密码学

Cryptography

极夜酱

目录

Chapter 1 古典密码学

1.1 密码学

1.1.1 密码学（Cryptography）

密码学是一种用来混淆的技术，它希望将正常的、可识别的信息转变为无法识别的信息。

密码学算法可以将明文（plaintext）加密（encrypt）为密文（ciphertext），也可将密文解密（decrypt）为明文。因此密码编码者（cryptographer）需要设计更加安全的加密算法，而破译者（cryptanalyst）的目的就是破解这些加密算法。

算法的安全性取决于破解的难度。例如一个旋转密码锁有 40 个数字，如果密码是一个 3 元组（如 $5R, 12L, 7R$），那么可能的密码组合有 $40^3 = 64000$ 种。假设尝试一种组合需要花费 10 秒，一共需要大约 178 小时才能尝试完。

图 1.1: 旋转密码锁

如果密码的组合是一个 4 元组，那么可能的密码组合有 $40^4 = 2560000$ 种。假设尝试一种组合需要花费 13 秒，一共需要大约 9244 小时才能尝试完。

大部分商业的加密算法是公开的，而军用的加密算法是保密的。

1.1.2 加密算法

一种最简单的加密算法就是换位密码/转置密码（Transposition Cipher），它将明文中的字符以某种规则移动位置，形成密文。

例如栅栏密码 (Rail Fence Cipher)，将明文的字符在行与行之间交替，形成密文。

比如有一段明文“meet me after the party”，将其分成两行交错排列，得到：

m e m a t r h p r y
 e t e f e t e a t

这种加密方式仅仅只是打乱了顺序，并没有改变明文中的字符和出现频率，因此很容易破解。

1.1.3 安全性

加密算法的安全性分为：

1. 无条件安全性（unconditionally secure）：即使密码分析者拥有无限的计算资源和密文，都没有足够的信息恢复出明文。事实上，只有一次一密乱码本，才是不可破的，对于实际应用的密码算法都是可破的。在实际中，无条件安全的系统是不存在的。

2. 计算安全性（Computationally secure）：当破解所花费的代价远超于被加密信息的生命周期和本身的价值时，破解就失去了意义。

1.2 凯撒密码

1.2.1 凯撒密码（Caesar Cipher）

凯撒密码是一种最简单的加密算法，它是由罗马帝国的凯撒（Julius Caesar）发明的。

凯撒加密将明文中的字母用另一个字母来替换，替换的规则是将字母向后移动 3 位。例如，明文中的 A 将被替换为 D，明文中的 B 将被替换为 E，依此类推。

- 明文：meet me after the party
- 密文：phhw ph diwhu wkh sduwb

破解的方法也很容易，只需要将密文的字母向前移动 3 位即可。

改进后的凯撒加密不再采用固定移动 3 位的策略，而是可以移动任意位数。对于英文字母的加密而言，一个字母可能被替换为另外的 25 个字母之一。通过暴力枚举每个移位，还是可以破解凯撒加密。

凯撒密码（加密）

```python
def encrypt(plaintext, shift=3):
    shift %= 26
    ciphertext = ""

    for c in plaintext:
        if not c.isalpha():
            ciphertext += c
            continue
        if c.isupper():
            ciphertext += chr((ord(c) + shift - 65) % 26 + 65)
        else:
            ciphertext += chr((ord(c) + shift - 97) % 26 + 97)

    return ciphertext
```

凯撒密码（解密）

```python
def decrypt(ciphertext, shift=3):
    plaintext = ""

    for c in ciphertext:
        if not c.isalpha():
            plaintext += c
            continue
        if c.isupper():
            plaintext += chr((ord(c) - shift - 65) % 26 + 65)
        else:
            plaintext += chr((ord(c) - shift - 97) % 26 + 97)

    return plaintext
```

1.3 单表密码

1.3.1 单表密码（Monoalphabetic Cipher）

单表密码并不像凯撒密码一样仅仅只是将字母后移，而是随机地将一个字母替换为另一个字母。例如字母的替换规则如下：

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

- 明文：if we wish to replace letters
- 密文：WI RF RWAJ UH YFTSDVF SFUUFYA

单表密码一共有 $25! = 1.55 \times 10^{25}$ 种可能的替换规则，但是它并不安全。由于人类语言的特性，不同的字母的使用频率是不同的。

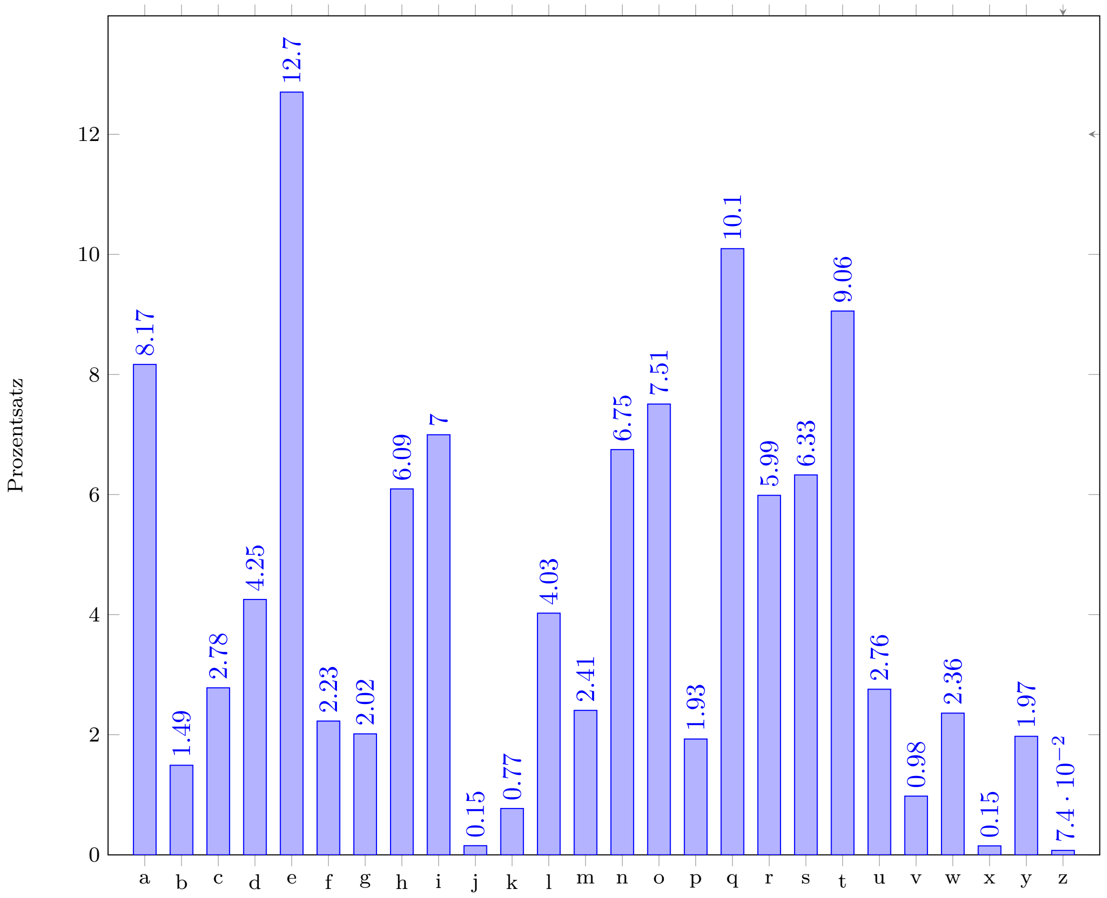

图 1.2: 字母使用频率

例如有这么一段密文:

UZ QSO VUOHXMOPV GPOZPEVSG ZWSZ OPFPESX UDBMETSX
AIZ VUEPHZ HMDZSHZO WSFP APPD TSVP QUZW YMXUZUHSX
EPYEPOPDZSZUFPO MB ZWP FUPZ HMDJ UD TMOHMQ

字母出现频率

```python
import re

def get_frequency(text):
    frequency = {}
    for c in text:
        if c in frequency:
            frequency[c] += 1
        else:
            frequency[c] = 1
    return frequency


def main():
    with open("ciphertext.txt") as file:
        text = file.readlines()
        text = "".join(text)
        # remove all non-alphabetic characters
        text = re.sub("[ \n\r]", '', text)

        print(get_frequency(text))


if __name__ == '__main__':
    main()
```

这段密文的字母使用频率如下:

{’U’: 10, ’Z’: 14, ’Q’: 3, ’S’: 10, ’O’: 9, ’V’: 5, ’H’: 7, ’X’: 5, ’M’: 8, ’P’: 16, ’G’: 2, 'E': 6, 'W': 4, 'F': 4, 'D': 6, 'B': 2, 'T': 3, 'A': 2, 'I': 1, 'Y': 2, 'J': 1}

其中 P 和 Z 的出现次数最多，分别为 16 和 14 次。可以猜测 P 和 Z 是对应明文中最常用的字母，因此 P 和 Z 非常有可能是 e 和 t。

Ut QSO VUOHXMOeV GeOteEVSG tWSt OeFeESX UDBMETSX
AIt VUEeHt HMDtSHtO WSFe AeeD TSVe QUtW YMXUtUHSX
EeYEeOeDtStUFeO MB tWe FUet HMDJ UD TMOHMQ

通过英语中单词的组合，可以猜测 S 对应 a、W 对应 h，因此 ZWSZ 对应 that、ZWP 对应 the。

Ut QaO VUOHXMOeV GeOteEVaG that OeFeEaX UDBMETaX
AIt VUEeHt HMDtaHtO haFe AeeD TaVe QUth YMXUtUHaX
EeYEeOeDtatUFeO MB the FUet HMDJ UD TMOHMQ

依次类推，可以破解整个密文：

it was disclosed yesterday that several informal
but direct contacts have been made with political
representatives of the viet cong in moscow

1.4 维吉尼亚密码

1.4.1 维吉尼亚密码（Vigenere Cipher）

维吉尼亚密码是在凯撒密码的基础上产生的一种加密方法，它将凯撒密码的全部 25 种位移排序为一张表，与原字母序列共同组成 26 行 26 列的密码表。

	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**
A	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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

表 1.1: 维吉尼亚密码表

除了密码表，还必须有一个密钥（key）。密钥由字母组成，最少一个字母，最多可与明文字母数相等。如果密钥只有 1 个字母，相当于凯撒密码。

1.4.2 加密

例如：

- 明文：I Love You
- 密钥：OK

首先，如果密钥长度小于明文长度（非字母忽略），就重复拼接密钥，使其与明文长度相等。

- 明文：I Love You
- 密钥：OKOKOKOK

接着根据密码表进行加密。明文第一个字母是 I，密钥的第一个字母是 O，因此在查找密码表的 I 行 O 列，得到加密后的字母 W。同理，第二个字母为位于 L 行 K 列的字母 V。

依次类推，得到密文：W VCFS ICE。

1.4.3 解密

例如：

- 密文：PWZRNZBZEANQKBUHNLNB
- 密钥：WIND

首先把密钥重复拼接，直到和密文长度相同。

- 密文：PWZRNZBZEANQKBUHNLNB
- 密钥：WINDWINDWINDWINDWIND

根据密文的第一个字母 P，在密码表中找到 P 行，在 P 行找出值为 W 的列，沿着该列向上找到该列为 T，因此 P 解密后可以得到 T。

根据相同的方法，可以得到明文：TOMORROWISANOTHERDAY。

维吉尼亚密码

```python
import re

def generate_key(plaintext, key):
    n = len(plaintext) - len(key)
    for i in range(n):
        key += key[i % len(key)]
    return key


def encrypt(plaintext, key):
    # remove all non-alphabetic characters
    plaintext = re.sub("[ \n\r]", '', plaintext).upper()
    key = generate_key(plaintext, key)

    ciphertext = ""
    for i in range(len(plaintext)):
        ciphertext += chr((ord(plaintext[i]) + ord(key[i])) % 26 + 65)
    return ciphertext


def decrypt(ciphertext, key):
    # remove all non-alphabetic characters
    ciphertext = re.sub("[ \n\r]", '', ciphertext).upper()
    key = generate_key(ciphertext, key)

    plaintext = ""
    for i in range(len(ciphertext)):
        plaintext += chr((ord(ciphertext[i]) - ord(key[i])) % 26 + 65)
    return plaintext
```

1.5 一次性密码本

1.5.1 一次性密码本（One-Time Pad）

一次性密码本算法所用的密钥是一次性的，因此不会出现因为密钥泄露导致之前的加密内容被破解的情况。即使密钥被泄露了，也只会影响一次通信过程。

计算机在传输数据时，会将文字转换为对应的二进制编码。通过随机生成一个与明文的二进制编码长度相同的密钥，一次性密码本将明文和密钥进行异或（XOR）运算，得到密文。

由于 XOR 运算的特性，即比特位相同为 0，不同为 1：

- 0 XOR 0 = 0
- 0 XOR 1 = 1
- 1 XOR 0 = 1
- 1 XOR 1 = 0

同时另一个 XOR 的特性就是可逆性，即 A XOR B = C，则 C XOR B = A。那么通过将密文和密钥再次异或操作就可以得到原文。

1.5.2 优势

虽然一次性密码本非常简单，但是一次性密码本是无法破译的，这个破译并不是指现有的计算能力不够，而是指即使拥有无穷大的计算能力也无法破译。

假如你拿到了一段 128 bits 的密文，通过遍历等长的密钥进行暴力破解，将会生成 2^{128} 个原文，即原文的所有排列组合。

在这些排列组合中，可能会出现一些有意义的文字，但是你并不能确定这些文字是否就是原文，因为在所有的排列组合中会产生大量有意义的问题。

因此破解一次性密码本算法是无意义的，就像是知道了原文的长度，然后自己构造了一个这个长度的原文。

1.5.3 缺陷

既然一次性密码本这么好，那么为什么我们在实际中很少用到呢?

- 密钥太长：密钥的长度与原文相同，如果原文很大，那么对应的密钥也很大。
- 密钥无法重用：每个密钥只用一次，既是优点也是缺点。这意味着每次都要不停地更换密钥，增加了复杂性。
- 密钥的配送：目标端如果想解密就必须拿到密钥，如果能够机密地传输密钥给目标端，那为什么不直接将原文机密地传送给目标端呢?
- 密钥的保存：每一个明文都需要保存一个同样长度的密钥。

1.6 Playfair 密码

1.6.1 Playfair 密码（Playfair Cipher）

Playfair 密码是一种使用一个关键词方格来加密字符对的加密法，在 1854 年由 Charles Wheatstone 发明。曾在一战时期被英军所使用，二战时期澳大利亚所使用。

首先选取一个英文单词作为密钥，将这个单词（去除重复字母）填入一个 5×5 的矩阵中，剩余的位置按照字母表顺序填入方格中，其中 I 和 J 视作同一个字母。

例如选取单词 PLAYFAIR 作为密钥，生成的密钥矩阵如下：

$$\begin{bmatrix} P & L & A & Y & F \\ I & R & B & C & D \\ E & G & H & K & M \\ N & O & Q & S & T \\ U & V & W & X & Z \end{bmatrix}$$

假设需要加密的明文为 HELLO THERE，将明文中的字符两两配对。当一对中出现两个相同字母时，用 x 分隔。当最后一对只剩一个字母时，也是用 x 配对。

- 明文：HELLO THERE
- 配对：HE LX LO TH ER EX

对于每一对字母，如果它们在密钥矩阵中位于同一行，将它们替换为各自右边的字母，如 HE 加密为 KG。如果它们位于同一列，将它们替换为各自下面的字母，如 LO 加密为 RV。其它情况，将它们的列下标互换，如 LX 加密为 YV。

- 密文：KG YV RV QM GI KU

1.6.2 解密

假设目前截获了一份使用 Playfair 加密的密文和对应的原文，以及一个不完整的密钥矩阵，现在需要对另一份密文进行解密。

- 明文：THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
- 密文：RMAPLHBMUAWRVDPWHLISNPXGEGIRBEVZHNEV
- 待解密密文：RMTGNTEATG
- 不完整的密钥矩阵：

$$
\begin{bmatrix}
 & & & A & T \\
 & U & & B & C \\
 & & I & & \\
N & & P & & S \\
 & W & & & Z
\end{bmatrix}
$$

解密的方法就是根据已有的一组明文和密文，推算出完整的密钥矩阵。再根据密钥矩阵，反向利用 Playfair 的加密方法进行解密。

首先将已有的明文和密文进行两两匹配：

- 明文：TH EQ UI CK BR OW NF OX JU MP SO VE RT HE LA ZY DO GX
- 密文：RM AP LH BM UA WR VD PW HL IS NP XG EG IR BE VZ HN EV

可以发现在这些配对中，有些配对在明文和密文中出现了相同的字母：

- 明文：TH EQ UI CK BR OW NF OX JU MP SO VE RT HE LA ZY DO GX
- 密文：RM AP LH BM UA WR VD PW HL IS NP XG EG IR BE VZ HN EV

这种情况只有可能发现在被加密的两个字母出现在相邻的位置上，这样被加密时，使用它们各自右边或下面的字母，就会出现重复。

根据这个信息，可以推算出 V、Y、R、O 的位置。

$$\begin{bmatrix} & R & & A & T \\ & U & & B & C \\ & & I & & \\ N & O & P & & S \\ V & W & & Y & Z \end{bmatrix}$$

由于秘钥矩阵后，后面的字母都是按照字母表的顺序排列的，所以推算出一部分缺失的字母。

$$\begin{bmatrix} & R & & A & T \\ & U & & B & C \\ & & I & & \\ N & O & P & Q & S \\ V & W & X & Y & Z \end{bmatrix}$$

再次根据已有的明文和密文，可以推算出其它字母的位置。

- 明文：TH EQ UI CK BR OW NF OX JU MP SO VE RT HE LA ZY DO GX
- 密文：RM AP LH BM UA WR VD PW HL IS NP XG EG IR BE VZ HN EV

$$\begin{bmatrix} G & R & E & A & T \\ F & U & L & B & C \\ D & H & I & K & M \\ N & O & P & Q & S \\ V & W & X & Y & Z \end{bmatrix}$$

在破解出完整的密钥矩阵后，就可以利用 Playfair 的解密方法对密文 RMTGNTEATG 解密了。

- 密文：RM TG NT EA TG
- 明文：TH AT SG RE AT

Chapter 2 数论

2.1 最大公约数 / 最小公倍数

2.1.1 最大公约数（GCD, Greatest Common Divisor）

两个整数 a 和 b 的最大公约数 $gcd(a, b)$ 为能够同时整除 a 和 b 的最大整数。

例如：

- $gcd(24, 36) = 12$
- $gcd(17, 22) = 1$
- $gcd(500, 128) = 4$

欧几里得（Euclidean）算法/辗转相除法可以用于计算最大公约数。

最大公约数

```python
def gcd(a, b):
    while b != 0:
        remainder = a % b
        a = b
        b = remainder
    return a


def euclid_gcd(a, b):
    if b == 0:
        return a
    return gcd(b, a % b)
```

2.1.2 最小公倍数（LCD, Least Common Multiple）

两个整数 a 和 b 的最小公倍数 $lcm(a, b)$ 为能够同时被 a 和 b 整除的最小整数。

例如：

- $lcm(24, 36) = 72$
- $lcm(17, 22) = 374$
- $lcm(500, 128) = 16000$

最小公倍数

```
def lcm(a, b):
    return a * b // gcd(a, b)
```

2.2 同余定理

2.2.1 模算数（Modular Arithmetic）

当 $a \in \mathbb{Z}$、$M \in \mathbb{Z}^{+}$，那么将 a 除以 m 的余数记为 $a \bmod m$。

例如：

- $17 \bmod 5 = 2$
- $2001 \bmod 101 = 82$
- -10 mod 3 = -1

2.2.2 同余定理（Congruence Theorem）

当 $a \in \mathbb{Z}$、$b \in \mathbb{Z}$、$M \in \mathbb{Z}^{+}$，如果 m 能够整除 $a - b$，那么就称 a 和 b 对模 m 同余，记作 $a \equiv b \pmod{m}$。

因此，

$$a \equiv b \pmod{m} \leftrightarrow a \bmod m \equiv b \bmod m \tag{2.1}$$

例如：

- $17 \equiv 5 \pmod{6}$
- $17 \equiv 12 \pmod{5}$
- $24 \equiv 3 \pmod{7}$

当 $a \equiv b \pmod{m}$、$c \equiv d \pmod{m}$，同余定理满足以下性质：

- $a + c \equiv b + d \pmod{m}$
- $ac \equiv bd \pmod{m}$

Exercise

因为 $7 \equiv 2 \pmod 5$、$11 \equiv 1 \pmod 5$。

(a) $7 + 11 \pmod 5 = 2 + 1 \pmod 5 = 3$

(b) $7 \cdot 11 \pmod 5 = 2 \cdot 1 \pmod 5 = 2$

Exercise

(a) $7^{10} \bmod 5 = 2^{10} \bmod 5 = 4$

(b) $7^{100} \bmod 3 = 1^{100} \bmod 5 = 1$

2.3 中国余数定理

2.3.1 中国余数定理（CRT, Chinese Remainder Theorem）

中国余数定理/孙子定理是中国古代求解一次同余式组的算法。在《孙子算经》中有一个叫“物不知数”的问题：

有物不知其数，三三数之剩二，五五数之剩三，七七数之剩二。问物几何？

也就是说，一个整数 x 除以三余二，除以五余三，除以七余二，求这个整数。

该问题可表示为：

$$x \equiv 2 \pmod 3$$
$$x \equiv 3 \pmod 5$$
$$x \equiv 2 \pmod 7$$

或者：

$$x = (2,3,2)S(3,5,7)$$

对于只有两个同余式的问题，可以通过列表的方式直接求解。

例如 $x = (2,4)S(3,5)$，需要创建一个 3 行 5 列的表格，然后以对角线的顺序，从 0 开始依次填入数字。

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

找到表格第 2 行第 3 列的数值，即 $x = 8$。

同理，用同样的方法可以算出 $(2,4)S(3,5) = 14$。

除了列表的方法外,还有一种更加通用的求解同余式组算法。例如对于 $(2,3,4)S(3,5,13)$ 这个问题，$(3,5,13)$ 中两两互素。

如果只考虑 $(4)S(13)$，那么可以得出 $x_0 = 4$。

然后再进一步考虑 $(3,4)S(5,13)$ 的情况，那么

$$\begin{aligned} x_1 &= 4 + 13m = 3 \pmod 5 \\ &= 4 + 13 \times 1 = 17\,(\text{不满足 } 3 \pmod 5) \\ &= 4 + 13 \times 2 = 30\,(\text{不满足 } 3 \pmod 5) \\ &= 4 + 13 \times 3 = 43 \end{aligned}$$

最后在目前的情况下考虑 $(2,3,4)S(3,5,13)$，那么可以得出

$$\begin{aligned} x_2 &= 43 + (5 \times 13)m = 2 \pmod 3 \\ &= 43 + 65 \times 1 = 108\,(\text{不满足 } 2 \pmod 3) \\ &= 43 + 65 \times 2 = 173 \end{aligned}$$

2.4 希尔密码

2.4.1 希尔密码（Hill Cipher）

希尔密码运用基本矩阵运算来对明文加密，它将字母 $A \sim Z$ 用 $0 \sim 25$ 表示。因此一段长度为 n 的明文可以表示成一个包含 n 个元素的明文矩阵。将这个明文矩阵与密钥矩阵相乘，得到的结果经过模 26 后就是密文。

例如密钥矩阵为：

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 11 & 9 & 8 \end{bmatrix}$$

明文为 $ABC = (0, 1, 2)$。

密文可以通过矩阵乘法得到：

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 11 & 9 & 8 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 8 \\ 17 \\ 0 \end{bmatrix} \pmod{26} = IRA$$

希尔密码的好处在于，如果明文中的一个字母发生变化，密文中所有的字母都会受到影响。

例如当明文为 $BBC = 1, 1, 2$ 时：

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 11 & 9 & 8 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 9 \\ 21 \\ 10 \end{bmatrix} \pmod{26} = JVK$$

2.4.2 解密

例如已知一个模 5 的密钥矩阵：

$$K = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 2 & 1 \\ 3 & 1 & 4 \end{bmatrix}$$

和一个待破解的密文：

$$C = \begin{bmatrix} A & F & S \\ A & S & E \\ F & A & E \end{bmatrix}$$

以及字母与数字的对应关系：

0	1	2	3	4
A	E	F	S	T

希尔密码的解密过程如下：

1. 计算密钥矩阵 K 的模逆矩阵 K^{-1}
2. 计算 $K^{-1} \times C$
3. 将结果转换为字母

利用初等行变换，可以计算 K 的模逆矩阵 K^{-1}。注意在矩阵的运算过程中，始终要保证模 5 的操作，如 $0 - 1 = 4$。

最终得到明文为 SAFE SEATS。

$$\left(\begin{array}{ccc|ccc}1&0&1&1&0&0\\1&2&1&0&1&0\\3&1&4&0&0&1\end{array}\right)\xrightarrow{R_2=R_2-R_1}\xrightarrow{R_3=R_3-3R_1}\left(\begin{array}{ccc|ccc}1&0&1&1&0&0\\0&2&0&4&1&0\\0&1&1&2&0&1\end{array}\right)$$

$$\xrightarrow{R_2=R_2-R_3}\xrightarrow{R_3=R_3-R2}\left(\begin{array}{ccc|ccc}1&0&1&1&0&0\\0&1&4&2&1&4\\0&0&2&0&4&2\end{array}\right)$$

$$\xrightarrow{R_3=R_3/2}\left(\begin{array}{ccc|ccc}1&0&1&1&0&0\\0&1&4&2&1&4\\0&0&1&0&2&1\end{array}\right)$$

$$\xrightarrow{R_1=R_1-R_3}\xrightarrow{R_2=R_2-4R3}\left(\begin{array}{ccc|ccc}1&0&0&1&3&4\\0&1&0&2&3&0\\0&0&1&0&2&1\end{array}\right)$$

$$\begin{aligned}K^{-1}\times C &= \begin{bmatrix}1&3&4\\2&3&0\\0&2&1\end{bmatrix}\begin{bmatrix}A&F&S\\A&S&E\\F&A&E\end{bmatrix}\\&=\begin{bmatrix}1&3&4\\2&3&0\\0&2&1\end{bmatrix}\begin{bmatrix}0&2&3\\0&3&1\\2&0&1\end{bmatrix}\\&=\begin{bmatrix}3&1&0\\0&3&4\\2&1&3\end{bmatrix}\\&=\begin{bmatrix}S&E&A\\A&S&T\\F&E&S\end{bmatrix}\end{aligned}$$

最终得到明文为 SAFE SEATS。

Chapter 3 现代密码学

3.1 RSA

3.1.1 RSA

RSA 是一种非对称加密（asymmetric cryptography）算法，在 1977 年由 Ron Rivest、Adi ShamirLeonard Adleman 一起提出，RSA 就是他们三人姓氏的首字母。

所谓非对称加密，是指在网络通信中双方各有一对密钥，其中公钥（public key）被公开给外界，用于加密；私钥（private key）只有自己知道，用于解密。

例如 Alice 的一组密钥为 (P_A, S_A)，Bob 的一组密钥为 (P_B, S_B)。对一段消息 M 加密和解密的操作是互逆的：

$$M = S_A(P_A(M))$$
$$M = P_A(S_A(M))$$

如果 Bob 想要给 Alice 发送消息，Bob 首先使用 Alice 的公钥 P_A 对消息 M 进行加密得到密文 C：

$$C = P_A(M)$$

Bob 将密文 C 发送给 Alice，Alice 使用自己的私钥 S_A 对密文 C 进行解密得到原文 M：

$$S_A(C) = S_A(P_A(M)) = M$$

3.1.2 公钥/私钥生成

RSA 的安全性依赖于大整数的质因数分解，也就是对于两个大素数 p 和 q 而言，计算它们的乘积 pq 很容易，但是从积 pq 分解出 p 和 q 是个公认的数学难题。

RSA 公钥和私钥生成的过程如下：

1. 随机选择两个大素数（超过 100 位），为了简化说明，这里采用较小的素数 $p = 41$、$q = 59$
2. 计算 p 和 q 的乘积 $n = pq = 2419$
3. 计算 $p - 1$ 与 $q - 1$ 的乘积 $\phi(n) = (p - 1)(q - 1) = 40 * 58 = 2320$
4. 选择一个小奇数 e，使得 $gcd(e, \phi(n)) = 1$，例如 $e = 3$
5. 计算 d，使得 $d * e \equiv 1 \pmod{\phi(n)}$。当 $d = 1547$ 时，$1547 * 3 \bmod 2320 = 1$
6. 公钥 $P = (e, n) = (3, 2419)$
7. 私钥 $S = (d, n) = (1547, 2419)$
8. 对消息 M 加密的过程为 $P(M) = M^e \pmod{n} = M^3 \pmod{2419}$
9. 对消息 M 解密的过程为 $S(M) = M^d \pmod{n} = M^{1547} \pmod{2419}$

3.1.3 加密

假设 Alice 的公钥为 $(9, 2419)$，想要将消息“PACE”发送给 Bob。

首先将消息中的英文字母转换为对应的编码：

- PA = 1500
- CE = 0204

Letter	**Code**	**Letter**	**Code**
A	00	B	01
C	02	D	03
E	04	F	05
G	06	H	07
I	08	J	09
K	10	L	11
M	12	N	13
O	14	P	15
Q	16	R	17
S	18	T	19
U	20	V	21
W	22	X	23
Y	24	Z	25

分别对 PA 和 CE 进行加密:

- $P(1500) = 1500^9 \pmod{2419} = 1655$
- $P(0204) = 204^9 \pmod{2419} = 1639$

最终形成的密文为 1655 1639。

3.2 DES

3.2.1 DES（Data Encryption Standard）

DES 是一种分组密码，它将数据分成多个 64 位的块，每个块独立加密。因此，DES 将 64 位的明文作为输入，输出 64 位的密文。

DES 首先需要对原始数据进行一次初始置换 (IP, Initial Permutation)，接着进行 16 轮迭代运算，对数据重新排列和置换，最后再对数据进行一次最终置换（FP, Final Permutation)，得到最终的密文。

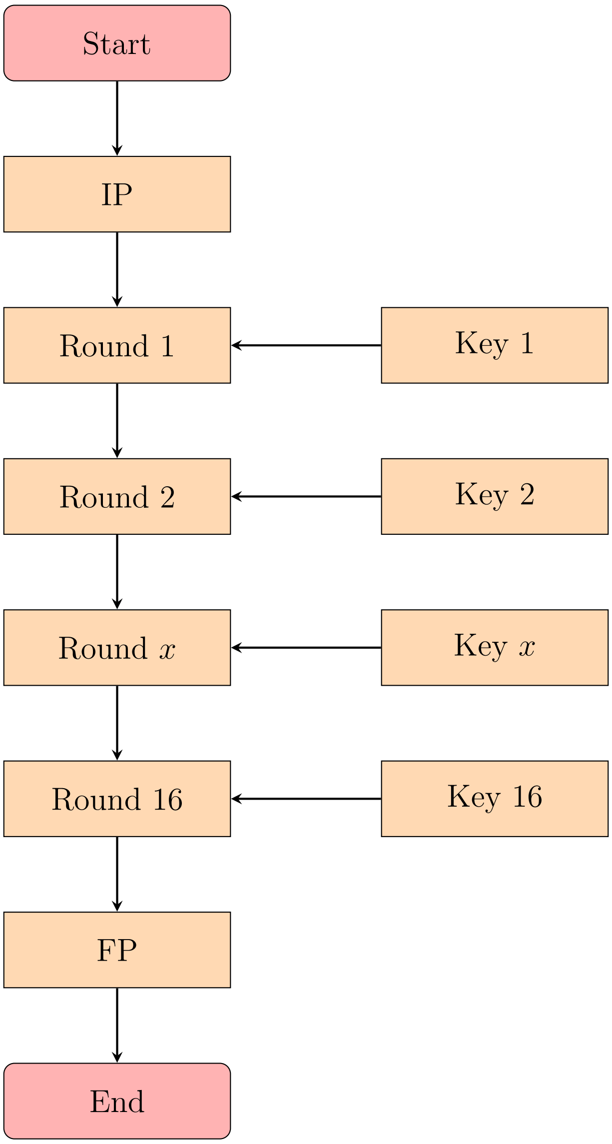

图 3.1: DES

3.2.2 密钥生成

由于在 DES 的 16 轮迭代加密的过程中，每一轮都需要使用一个不同的密钥。因此，在 DES 开始加密之前，需要先根据原始的密钥，生成 16 个不同的密钥。

原始的密钥的长度为 64 位，其中第 8、16、24、32、40、48、56、64 位为奇偶校验位。因此，忽略这 8 位奇偶校验位，然后对剩余的 56 位进行重新排列，排列的顺序参照 PC-1。

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

表 3.1: PC-1

其中 PC-1 的每个数字代表依次排列的位置，例如 57 代表将原始密钥第 57 位放到第 1 位，40 代表将原始密钥第 40 位放到第 2 位，依次类推。由于 PC-1 表中只包含 56 个位置,在进行重新选择排列的过程中,即可忽略掉 8 位奇偶校验位。

例如原始密钥为：

```
00010011 00110100
01010111 01111001
10011011 10111100
11011111 11110001
```

重新选择排列后得到一个 56 位的密钥：

```
1111000 0110011
0010101 0101111
```

```
0101010 1011001
1001111 0001111
```

接着将 56 位的密钥分为两部分，分别为左 28 位和右 28 位，分别对它们进行 16 次循环左移，循环左移的次数参照移位表。

第 i 轮	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
循环左移位数	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

表 3.2: 移位表

在每一轮循环左移后，将左右两部分重新，拼接成一个 56 位的密钥，然后对这个 56 位的密钥进行重新选择排列，最终生成每一轮的 48 位密钥，排列的顺序参照 PC-2。

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

表 3.3: PC-2

3.2.3 IP 置换

在生成 16 个密钥后，就可以对明文进行加密了。第一步需要先对明文进行 IP 置换，IP 置换的顺序参照 IP 表。

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

表 3.4: IP

3.2.4 16 轮迭代

将置换后的 64 位明文分为左 32 位和右 32 位，然后进行 16 轮迭代加密，每一轮的处理过程如下：

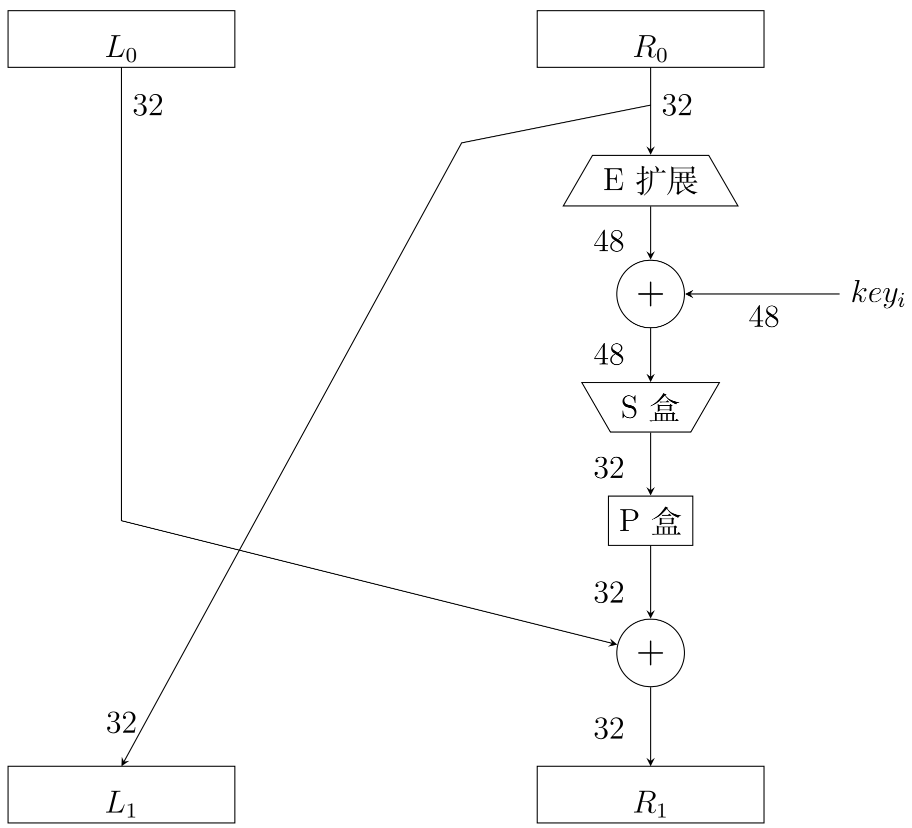

图 3.2: 16 轮迭代

3.2.5 E 扩展（Expansion Permutation）

E 扩展将 32 位的右半部分扩展为 48 位，扩展的顺序参照 E 表。

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

表 3.5: E

E 扩展的规则其实是将原来的 32 位以 4 位一组划分，然后将每一组的最后一位复制到后一组的前面，将每一组的第一位复制到前一组的后面，(其中第一组的前一组为最后一组，最后一组的下一组为第一组)。

这样就可以将 4 位一组扩展为 6 位一组,最后按顺序合并,即可得到 48 位的结果。

最后将扩展完的 48 位与之前生成的该轮密钥进行异或运算，得到 48 位的结果。

3.2.6 S 盒（S-Box Substitution）

S 盒用于将 48 位的数据压缩为 32 位。首先将 48 位的数据分为 8 组，每组 6 位，每一组都会根据对应的 S 盒（共 8 个，$S_1 \sim S_8$）进行压缩。因此，每个 S 盒都需要将 6 位输入压缩为 4 位输出。

每个 S 盒都由 4 行 16 列组成:

	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

表 3.6: S_1

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

表 3.7: S_2

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

表 3.8: S_3

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

表 3.9: S_4

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

表 3.10: S_5

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

表 3.11: S_6

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

表 3.12: S_7

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

表 3.13: S_8