xidxxe rsdso me vovk apvmin. Junkpxy ri pemmpec xa gy lamo ezd kww hsw yodlqr.Dlq ovh tobwq acoe, "Wrc po isg tkoq tri ihoef bkgw Wrefs gvanq autr cau Wcohspp."Tri xidxxe rsdso ezsgids cepli, "Xtebi us k vuvov un pvand sr mo. Egnd Gaw ceud sx iac rat niqp. Lyf tri xidxxe cugibvql ceud sx iac hqez. Atad wtavp U dy ".Xte ypp hyvee ceks, "Wc ohspp, yyy ehyyxd dvk ty gdocw fho vuvov ny isgrcixf. Sj koe hanyx fri, law ns koe ozog xte bmheb me doib ob rat ".Dlq lsxflo larci oabvuec xte glqad ezd bifubre ty xte bmhebwudo. Ef lkwf, ho wgcmiqdc mzcbsessrs tri difid. Nya, Te urawc law niqp dlq rszqr sw.

维吉尼亚密码破解

Fhovq abi mn ypp hyvee krp a vmftvi tobwq ox e rabq. Ano hmy dlq ovh tobwq acoe **tri** xidxxe rsdso xa sorp **tri** ihoef ty xte wmxl. Dlq lsxflo larci us fidy rebpi. Lq ckvdiow fho atekx mnn vgnc xawkvp **tri** yivp. Nud xtebi us k vuvov un pvand sr **tri** xidxxe rsdso. Lq sdsbs krp dyie nyx wnya ihkx fo ns zehx. Vucx fhor Muxx Oog me pkweixk ny.Dlq lsxflo larci msuw, "Muxx Oog, txekwq topx mo. Gmn S gdocw fho vuvov ".Muxx Oog ezsgids, "Sx us xsf doib, yyy oax gdocw ut."Glqn dlq lsxflo larci neqmzs ds orywe **tri** difid, a vmftvi eqemdrop ehyyfs kx tiw, "Putdpqhyvee, nszt mvasc mf, yyy iivp ne nvawxip. Yowfebhmy yrq op qk fbmqnnw iac hdogrqd sr fhsw difid."**Tri** xidxxe rsdso me vovk apvmin. Junkpxy ri pemmpec xa gy lamo ezd kww hsw yodlqr.Dlq ovh tobwq acoe, "Wrc po isg tkoq **tri** ihoef bkgw Wrefs gvanq autr cau Wcohspp."**Tri** xidxxe rsdso ezsgids cepli, "Xtebi us k vuvov un pvand sr mo. Egnd Gaw ceud sx iac rat niqp. Lyf **tri** xidxxe cugibvql ceud sx iac hqez. Atad wtavp U dy ".Xte ypp hyvee ceks, "Wc ohspp, yyy ehyyxd dvk ty gdocw fho vuvov ny isgrcixf. Sj koe hanyx fri, law ns koe ozog xte bmheb me doib ob rat ".Dlq lsxflo larci oabvuec xte glqad ezd bifubre ty xte bmhebwudo. Ef lkwf, ho wgcmiqdc mzcbsessrs **tri** difid. Nya, Te urawc law niqp dlq rszqr sw.

维吉尼亚密码破解

20

92

124

320

528

640

864

GCD=4 → Key Len=4

维吉尼亚密码破解

确定密钥的内容

求得秘钥的关键在于，确定了秘钥长度 k 之后，可以把密文的第 1 个，第 k + 1个，第 2k+1个...密文字母提取出来组成一个新的文本，这个抽取出来的文本，都是由秘钥的第一个字符加密得到的，就相当于一个凯撒密码。

然后利用重合指数法，也很好理解。我们就把这一串提取出来的文本反向凯撒回去，26个字母一个个试，只要他给的原文文本是有意义的，符合英文文本的正常规律（字母出现的大数据频率）就可以了。

p = [0.08167, 0.01492, 0.02782, 0.04253, 0.12702, 0.02228, 0.02015, 0.06094, 0.06966, 0.00153, 0.00772, 0.04025, 0.02406, 0.06749, 0.07507, 0.01929, 0.00095, 0.05987, 0.06327, 0.09056, 0.02758, 0.00978, 0.02360, 0.00150, 0.01974, 0.00074]

维吉尼亚密码破解

FqmpepftqrAmqtqexxdapiftxqfaudbqdftmgapyNtuuuarxxdqbpewifzVfMOeenqfa

mMOxqxmdfuMOzdufboduqqfanzoeddfedeftuqezafinapfmqkqidqfddxxdekmuxp

paazwyqqtqepgqifwfauaopxxdzdptuuuargauiaqfxxgquiqttUtpekopexkdfungxkafa

kzthebaqfaoutqzfethuffgqzesddTaaqqq

Key = m

TeadsdthefOaehesllrodwthletoirperthauodmBhiiiofllrepdskwtnJtACssbetoaAClel

artiACnritpcrieetobncsrrtsrsthiesnotwbodtaeyewretrrllrsyailddoonkmeehesdue

wtktoiocdllrnrdhiiiofuoiwoetllueiwehhIhdsycdslyrtibulyotoynhvspoetocihentshvi

ttuensgrrHooeee

维吉尼亚密码破解

[0.04166666666666664, 0.020833333333333332, 0.03333333333333333,
0.05416666666666667, 0.11666666666666667, 0.0125,
0.004166666666666667, 0.0625, 0.07916666666666666,
0.004166666666666667, 0.0125, 0.06666666666666667,
0.008333333333333333, 0.0375, 0.09166666666666666,
0.016666666666666666, 0.0, 0.07916666666666666, 0.06666666666666667,
0.10416666666666667, 0.025, 0.008333333333333333,
0.029166666666666667, 0.0, 0.025, 0.0]

维吉尼亚密码破解

Key = make

There are an old horse and a little horse on a farm. One day the old horse asks the little horse to send the wheat to the mill. The little horse is very happy. He carries the wheat and runs toward the mill. But there is a river in front of the little horse. He stops and does not know what to do next. Just then Aunt Cow is passing by.The little horse asks, “Aunt Cow, please tell me. Can I cross the river ”.Aunt Cow answers, “It is not deep, you can cross it.”When the little horse begins to cross the river, a little squirrel shouts at him, “Littlehorse, dont cross it, you will be drowned. Yesterday one of my friends was drowned in this river.”The little horse is very afraid. Finally he decides to go home and ask his mother.The old horse asks, “Why do you take the wheat back Whats wrong with you Mychild.”The little horse answers sadly, “There is a river in front of me. Aunt Cow said it was not deep. But the little squirrel said it was deep. What shall I do….

置换密码

加密变换使得信息元素只有位置变化而内容不变

例如，假设置换表：

X	1	2	3	4	5	6
E(X)	3	5	1	6	4	2

明文P=crypto basic

加密过程，先分组，这里刚好两组（如果不足一组，用空格填充）：[crypto], [basic]

针对每一组用置换表进行置换：

[crypto] -> [yoctrp]

[basic] -> [ac ibs]

最终得到密文C=yoctrpac ibs

栅栏密码

不同于刚刚举的例子，栅栏密码把明文分割成k行，然后重新拼接，这里k即为加密的密钥。例如k=3，明文P=crypto basic

加密过程如下，首先按明文顺序分割成3行：

(1) cp s
(2) rtbi
(3) yoac

重新按行拼接得到密文C= cp srtbiyoac

栅栏密码

解密环节：k=4，密文c=nacbaol ansgektily，明文？

nostalgic elbakyan

Hill密码

希尔密码（Hill Cipher）是运用基本矩阵论原理的替换密码。每个字母当作26进制数字：

A=0, B=1, C=2... 一串字母当成n维向量，跟一个n×n的矩阵相乘，再将得出的结果mod26。

其中n×n的矩阵充当加密的密钥

举例：加密密钥K=$\begin{bmatrix}11 & 8\\ 3 & 7\end{bmatrix}$，明文P=$[9 \quad 20]$

因此加密过程即为C=P*K=$[159 \quad 212]$ mod 26 = $[3 \quad 4]$

解密时需要用到解密密钥$K^{-1} = \begin{bmatrix}7 & 18\\ 23 & 11\end{bmatrix}$ $(K * K^{-1} = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$ mod 26)

P=C*K^{-1}= $[9 \quad 20]$

编码算法

编码算法不是加密算法，因为编码本身的过程没有密钥，只是用于将信息用其他形式进行表示的一种方法。

- ASCII & Unicode & UTF-8
- Hex&Bin表示
- Base64
- 摩斯密码
- 培根密码

ASCII & Unicode & UTF-8

ASCII码

在计算机中8个bit组成一个byte，所以人们早期也用8个bit来编码英文字母，最终形成了ASCII码。总体来说，ASCII码包含英文字母，数字，符号以及控制符，共128个，所以ASCII码的最高位总是0。

然而，ASCII码仅仅表示了英文字母，对于世界上其他的字符并不支持。因此后续出现了Unicode码。

ASCII & Unicode & UTF-8

Unicode码

Unicode 为世界上所有字符都分配了一个唯一的数字编号，这个编号范围从 0x000000 到 0x10FFFF (十六进制)，有 110 多万，每个字符都有一个唯一的 Unicode 编号，这个编号一般写成 16 进制，在前面加上 U+。例如：“马”的 Unicode 是U+9A6C。

Unicode 就相当于一张表，建立了字符与编号之间的联系，它是一种规定，Unicode 本身只规定了每个字符的数字编号是多少，并没有规定这个编号如何存储。

编号怎么对应到二进制表示呢？有多种方案：主要有 UTF-8，UTF-16，UTF-32。

ASCII & Unicode & UTF-8

UTF-8码

UTF-8 就是使用变长字节表示,顾名思义，就是使用的字节数可变，这个变化是根据 Unicode 编号的大小有关，编号小的使用的字节就少，编号大的使用的字节就多。使用的字节个数从 1 到 4 个不等。

UTF-8 的编码规则是：

① 对于单字节的符号，字节的第一位设为 0，后面的7位为这个符号的 Unicode 码，因此对于英文字母，UTF-8 编码和 ASCII 码是相同的。

ASCII & Unicode & UTF-8

UTF-8码

② 对于n字节的符号（n>1）,第一个字节的前 n 位都设为 1，第 n+1 位设为 0，后面字节的前两位一律设为 10，剩下的没有提及的二进制位，全部为这个符号的 Unicode 码 。

举个例子：比如说一个字符的 Unicode 编码是 130，显然按照 UTF-8 的规则一个字节是表示不了它（因为如果是一个字节的话前面的一位必须是 0），所以需要两个字节(n = 2)。根据规则，第一个字节的前 2 位都设为 1，第 3(2+1) 位设为 0，则第一个字节为：110X XXXX，后面字节的前两位一律设为 10，后面只剩下一个字节，所以后面的字节为：10XX XXXX。所以它的格式为 110XXXXX 10XXXXXX 。最后将该二进制数从右向左依次填入二进制格式的 X 中，如果还有 X 未填，则设为 0 。最终得到： 11000010 10000010

ASCII & Unicode & UTF-8

UTF-8码

“马”的 Unicode 编号是：0x9A6C，UTF-8 = ？

编号范围 （编号对应的十进制数）	二进制格式
0x00 – 0x7F (0 – 127)	0XXXXXXX
0x80 – 0x7FF (128 – 2047)	110XXXXX 10XXXXXX
0x800 – 0xFFFF (2048 – 65535)	1110XXXX 10XXXXXX 10XXXXXX
0x10000 – 0x10FFFF (65536以上)	11110XXX 10XXXXXX 10XXXXXX 10XXXXXX

1001 1010 0110 1100

11101001 10101001 10101100

Hex&Bin表示

字符在计算机实际上就是数据，以byte为单位进行存储，将byte以不同形式表示出来就有了不同的结果。例如，针对字符串“abc”，在计算机中实际就是数值分别为97，98，99的三个byte。

为了方便描述和转换，Hex表示和Bin表示是两种常用的方法。Hex即为用16进值进行表示，Bin即为用0，1的bit进行表示。

在这里，“abc”的hex表示就是616263（97=0x61），而Bin表示就是

011000010110001001100011

Base64

Base64是一种二进制到文本的编码方式。如果要更具体一点的话，可以认为它是一种将byte数组编码为字符串的方法，而且编码出的字符串只包含ASCII基础字符。

Base64，顾名思义，就是包括小写字母a-z、大写字母A-Z、数字0-9、符号“+”、“/”一共64个字符的字符集，（另加一个“=”，实际是65个字符，至于为什么还会有一个“=”，这个后面再说）。任何字符都可以转换成这个字符集中的字符，这个转换过程就叫做base64编码。

Base64

索引	对应字符	索引	对应字符	索引	对应字符	索引	对应字符
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w		
15	P	32	g	49	x		
16	Q	33	h	50	y		

Base64索引表

直接上个例子，针对字符串abc，

首先转换成Bin表示，得到

011000010110001001100011，由

于base64有64个字符，即2^6，因

此把Bin表示分成6个一组，得到

011000 010110 001001 100011

分别为24，22，9，35，对应到

base64表就是YWJj

Base64

仔细观察可以发现，Base64实际上就是用64个可见字符来表示0b000000-0b111111范围之间的数值。由于原先字符均为8个bit（ASCII码），因此需要8/6个Base64字符，也就是说，3个ASCII码会变成4个Base64字符（也就是刚刚的那个例子）。

如果原先的ASCII码的个数不是三的倍数怎么办？

（1）余1，此时需要在后面补0，例如字符a，即01100001，需要补4个0，变成YQ，最后再跟上==，对齐到4个字符（同时也表明原先只有一个字符），得到YQ==。

（2）余2，例如字符ab，即0110000101100010，需要补2个0，得到YWI，最后再补一个=，得到YWI=。

所以先前的=就是为了进行padding，因此不算在Base64实际用于表示的字符中。

Base64

练习：需要编码的字符串为Base，编码后的字符串为？

索引	对应字符	索引	对应字符	索引	对应字符	索引	对应字符
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w		
15	P	32	g	49	x		
16	Q	33	h	50	y		

QmFzZQ==

Base64

```python
from base64 import b64encode, b64decode, encode

plaintext = b"abcdef\x00\x0a"
print(b64encode(plaintext))

encodetext = b"QmFzZQ=="
print(b64decode(encodetext))
```

b'YWJjZGVmAAo='

b'Base'

摩斯密码

摩尔斯电码是一种早期的数码化通信形式， 它依靠一系列的 **点和划** 来传递编码信息，它的代码包括五种：

点（·）：1（读“滴”dit，时间占据1t）

划（—）：111（读“嗒”dah，时间占据3t）

字符内部的停顿（在点和划之间）：0（时间占据1t）

字符间停顿：000（时间占据3t）

单词间的停顿：0000000（时间占据7t）

摩斯密码

举例，我们现在要发送 “M O R S E(空格) C O D E”（morse code）这单词，通过查表可知，它应该是这样 —— ——— ·—·· ··· · / —·—· ——— —·· ·

对应的报文应该如下（滴 表示敲击，▢ 表示停顿）

滴滴滴▢滴滴滴▢▢▢滴滴滴▢滴滴滴▢滴滴滴▢▢▢滴▢滴滴滴▢滴▢▢▢滴▢滴▢滴

▢▢▢滴▢▢▢▢▢▢▢滴滴滴▢滴▢滴滴滴▢滴▢▢▢滴滴滴▢滴滴滴▢滴滴滴

字符	电码符号	字符	电码符号	字符	电码符号	字符	电码符号
A	· —	B	—· · ·	C	— · —·	D	— · ·
E	·	F	· · —·	G	— —·	H	· · · ·
I	· ·	J	· — — —	K	— · —	L	· —· ·
M	— —	N	—·	O	— — —	P	· — —·
Q	— —· —	R	· — ·	S	· · ·	T	—
U	· · —	V	· · · —	W	· — —	X	— · · —
Y	—· — —	Z	— — · ·				

培根密码

培根密码实际上就是一种替换密码，根据所给表一一对应转换即可加密解密。它的特殊之处在于：可以通过不明显的特征来隐藏密码信息，比如大小写、正斜体等，只要两个不同的属性，密码即可隐藏。

例如针对字符串CRYPTO，转换为aaababaaabbbaaaabbbbbaabbabbba

为了进一步隐藏信息，可以试先准备一条信息，要求长度相同，例如‘behind the mountain there are peopl’（空格不算），然后将b对应的字符变为粗体或者大写，得到

BehInD the MOUntaiN THERe aRE pEOPl

beh**i**n**d** the **mou**ntai**n** **ther**e a**re** p**eop**l

A/a	aaaaa	H/h	aabbb	O/o	abbba	V/v	babab
B/b	aaaab	I/i	abaaa	P/p	abbbb	W/w	babba
C/c	aaaba	J/j	abaab	Q/q	baaaa	X/x	babbb
D/d	aaabb	K/k	ababa	R/r	baaab	Y/y	bbaaa
E/e	aabaa	L/l	ababb	S/s	baaba	Z/z	bbaab
F/f	aabab	M/m	abbaa	T/t	baabb		
G/g	aabba	N/n	abbab	U/u	babaa		

培根密码转换表

隐写术

隐写术是一门关于信息隐藏的技巧与科学，所谓信息隐藏指的是不让除预期的接收者之外的任何人知晓信息的传递事件或者信息的内容。

一般来说，隐写的信息看起来像一些其他的东西，例如一张购物清单，一篇文章，一篇图画或者其他“伪装”（cover）的消息。

隐写的信息通常用一些传统的方法进行加密，然后用某种方法修改一个“伪装文本”（covertext），使其包含被加密过的消息，形成所谓的“隐秘文本”（stegotext）。例如，文字的大小、间距、字体，或者掩饰文本的其他特性可以被修改来包含隐藏的信息。只有接收者知道所使用的隐藏技术，才能够恢复信息，然后对其进行解密。

刚刚介绍的培根密码就用到了隐写术。

其他古典密码

事实上，古典密码种类非常多，古典密码编码方法归根结底主要有两种，即**置换和代换**。

除了先前介绍的，还有：

- 棋盘密码
- ADFGX密码
- Playfair密码
- 键盘密码
- 猪圈密码
- 跳舞的小人（福尔摩斯）
- 云影密码

……

流密码

流密码的基本思想是利用密钥k产生一个密钥流z=z0z1z2…（长度无限，明文有多长，密钥流就有多长）然后与明文进行$x=x_0x_1x_2…$操作（例如异或），得到密文。

密钥流由密钥流发生器生成，为了保证安全新，使密钥流生成器要有不可预测性，具体表现如下：

- 长周期
- 高线性复杂度
- 统计性能良好
- 足够的“混乱”
- 足够的“扩散”
- 抵抗不同形式的攻击

RC4算法

RC4算法是Ron Rivest在1987年设计的流密码，密钥长度可变，使用面向字节的操作。有分析指出，RC4算法的密码流周期完全可能大于10^100。根据初始密钥K生成密钥流Z的过程如下：

（1）设置一个初始数组S=[0,1,2,…,255]

（2）创建临时数组T，如果密钥K的长度为256位，则直接将K赋给T，否则一直重复复制K，直到填满256字节长度的T

（3）利用T打乱S，此时S变成了由0-255打乱后数组

```
/*S的初始排列*/
j = 0;
for i = 0 to 255 do
    j = (j + S[i] + T[i]) mod 256;
    Swap(S[i], S[j]);
```

RC4算法

（4）利用S产生密码流，T不再使用

```
/*流产生*/
i,j = 0;
while (true)
    i = (i + 1) mod 256;  //i 不断从0~255循环
    j = (j + S[i] ) mod 256;
    Swap (S[i], S[j]);  //每产生一个k都重新排列S
    t = (S[i] + S[j]) mod 256;
    k = S[t];  //获取S中的一个随机元素作为密码流字节
```

（5）利用生成的密码流与明文进行一一异或，得到密文

RC4算法

```python
from Crypto.Cipher import ARC4

def encrypt(data, key):
    enc = ARC4.new(key)
    res = enc.encrypt(data)
    return res

data = b'1234567890'
# key len need in [5,256]
key = b"12345"

print(encrypt(data, key))
```

b'a\x07\x0bV\x19(;W\xa2\x13'

伪随机生成器

伪随机生成器与密钥流生成器类似，目标就是生成随机的输出。介绍三种最为常见的伪随机生成器：

- 线性同余生成器（Linear congruential generator，LCG）
- 线性反馈移位寄存器（linear feedback shift register，LFSR）
- MT19937（Python Random库内部实现）

线性同余生成器（LCG）

线性同余生成器的生成算法较为简单，其递推关系式如下：

$$X_{n+1} = (aX_n + c) \mod m.$$

其中a,c,m为常数。

C语言中的伪随机生成算法实际上就是使用了LCG

```
1   unsigned long int next = 1;
2   /* rand: return pseudo-random integer on 0..32767 */
3   int rand(void)
4   {
5           next = next * 1103515245 + 12345;
6           return (unsigned int)(next / 65536) % 32768;
7   }
8   /* srand: set seed for rand() */
9   void srand(unsigned int seed)
10  {
11          next = seed;
12  }
```

线性同余生成器（LCG）

$$X_{n+1} = (aX_n + c) \mod m.$$

在未知m的情况下，已知其中四个连续的生成数可以推出m

$$x1 = a * x0 + c \bmod m$$

$$x2 = a * x1 + c \bmod m$$

$$x1 + k0 * m = a * x0 + c \qquad (1)$$

$$x2 + k1 * m = a * x1 + c \qquad (2)$$

(1)-(2):

$$(k0 - k1) * m = a * (x0 - x1) + x2 - x1$$

线性同余生成器（LCG）

$$(k0 - k1) * m = a * (x0 - x1) + x2 - x1$$

同理有：

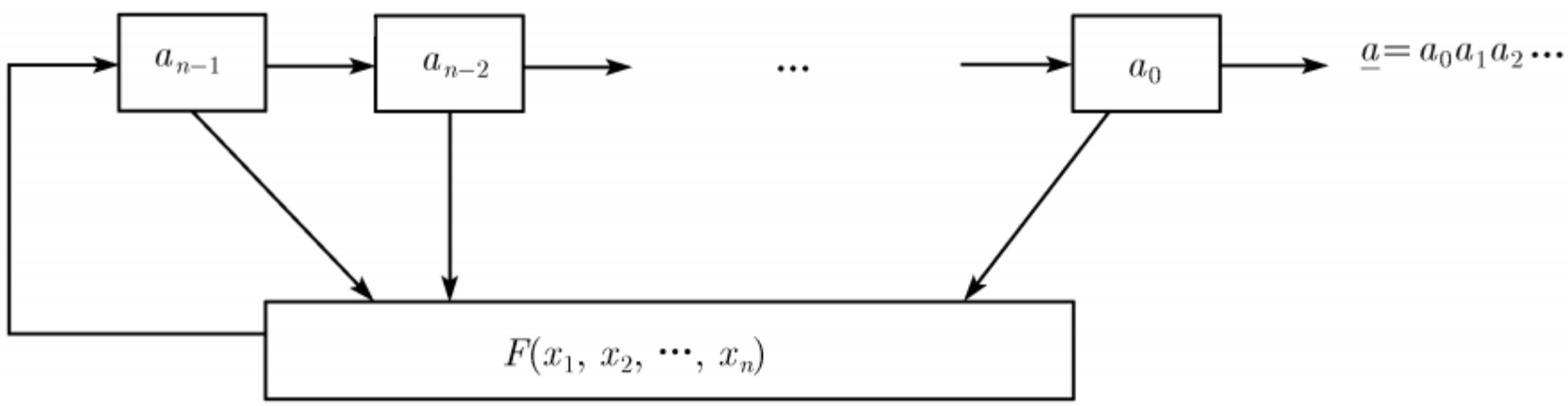
$$(k1 - k2) * m = a * (x1 - x2) + x3 - x2$$

$$\gcd(a * (x0 - x1) + x2 - x1, a * (x1 - x2) + x3 - x2) = m * k$$

线性反馈移位寄存器（LFSR）

线性反馈移位寄存器，顾名思义，存在多个寄存器保存当前状态，当需要输出时用这些寄存器可以产生新的输出，同时更新当前寄存器。

如下图所示，Q1,Q2,Q3,…,Qn是n个寄存器，然后根据这些寄存器以及输出函数反馈函数y=a1*Q1+ a2*Q2 + a3*Q3...+an*Qn mod 2每次可以输出一个bit（输出），同时根据反馈函数y=b1*Q1+ b2*Q2 + b3*Q3...+bn*Qn mod 2得到新的一个bit。此时，所有寄存器右移（最后一个被抛弃），新的bit置入第一个寄存器（更新）



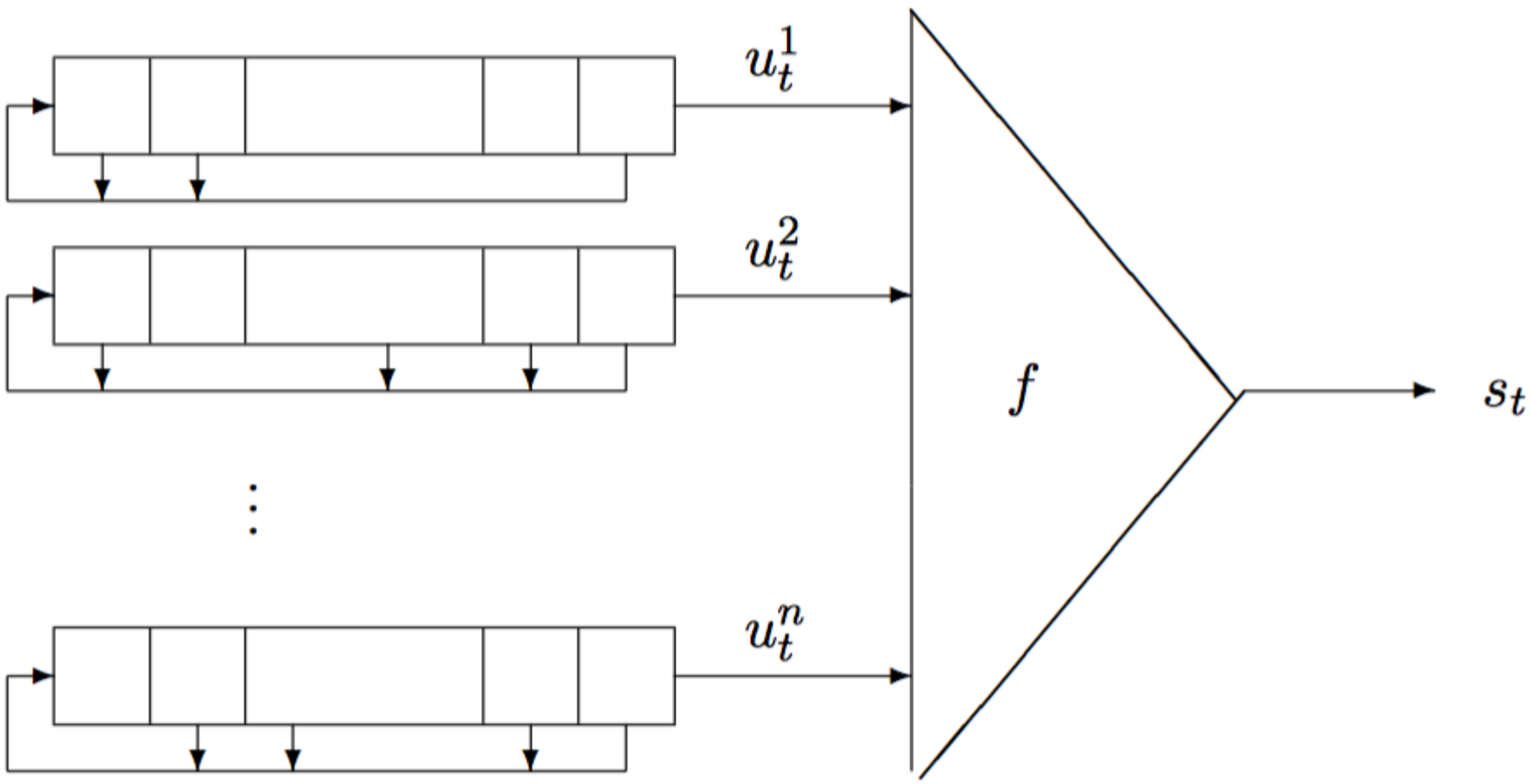
线性反馈移位寄存器（LFSR）

```
N = 100
MASK = 2**(N+1) - 1

def lfsr(state, mask):
    feedback = state & mask
    feed_bit = bin(feedback)[2:].count("1") & 1
    output_bit = state & 1
    state = (state >> 1) | (feed_bit << (N-1))
    return state, output_bit
```

非线性反馈移位寄存器（NLFSR）

非线性反馈移位寄存器由多个线性反馈寄存器组合而此，即把每个线性反馈寄存器的输出再做一次运算得到最终输出。



MT19937

MT19937是一种周期很长的的伪随机数生成算法，可以快速的产生高质量的伪随机数，

主要分为三部分。

1.利用seed初始化624个int构成的状态

2.对状态进行旋转

3.根据状态提取伪随机数

MT19937

1.利用seed初始化624个int构成的状态

```
def _int32(x):
    return int(0xFFFFFFFF & x)
```

```
class MT19937:
    # 根据seed初始化624的state
    def __init__(self, seed):
        self.mt = [0] * 624
        self.mt[0] = seed
        self.mti = 0
        for i in range(1, 624):
            self.mt[i] = _int32(1812433253 * (self.mt[i - 1] ^ self.mt[i - 1] >> 30) + i)
```

MT19937

2.对状态进行旋转

```
def twist(self):
    for i in range(0, 624):
y = _int32((self.mt[i] & 0x80000000) + (self.mt[(i + 1) % 624] & 0x7fffffff))
self.mt[i] = (y >> 1) ^ self.mt[(i + 397) % 624]
if y % 2 != 0:
    self.mt[i] = self.mt[i] ^ 0x9908b0df
```

MT19937

3.根据状态提取伪随机数

```
def extract_number(self):
        if self.mti == 0:
            self.twist()
    y = self.mt[self.mti]
    y = y ^ y >> 11
    y = y ^ y << 7 & 2636928640
    y = y ^ y << 15 & 4022730752
    y = y ^ y >> 18
    self.mti = (self.mti + 1) % 624
    return _int32(y)
```

MT19937

完整的随机数过程如下：

1. 首先调用前面的__init__函数完成初始化。
2. 然后每次需要生成随机数时调用extract_number
3. 每隔624次extract_number，会进行一次twist操作（旋转）

MT19937

逆向 extract_number

首先分析extract_number函数，可以发现输出的伪随机数是对state[i]进行了异或，位运算后的结果。倒着看

y1 = y ^ y >> 18

MT19937

逆向 **extract_number**

o = 2080737669

y = o^o>>18

decrypt

y = y^(y>>18)

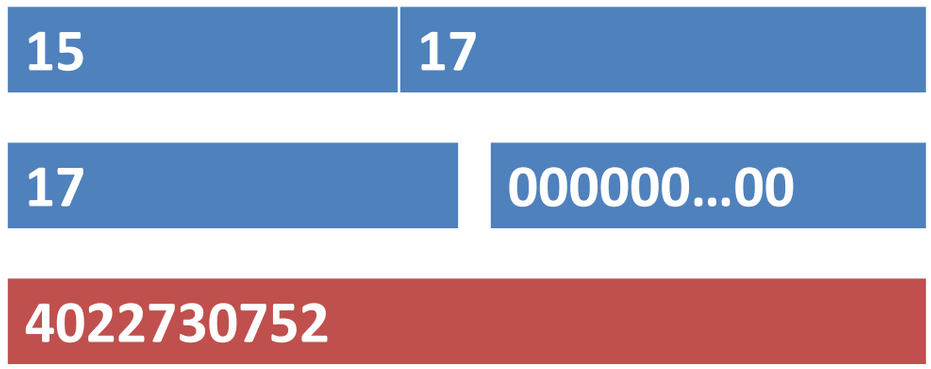
print(y==o)

MT19937

逆向 extract_number

继续分析： y = y ^ y << 15 & 4022730752

y1 = y^((y<<15)&4022730752)



y1的低15bit就是y异或0&4022730752（0）的低15bit。所以y1的低15位就是y的低15位

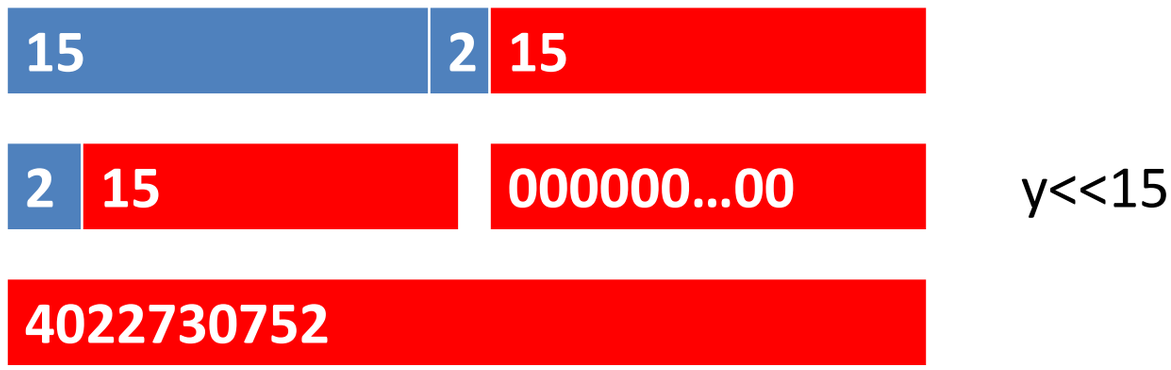
y1_low15b = y_low15b

MT19937

逆向 extract_number

继续分析： y = y ^ y << 15 & 4022730752

y1 = y^((y<<15)&4022730752)



此时可以得到第y<<15的低30位，进而恢复出y的低30bit。那y<<15的高2bit也就知道了。

再做一次这样的运算就可以恢复出高两位了。

MT19937

逆向 extract_number

继续分析： y = y ^ y << 15 & 4022730752

y1 = y^((y<<15)&4022730752)

```
o = 2080737669
y = o ^ o << 15 & 4022730752
tmp = y
for i in range(32 // 15):
    # (y<<15)&40022730752 每次可以恢复y的15位
    y = tmp ^ y << 15 & 4022730752
print(y==o)
```

MT19937

逆向 extract_number

同样的方法可以逆向出y = y ^ y << 7 & 2636928640以及y = y ^ y >> 11

MT19937

预测随机数！

```
def generate():
    fw = open("random", "w")
    for i in range(700):
        fw.write(str(random.getrandbits(32))+"n")
    fw.close()

generate()
f = open("flag.txt", "w")
key = str(random.getrandbits(32))
ciphertext = encryption(flag, key)
f.write(ciphertext)
f.close()
```

MT19937

预测随机数！

ciphertext = encryption(flag, key)

加密时的key是第701个随机数。

由于我们已知前624个随机数，通过前面对**extract_number**的逆向我们可以恢复出初始的self.mt（即state）。然后我们就能预测后面生成的随机数了！

现代密码

古典密码时期使用的技术工具普遍为手工或者机械设备，因此无法进行复杂的加密操作。

进入现代后，计算机的普及下诞生了一些较为复杂且安全的加密技术。

现代密码学的主要标志：一是美国数据加密标准 DES 的公布实施（对称密码），二是 Diffie Hellman提出的公钥密码体制（非对称密码）。

- 对称密码：加解密时使用的密钥相同
- 非对称密码：加解密时使用的密钥不相同，分为公钥，私钥。公钥是可以公开的（即他人可以知道），而私钥需要被保护（只有自己知道）。

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。尽管对称密码术有一些很好的特性，但它也存在着明显的缺陷，例如进行安全通信前需要以安全方式进行密钥交换。这一步骤，在某种情况下是可行的，但在某些情况下会非常困难，甚至无法实现。

非对称密码

非对称密码，又称公钥密码，加密和解密会使用两把不同的密钥，加密密钥(公开密钥)向公众公开，谁都可以使用，解密密钥(秘密密钥)只有解密人自己知道，非法使用者根据公开的加密密钥无法推算出解密密钥。

公钥密码体制的算法中最著名的代表是RSA系统，此外还有:背包密码、McEliece密码、Diffe_Hellman、Rabin、零知识证明、椭圆曲线、ElGamal算法等。

非对称密码体制的优点在于：首先，在多人之间进行保密信息传输所需的密钥组和数量很小；第二，密钥的发布不成问题；第三，公开密钥系统可实现数字签名。缺点：公开密钥加密比私有密钥加密在加密／解密时的速度慢。

对称密码

- DES
- AES

DES

DES是1977年美国联邦信息处理标准（FIPS）中所采用的一种对称密码（FIPS46.3）。DES一直以来被美国以及其他国家的政府和银行等广泛使用。然而，随着计算机的进步，现在DES已经能够被暴力破解，强度大不如前了。RSA公司举办过破解DES密钥的比赛（DESChallenge），我们可以看一看RSA公司官方公布的比赛结果：

1997年的DES Challenge1中用了96天破译密钥

1998年的DES ChallengeII-I中用了41天破译密钥

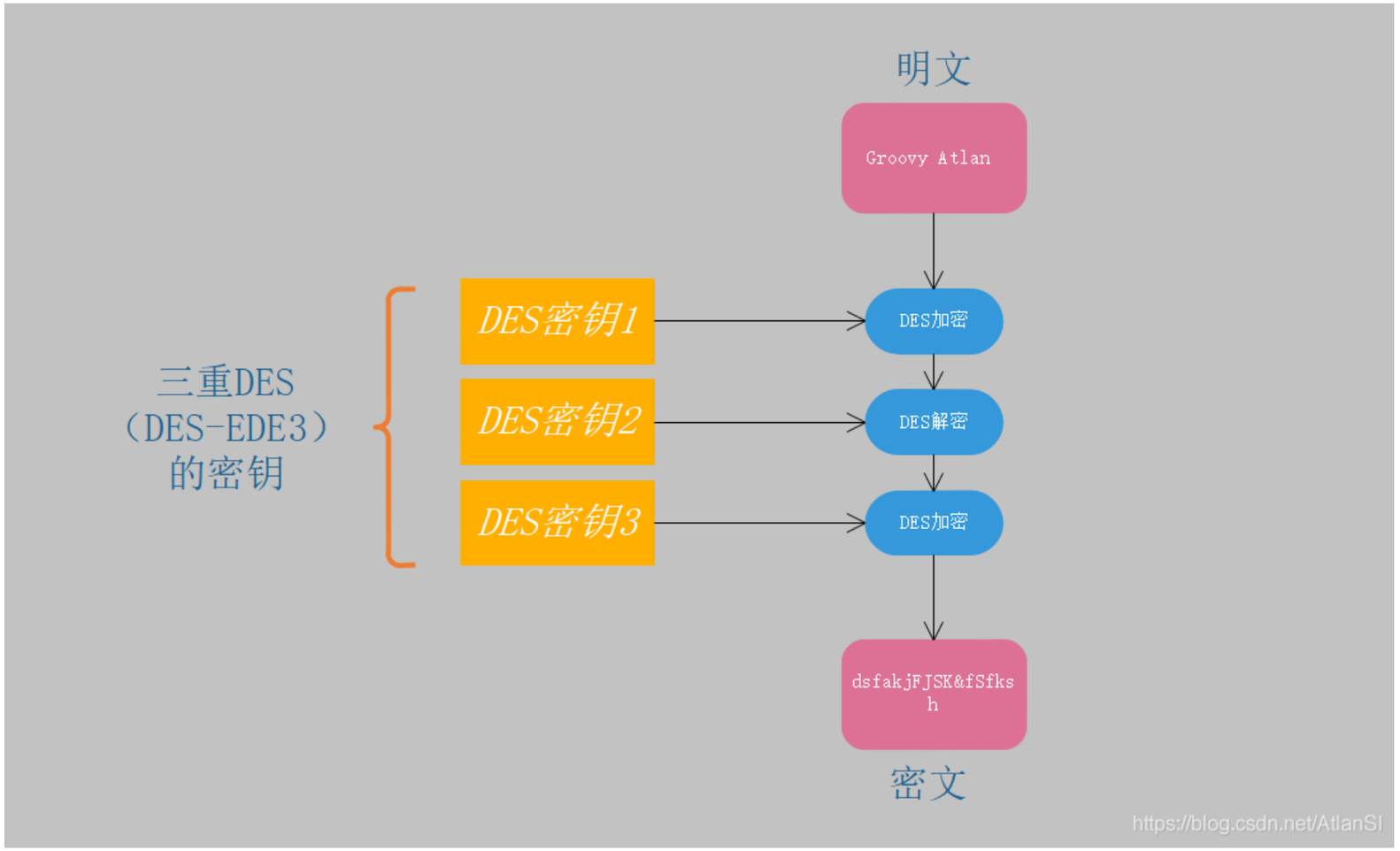
1998年的DES ChallengeII-2中用了56小时破译密钥

1999年的DES ChallengeIII中只用了22小时15分钟破译密钥

由于DES的密文可以在短时间内被破译，所以目前DES并不是一种安全的加密算法。

3DES

由于DES的不安全性，后来诞生了3DES，顾名思义，就是利用三次DES的加解密操作达到安全性。3DES的密钥长度是112（56*2）,实际上就是两个DES的密钥长度，加密时采用的操作如下：



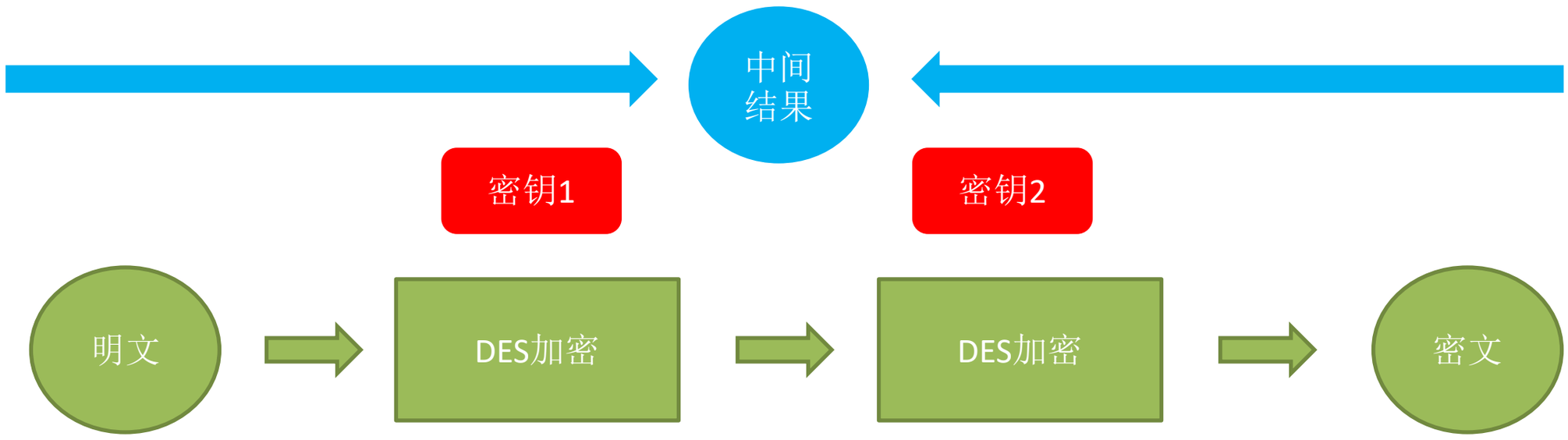
3DES

为什么不选择用DES直接加密两次（加密两次显然比现在的策略加密的速度会更快）。

原因是不安全。考虑中间相遇攻击：

枚举密钥1，将明文加密得到的结果存到数据库dataset中。

枚举密钥2，将密文解密得到的结果在数据库dataset中进行查找，如果找到说明破解了密钥1和密钥2。此时的爆破空间为2^56 * 2，而不是密钥长度的2^112



DES

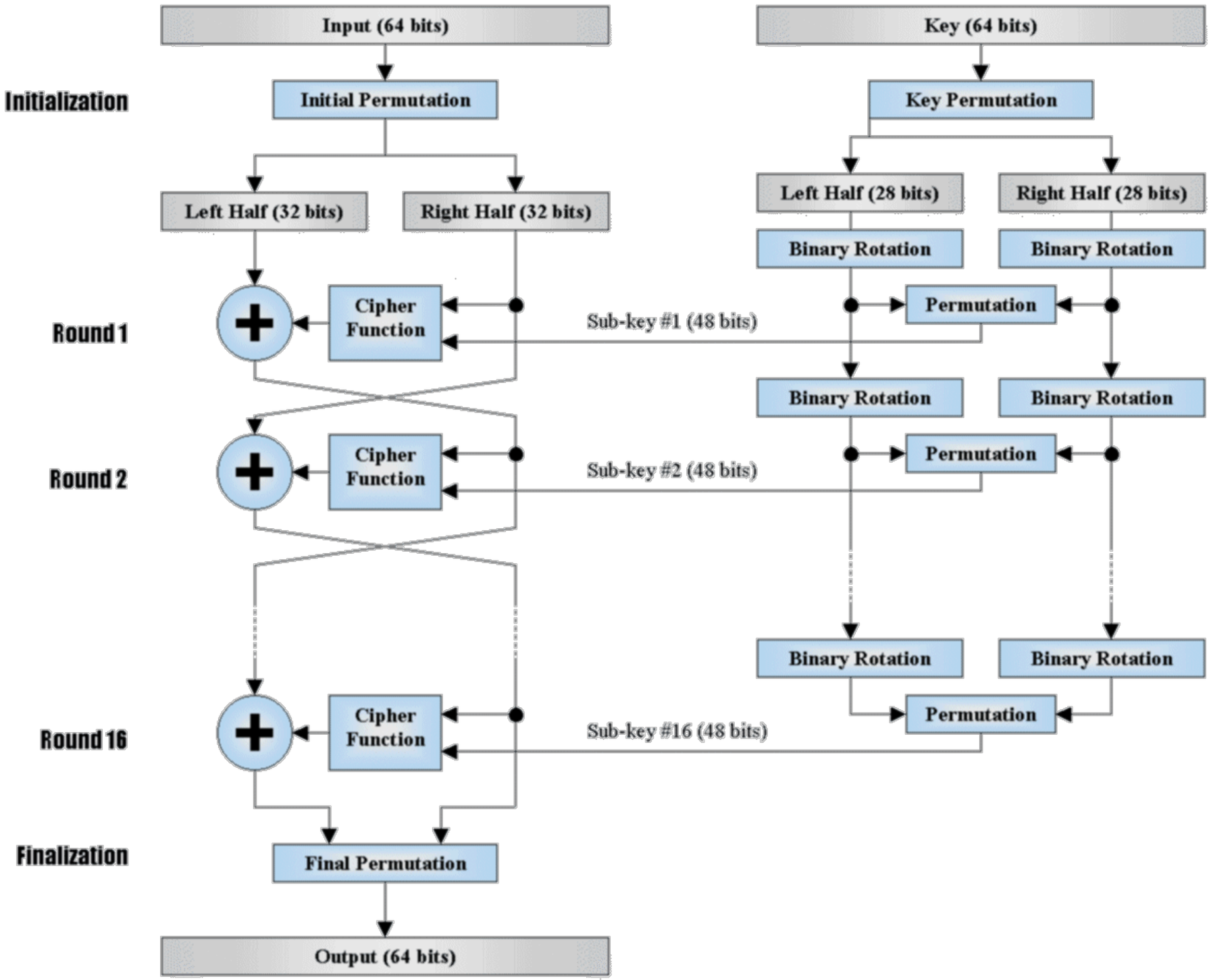
基本信息

- 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


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

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：

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

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

AES

输入：128 位

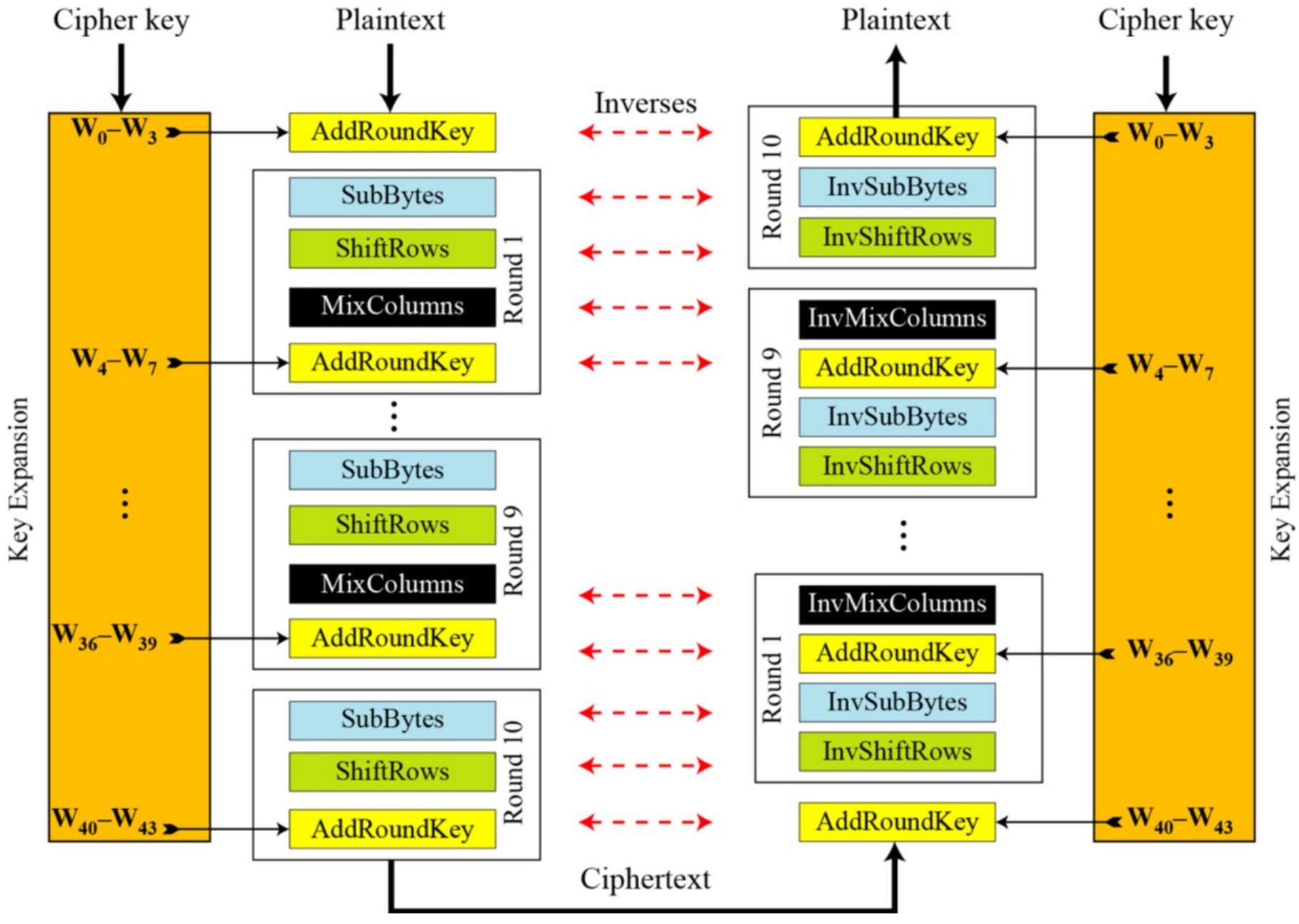
输出：128 位

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



非对称密码

- RSA
- ECC

RSA前置知识

欧拉函数

任意给定正整数n，请问在小于等于n的正整数之中，有多少个与n构成互质关系？

计算这个值的方法就叫做欧拉函数，以ϕ(n)表示。

- n=1, ϕ(n)=1
- n为素数, ϕ(n)=n-1
- n=p*q, ϕ(n)=ϕ(p)*ϕ(q)=(p-1)*(q-1) (p,q为不相等的两个素数)
- n=p^k,从定义上考虑，与n不互质的有p,2p,3p,...p^(k-1)*p,数一下共p^k-1个，那么剩下的就是互质的，因此ϕ(n)=$n*(1-\frac{1}{p})$

RSA前置知识

推广：

$$let\ n = p_1^{k1} * p_2^{k2} * ... * p_r^{kr}$$

$$\varphi(n) = \varphi(p_1^{k1}) * \varphi(p_2^{k2}) * ... * \varphi(p_r^{kr})$$

$$\varphi(n) = p_1^{k1} * (1 - \frac{1}{p_1}) * p_2^{k2} * (1 - \frac{1}{p_2}) * ... * p_r^{kr} * (1 - \frac{1}{p_r})$$

$$\varphi(n) = p_1^{k1} * p_2^{k2} * ... * p_r^{kr} * (1 - \frac{1}{p_1}) * (1 - \frac{1}{p_2}) * ... * (1 - \frac{1}{p_r})$$

$$\varphi(n) = n * (1 - \frac{1}{p_1}) * (1 - \frac{1}{p_2}) * ... * (1 - \frac{1}{p_r})$$

RSA前置知识

欧拉定理

$$a^{\varphi(n)} = 1 \, mod \, n, \quad gcd(a, n) = 1$$

RSA

RSA 算法的可靠性由极大整数因数分解的难度决定。换言之，对一极大整数做因数分解愈困难，RSA 算法愈可靠。

公钥和私钥生成过程：

- 随机选择两个不同大质数 p 和 q，计算 N=p*q
- 根据欧拉函数，求得 $\phi(N)=\phi(p)\phi(q)=(p-1)(q-1)$
- 选择一个小于 $\phi(N)$ 的整数 e，使 e 和 $\phi(N)$ 互质。并求得 e 关于 $\phi(N)$ 的模反元素，命名为 d，有 $ed\equiv1\ (mod\phi(N))$
- 将 p 和 q 的记录销毁

此时，(N,e) 是公钥，(N,d) 是私钥。

目前使用最多的是1024位和2048位（n）的RSA算法。

RSA

- **加密**

首先需要将消息以一个双方约定好的格式转化为一个小于N的整数m。如果消息太长，

可以将消息分为几段

例如，字符串ABC，先用Hex表示就是414243，16进制值即为4276803

$$m^e \equiv c \ mod \ n$$

- **解密**

$$\mathrm{c}^d \equiv m \ mod \ n$$

然后再将m转换为消息。

RSA

正确性证明

$$
\begin{aligned}
c^d =& (m^e)^d \\
=& m^{e*d} \ mod \ n \\
=& m^{1+k*\varphi(n)} \ mod \ n \\
=& m * m^{k*\varphi(n)} \ mod \ n \\
=& m * (m^{\varphi(n)})^k \ mod \ n \quad ① \\
=& m \ mod \ n \quad ②
\end{aligned}
$$

RSA

正确性证明

$$
\begin{aligned}
c^d &= (m^e)^d \\
&= m^{e*d} \ mod \ n \\
&= m^{1+k*\varphi(n)} \ mod \ n \\
&= m * m^{k*\varphi(n)} \ mod \ n \\
&= m * (m^{\varphi(n)})^k \ mod \ n \quad ① \\
&= m \ mod \ n \quad ②
\end{aligned}
$$

其中，①⇒②需要额外证明。

如果gcd(m,n)=1，直接由欧拉定理即可证明

RSA

正确性证明

如果gcd(m,n)!=1

$$\because n = p * q,\ gcd(m,n) \neq 1$$

$$\therefore m = sp \text{ 或} m = rq$$

不妨设m=sp

由欧拉定理, $m^{\phi(q)} = 1\ mod\ q$

$$\therefore m^{q-1} = 1\ mod\ q$$

$$\therefore m^{k*(p-1)*(q-1)} = 1\ mod\ q$$

$$\therefore m^{k*\phi(n)} = 1\ mod\ q$$

$$\therefore m^{k*\phi(n)} = tq + 1$$

$$\therefore m^{k*\phi(n)} * m = (tq+1) * m$$

$$\because m = sp$$

$$\therefore m^{k*\phi(n)+1} = tqm + m = tq * sp + m = tsn + m$$

$$\therefore m^{k*\phi(n)+1}\ mod\ n = m\ mod\ n$$

ECC

与传统的基于大质数因子分解困难性的加密方法不同，ECC 依赖于解决椭圆曲线离散对数问题的困难性。它的优势主要在于相对于其它方法，它可以在使用较短密钥长度的同时保持相同的密码强度。

作业

- 基础：完成对gcd以及inverse的求解算法，编程语言不限，但只能使用标准库。
- 挑战：完成一道LFSR的题目，要求解密出明文，编程语言不限，可以使用任意库。