《网络空间安全概论》实验报告

一、实验目的

掌握频度分析法原理和 Feistel 加解密原理.

二、实验项目内容

1. 使用频度分析法解密以下文本，并给出替换表：

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

2. 编程实现 Feistel 加密解密以下文本：

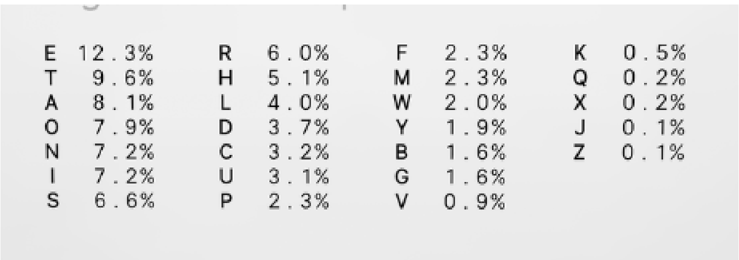
CQUINFORMATIONSECURITYEXP

三、实验设计

1. 频度分析法解密文本

1.1 实验原理

对于任何一种书面语言而言，不同的字母或字母组合出现的频率各不相同。如果以这种语言书写足够长的文本，都呈现出大致相同的特征字母分布规律，如下表所示：



E	12.3%	R	6.0%	F	2.3%	K	0.5%
T	9.6%	H	5.1%	M	2.3%	Q	0.2%
A	8.1%	L	4.0%	W	2.0%	X	0.2%
O	7.9%	D	3.7%	Y	1.9%	J	0.1%
N	7.2%	C	3.2%	B	1.6%	Z	0.1%
I	7.2%	U	3.1%	G	1.6%		
S	6.6%	P	2.3%	V	0.9%		

在上表中，不少字母出现的概率近乎相等，但也有极少数字母出现的概率有较大差异。为了分析方便密文信息，常将英文字母表按字母出现的概率大小分类，分类情况如下：

极高频 E

次高频 T A O I N S H R

中等频 D L

低频 C U M W F G Y P B

甚低频 V K J X Q Z

语言的单字母统计特性没有反映出英文双字母和多字母的特征，在双字母中统

计出概率最大的 30 对字母按概率大小排列为：

th he in er an re ed on es st en at to nt ha nd ou ea ng as or ti is et it ar te se hi of

类似的，按 Beker 在 1982 年统计的结果(样本总数 100 360)得到概率最大的 20 组三字母按概率大小排列为：
the ing and her ere ent tha nth was eth for dth hat she ionhis sth ers ver

特别地，the 出现的频率几乎为 ing 的 3 倍。

1.2 过程设计

对于频度分析法实验，采用 python(3.8.5)语言进行编写，采用 ipynb 环境运行，方便逐步调试。

首先将加密的文本放入 txt 文件中，然后读取采用 count_frequency()函数统计密文中每个字母出现的频率存入字典序列(频率保留 4 位小数)，按频率由大到小进行排序后，输出出现的字母的频率。随后结合实验原理中的词频统计进行逐一尝试，使用 replace_char()函数输出替换后的文本。为了便于观察，每一步用小写字母进行替换。最后，输出字母替换表和明文。

2.Feistel 加密解密文本

2.1 实验原理

Feistel 的加密解密过程如下：

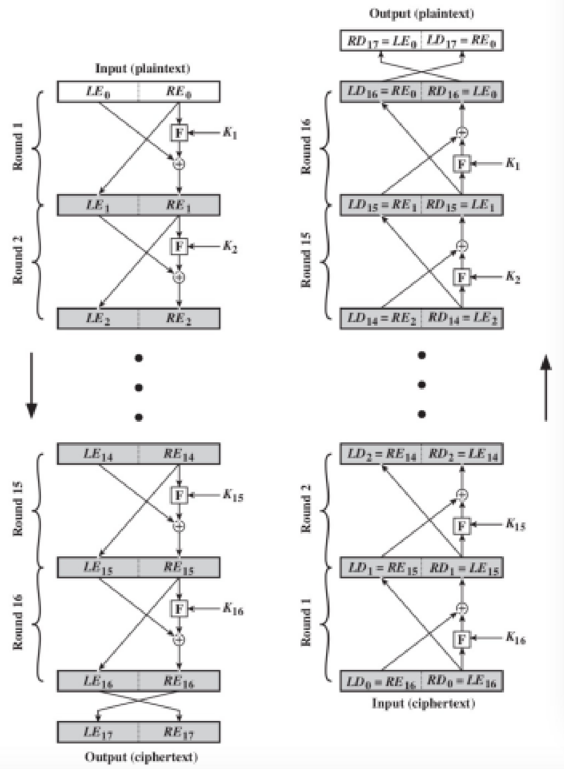


Figure 3.3 Feistel Encryption and Decryption (16 rounds)

2.2 过程设计

对于 Feistel 加密解密，使用 C++语言进行编写实现，具体设计如下：

• 迭代轮数设置为 16 轮，分组长度设置为每组 64 位，因在本次实验中，待加密文本转化为二进制后长度为 200，因此采取的措施是将其用 0bit 进行填充至 64 的倍数，即 256 位后再进行加密解密操作。

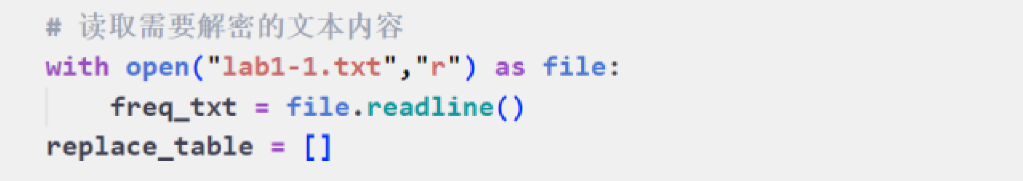
• 密钥 K：长度设置为经典使用的 128 位，采用 C++的 std::random_device 生成随机数，因此共有 16×128＝2048 位密钥。因每轮需进行异或操作，且分组为 64 位(左右 LR 各占 32 位)，因此对 2048 位密钥先进行预处理，按 4 位一组组内进行异或运算，最后可得 512 位的密钥，刚好为每轮提供 32 位的密钥。

• 为了简化设计，F 函数设置为直接异或运算。

四、实验过程或算法

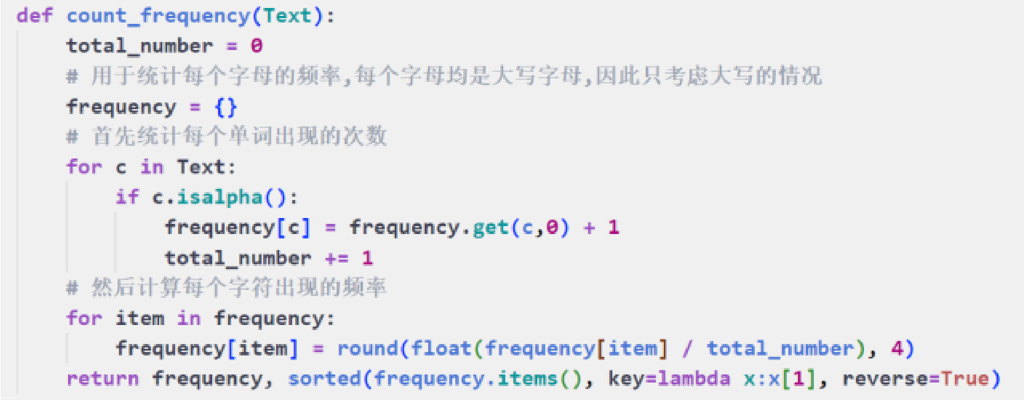
1. 频度分析法解密文本

• 首先，读取待解密的文本，设置替换表 replace_table：



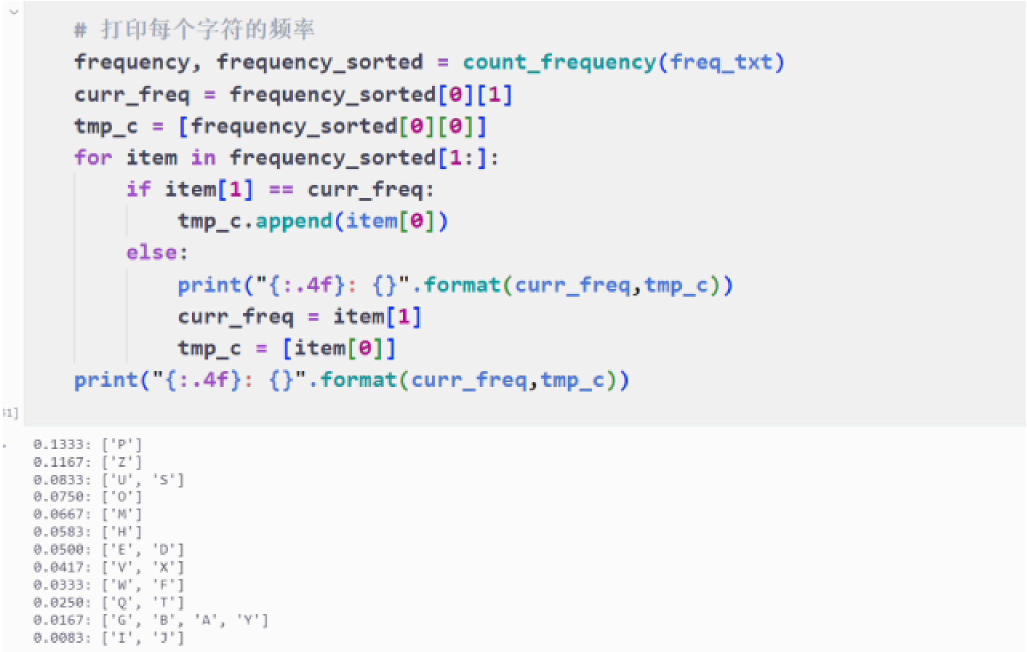
```python
# 读取需要解密的文本内容
with open("lab1-1.txt","r") as file:
    freq_txt = file.readline()
replace_table = []
```

• 然后，构造函数，统计每个字母的词频并按频率由大到小进行排列。



```python
def count_frequency(Text):
    total_number = 0
    # 用于统计每个字母的频率,每个字母均是大写字母,因此只考虑大写的情况
    frequency = {}
    # 首先统计每个单词出现的次数
    for c in Text:
        if c.isalpha():
            frequency[c] = frequency.get(c,0) + 1
            total_number += 1
    # 然后计算每个字符出现的频率
    for item in frequency:
        frequency[item] = round(float(frequency[item] / total_number), 4)
    return frequency, sorted(frequency.items(), key=lambda x:x[1], reverse=True)
```

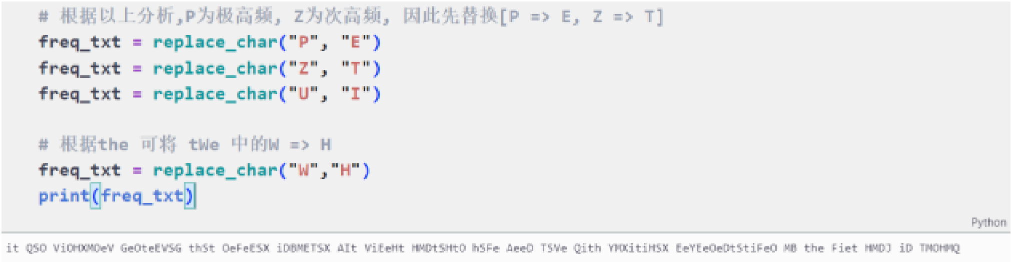
• 打印每个字母的频率，结果如下。可以看到，密文中字母 P 的频率最高，Z 的频率其次。



```python
# 打印每个字符的频率
frequency, frequency_sorted = count_frequency(freq_txt)
curr_freq = frequency_sorted[0][1]
tmp_c = [frequency_sorted[0][0]]
for item in frequency_sorted[1:]:
    if item[1] == curr_freq:
        tmp_c.append(item[0])
    else:
        print("{:.4f}: {}".format(curr_freq,tmp_c))
        curr_freq = item[1]
        tmp_c = [item[0]]
print("{:.4f}: {}".format(curr_freq,tmp_c))
```

```
0.1333: ['P']
0.1167: ['Z']
0.0833: ['U', 'S']
0.0750: ['O']
0.0667: ['M']
0.0583: ['H']
0.0500: ['E', 'D']
0.0417: ['V', 'X']
0.0333: ['W', 'F']
0.0250: ['Q', 'T']
0.0167: ['G', 'B', 'A', 'Y']
0.0083: ['I', 'J']
```

- 因此首先将 P =>E（P 替换为 E，=>为替换，下同），Z => T，U => I.并观察根据 the 可将 tWe 中的 W => H，得到如下结果：

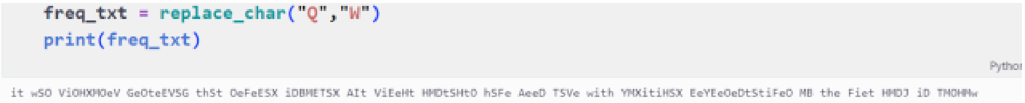


```python
# 根据以上分析,P为极高频, Z为次高频, 因此先替换[P => E, Z => T]
freq_txt = replace_char("P", "E")
freq_txt = replace_char("Z", "T")
freq_txt = replace_char("U", "I")

# 根据the 可将 tWe 中的W => H
freq_txt = replace_char("W","H")
print(freq_txt)
```

```
it QSO ViOHXMOeV GeOteEVSG thSt OeFeESX iDBMETSX AIt ViEeHt HMDtSHtO hSFe AeeD TSVe Qith YMXitiHSX EeYEeOeDtStiFeO MB the Fiet HMDJ iD TMOHMQ
```

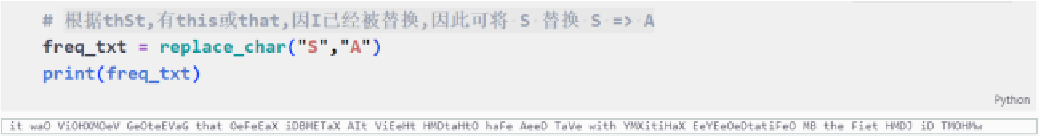
- 根据 Qith , Q 和 W 均为低频得, 可将 Q => W:



```python
freq_txt = replace_char("Q","W")
print(freq_txt)
```

```
it wSO ViOHXMOeV GeOteEVSG thSt OeFeESX iDBMETSX AIt ViEeHt HMDtSHtO hSFe AeeD TSVe with YMXitiHSX EeYEeOeDtStiFeO MB the Fiet HMDJ iD TMOHMw
```

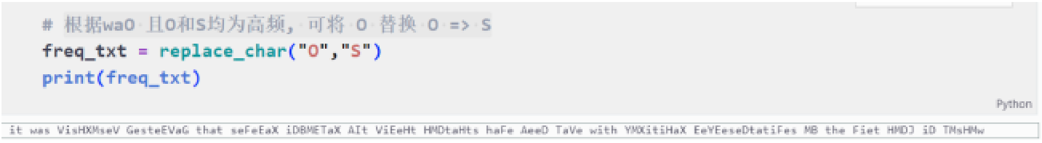
- 根据 thSt,有 this 或 that,因 I 已经被替换,因此可将 S 替换 S => A:



```python
# 根据thSt,有this或that,因I已经被替换,因此可将 S 替换 S => A
freq_txt = replace_char("S","A")
print(freq_txt)
```

```
it waO ViOHXMOeV GeOteEVaG that OeFeEaX iDBMETaX AIt ViEeHt HMDtaHtO haFe AeeD TaVe with YMXitiHaX EeYEeOeDtatiFeO MB the Fiet HMDJ iD TMOHMw
```

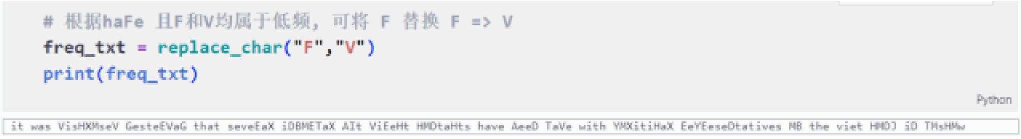
- 根据 waO 且 O 和 S 均为高频, 可将 O 替换 O => S:



```python
# 根据waO 且O和S均为高频, 可将 O 替换 O => S
freq_txt = replace_char("O","S")
print(freq_txt)
```

```
it was VisHXMseV GesteEVaG that seFeEaX iDBMETaX AIt ViEeHt HMDtaHts haFe AeeD TaVe with YMXitiHaX EeYEeseDtatiFes MB the Fiet HMDJ iD TMsHMw
```

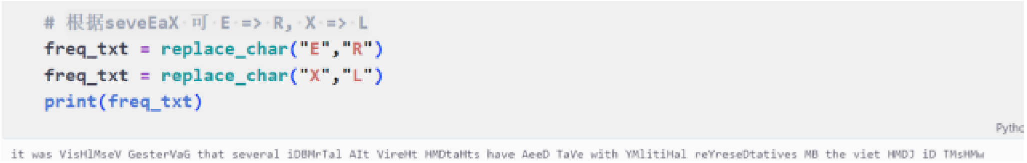
- 根据 haFe 且 F 和 V 均属于低频, 可将 F 替换 F => V:



```
# 根据haFe 且F和V均属于低频, 可将 F 替换 F => V
freq_txt = replace_char("F","V")
print(freq_txt)
```

```
it was VisHXMseV GesteEVaG that seveEaX iDBMETaX AIt ViEeHt HMDtaHts have AeeD TaVe with YMXitiHaX EeYEeseDtatives MB the viet HMDJ iD TMsHMw
```

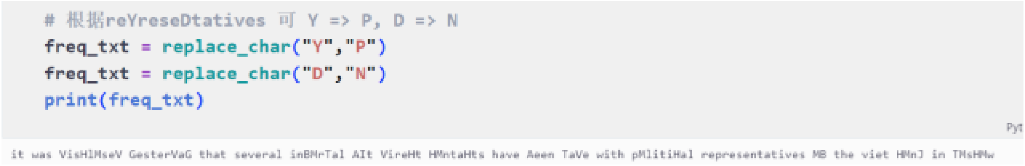
- 根据 seveEaX 可 E => R, X => L:



```
# 根据seveEaX 可 E => R, X => L
freq_txt = replace_char("E","R")
freq_txt = replace_char("X","L")
print(freq_txt)
```

```
it was VisHlMseV GesterVaG that several iDBMrTal AIt VireHt HMDtaHts have AeeD TaVe with YMlitiHal reYreseDtatives MB the viet HMDJ iD TMsHMw
```

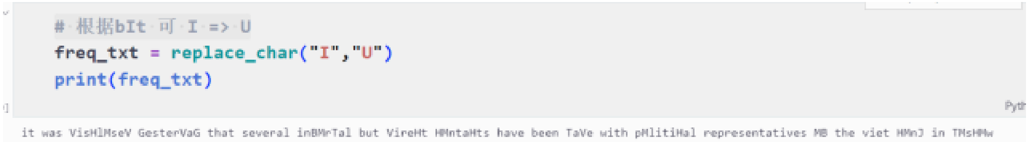
- 根据 reYreseDtatives 可 Y => P, D => N:



```
# 根据reYreseDtatives 可 Y => P, D => N
freq_txt = replace_char("Y","P")
freq_txt = replace_char("D","N")
print(freq_txt)
```

```
it was VisHlMseV GesterVaG that several inBMrTal AIt VireHt HMntaHts have Aeen TaVe with pMlitiHal representatives MB the viet HMnJ in TMsHMw
```

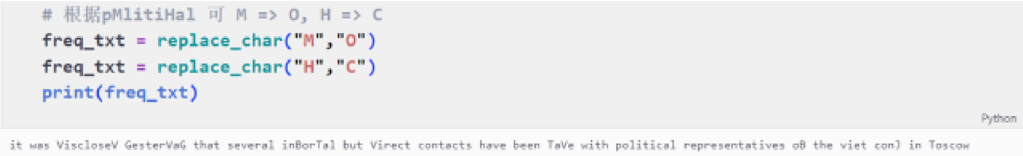
- 根据 Aeen 可 A => B，根据 bIt 可 I => U:



```
# 根据bIt 可 I => U
freq_txt = replace_char("I","U")
print(freq_txt)
```

```
it was VisHlMseV GesterVaG that several inBMrTal but VireHt HMntaMts have been TaVe with pMlitiHal representatives MB the viet HMnJ in TMsHMw
```

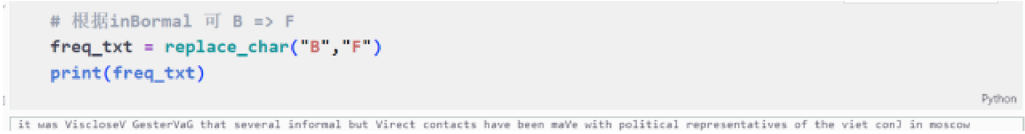
- 根据 pMlitiHal 可 M => O, H => C:



```
# 根据pMlitiHal 可 M => O, H => C
freq_txt = replace_char("M","O")
freq_txt = replace_char("H","C")
print(freq_txt)
```

```
it was ViscloseV GesterVaG that several inBorTal but Virect contacts have been TaVe with political representatives oB the viet conJ in Toscow
```

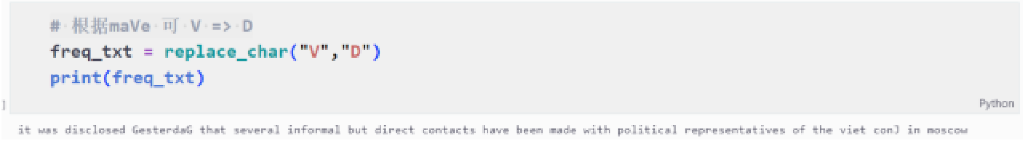
- 根据 Toscow 可 T => M，根据 inBormal 可 B => F:



```
# 根据inBormal 可 B => F
freq_txt = replace_char("B","F")
print(freq_txt)
```

```
it was ViscloseV GesterVaG that several informal but Virect contacts have been maVe with political representatives of the viet conJ in moscow
```

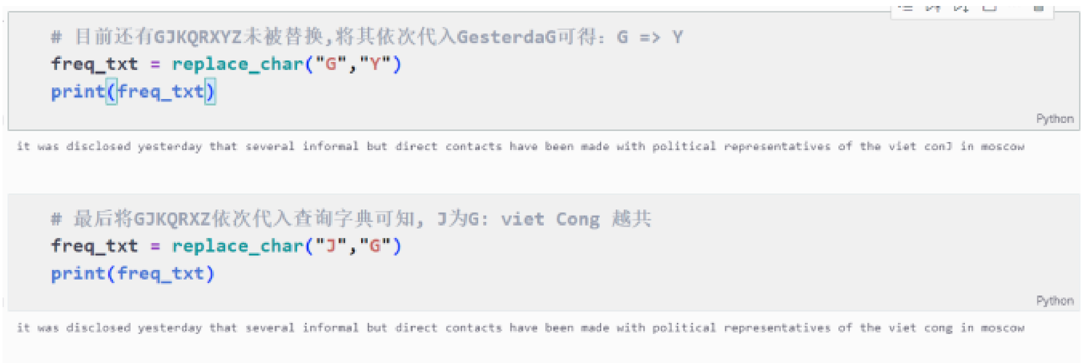
- 根据 maVe 可 V => D:



```
# 根据maVe 可 V => D
freq_txt = replace_char("V","D")
print(freq_txt)
```

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

最后:



```python
# 目前还有GJKQRXYZ未被替换,将其依次代入GesterdaG可得: G => Y
freq_txt = replace_char("G","Y")
print(freq_txt)
```

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

```python
# 最后将GJKQRXZ依次代入查询字典可知, J为G: viet Cong 越共
freq_txt = replace_char("J","G")
print(freq_txt)
```

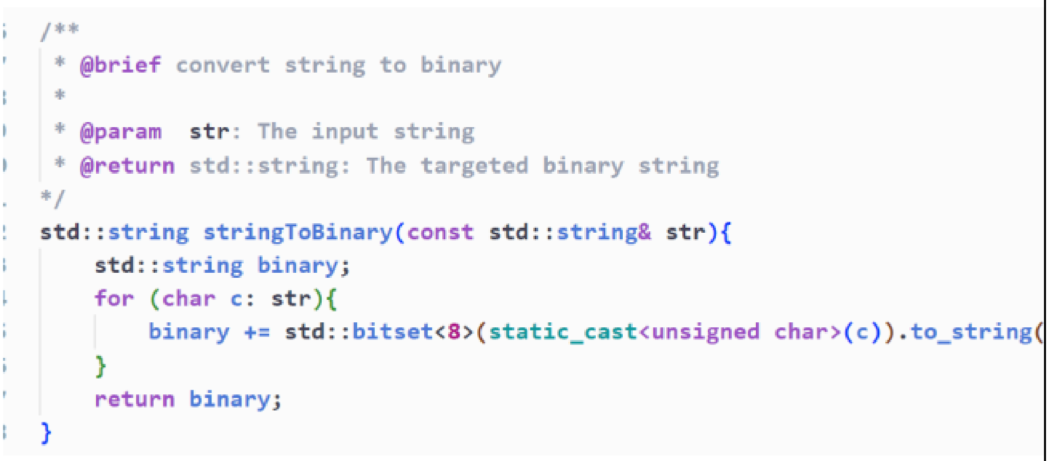
```
it was disclosed yesterday that several informal but direct contacts have been made with political representatives of the viet cong in moscow
```

得到的结果为:

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

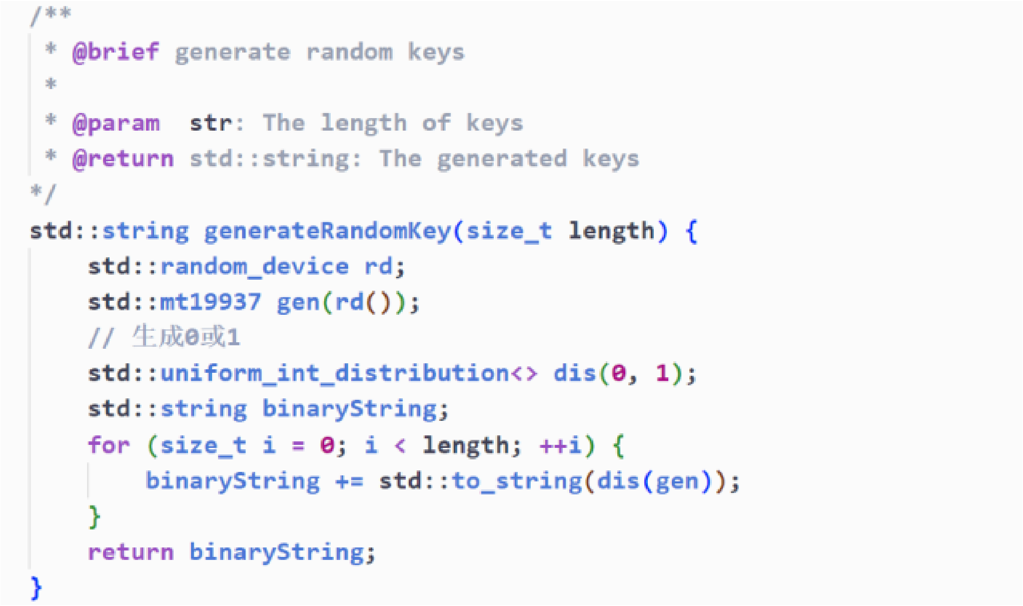
2. 编程实现 Feistel 加密解密

- 首先将文本字符串转化为二进制字符串，采用 bitset 库进行转化：



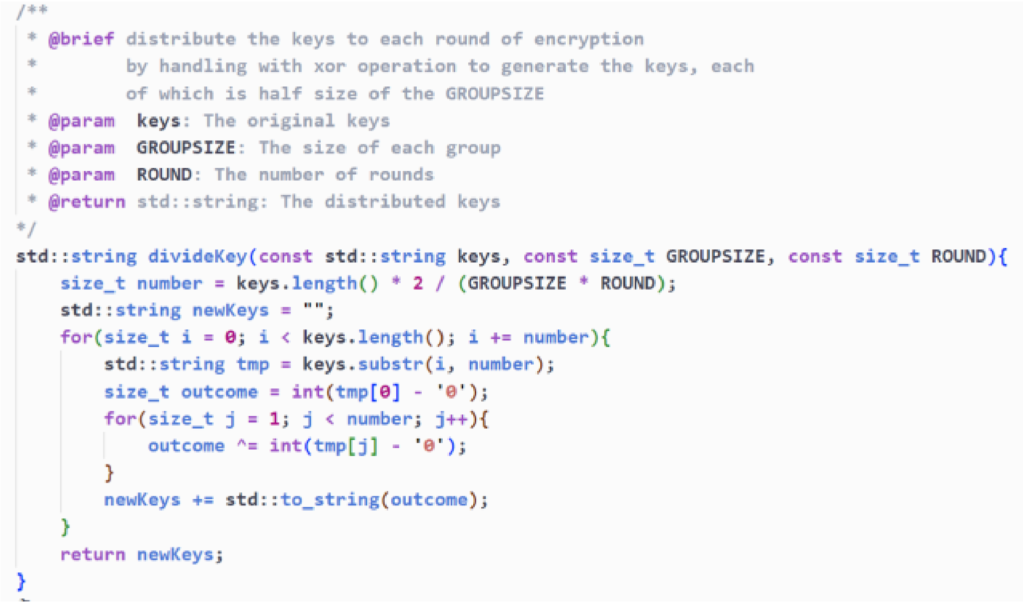
```cpp
/**
 * @brief convert string to binary
 *
 * @param  str: The input string
 * @return std::string: The targeted binary string
 */
std::string stringToBinary(const std::string& str){
    std::string binary;
    for (char c: str){
        binary += std::bitset<8>(static_cast<unsigned char>(c)).to_string(
    }
    return binary;
}
```

- 使用 random 库随机生成 128×16 位密钥，写入字符串中：



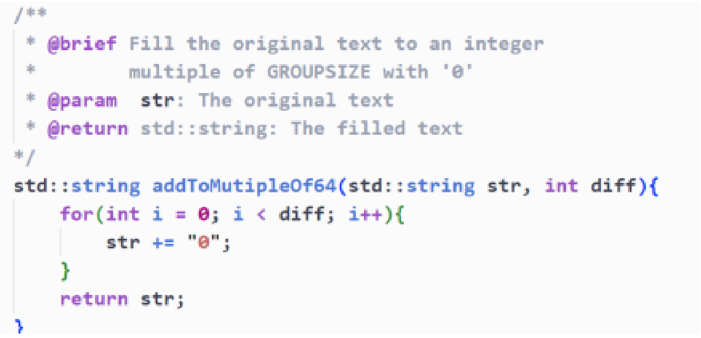
```cpp
/**
 * @brief generate random keys
 *
 * @param  str: The length of keys
 * @return std::string: The generated keys
*/
std::string generateRandomKey(size_t length) {
    std::random_device rd;
    std::mt19937 gen(rd());
    // 生成0或1
    std::uniform_int_distribution<> dis(0, 1);
    std::string binaryString;
    for (size_t i = 0; i < length; ++i) {
        binaryString += std::to_string(dis(gen));
    }
    return binaryString;
}
```

• **分发密钥：**为了便于异或运算，将冗余的密钥使用按 4 位分组进行组内异或的操作压缩为 512 位的密钥，每轮密钥长度为 32 位.



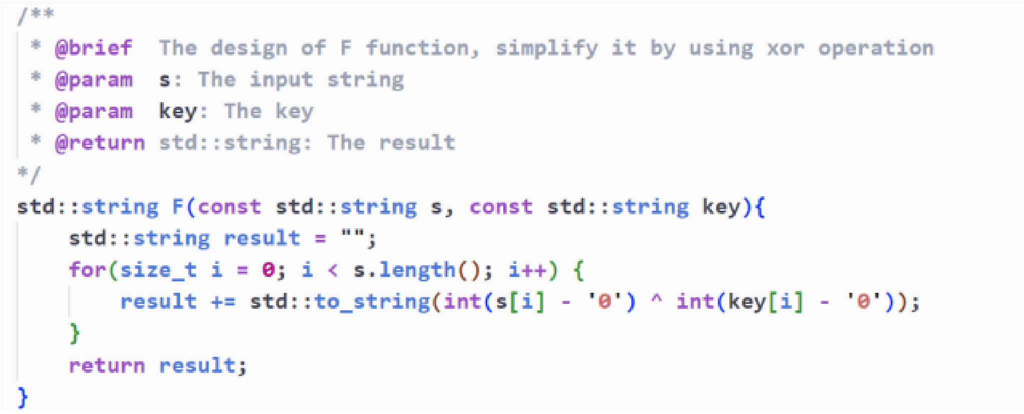
```cpp
/**
 * @brief distribute the keys to each round of encryption
 *        by handling with xor operation to generate the keys, each
 *        of which is half size of the GROUPSIZE
 * @param  keys: The original keys
 * @param  GROUPSIZE: The size of each group
 * @param  ROUND: The number of rounds
 * @return std::string: The distributed keys
*/
std::string divideKey(const std::string keys, const size_t GROUPSIZE, const size_t ROUND){
    size_t number = keys.length() * 2 / (GROUPSIZE * ROUND);
    std::string newKeys = "";
    for(size_t i = 0; i < keys.length(); i += number){
        std::string tmp = keys.substr(i, number);
        size_t outcome = int(tmp[0] - '0');
        for(size_t j = 1; j < number; j++){
            outcome ^= int(tmp[j] - '0');
        }
        newKeys += std::to_string(outcome);
    }
    return newKeys;
}
```

• **bit 填充：**对于待加密文本长度转化为二进制后不足 64 的倍数采用 0bit 填充，本实验中将原有的 200 位再最后填充至 256 位再进行分组计算。



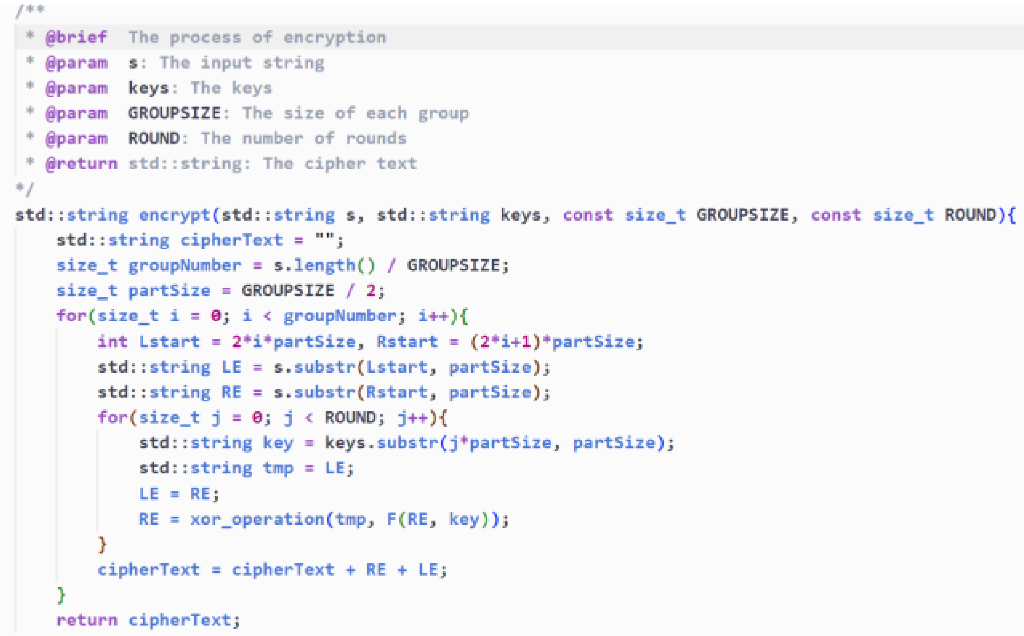
```cpp
/**
 * @brief Fill the original text to an integer
 *        multiple of GROUPSIZE with '0'
 * @param  str: The original text
 * @return std::string: The filled text
*/
std::string addToMutipleOf64(std::string str, int diff){
    for(int i = 0; i < diff; i++){
        str += "0";
    }
    return str;
}
```

• **F 函数：**在本实验中，为了简化运算，直接将原文的右半部分与密钥作异或运算：



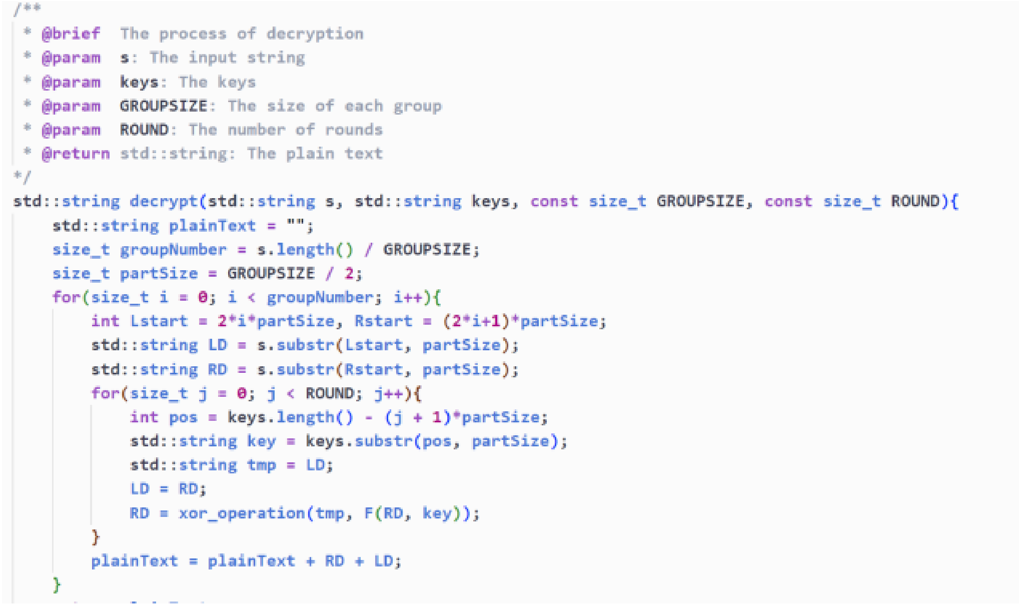
```
/**
 * @brief  The design of F function, simplify it by using xor operation
 * @param  s: The input string
 * @param  key: The key
 * @return std::string: The result
*/
std::string F(const std::string s, const std::string key){
    std::string result = "";
    for(size_t i = 0; i < s.length(); i++) {
        result += std::to_string(int(s[i] - '0') ^ int(key[i] - '0'));
    }
    return result;
}
```

• **加密过程：**本实验一共分成 4 组，16 轮，每轮分配一个不同密钥，先对 4 组一一进行加密，最后再将 4 组的加密结果进行拼接。



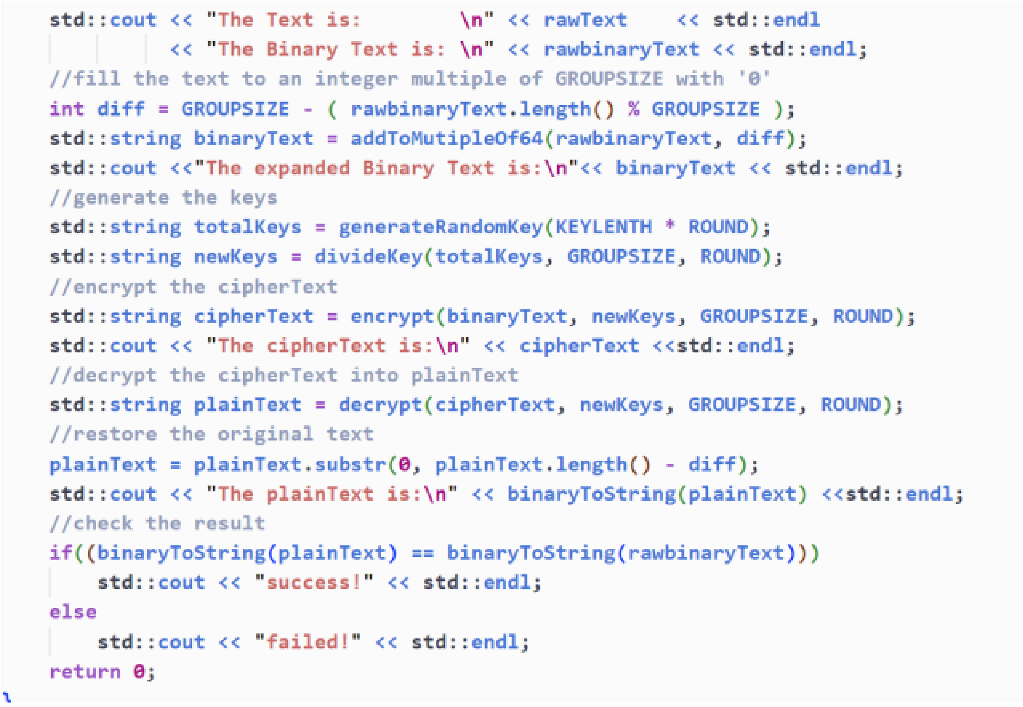
```
/**
 * @brief  The process of encryption
 * @param  s: The input string
 * @param  keys: The keys
 * @param  GROUPSIZE: The size of each group
 * @param  ROUND: The number of rounds
 * @return std::string: The cipher text
*/
std::string encrypt(std::string s, std::string keys, const size_t GROUPSIZE, const size_t ROUND){
    std::string cipherText = "";
    size_t groupNumber = s.length() / GROUPSIZE;
    size_t partSize = GROUPSIZE / 2;
    for(size_t i = 0; i < groupNumber; i++){
        int Lstart = 2*i*partSize, Rstart = (2*i+1)*partSize;
        std::string LE = s.substr(Lstart, partSize);
        std::string RE = s.substr(Rstart, partSize);
        for(size_t j = 0; j < ROUND; j++){
            std::string key = keys.substr(j*partSize, partSize);
            std::string tmp = LE;
            LE = RE;
            RE = xor_operation(tmp, F(RE, key));
        }
        cipherText = cipherText + RE + LE;
    }
    return cipherText;
```

• **解密过程：**思路与加密过程类似，只不过注意解密的密钥应与加密过程的分配顺序相反。



```
/**
 * @brief  The process of decryption
 * @param  s: The input string
 * @param  keys: The keys
 * @param  GROUPSIZE: The size of each group
 * @param  ROUND: The number of rounds
 * @return std::string: The plain text
*/
std::string decrypt(std::string s, std::string keys, const size_t GROUPSIZE, const size_t ROUND){
    std::string plainText = "";
    size_t groupNumber = s.length() / GROUPSIZE;
    size_t partSize = GROUPSIZE / 2;
    for(size_t i = 0; i < groupNumber; i++){
        int Lstart = 2*i*partSize, Rstart = (2*i+1)*partSize;
        std::string LD = s.substr(Lstart, partSize);
        std::string RD = s.substr(Rstart, partSize);
        for(size_t j = 0; j < ROUND; j++){
            int pos = keys.length() - (j + 1)*partSize;
            std::string key = keys.substr(pos, partSize);
            std::string tmp = LD;
            LD = RD;
            RD = xor_operation(tmp, F(RD, key));
        }
        plainText = plainText + RD + LD;
    }
```

• **将每一部分的结果进行输出验证对比：**



```
    std::cout << "The Text is:         \n" << rawText     << std::endl
              << "The Binary Text is: \n" << rawbinaryText << std::endl;
    //fill the text to an integer multiple of GROUPSIZE with '0'
    int diff = GROUPSIZE - ( rawbinaryText.length() % GROUPSIZE );
    std::string binaryText = addToMutipleOf64(rawbinaryText, diff);
    std::cout <<"The expanded Binary Text is:\n"<< binaryText << std::endl;
    //generate the keys
    std::string totalKeys = generateRandomKey(KEYLENTH * ROUND);
    std::string newKeys = divideKey(totalKeys, GROUPSIZE, ROUND);
    //encrypt the cipherText
    std::string cipherText = encrypt(binaryText, newKeys, GROUPSIZE, ROUND);
    std::cout << "The cipherText is:\n" << cipherText <<std::endl;
    //decrypt the cipherText into plainText
    std::string plainText = decrypt(cipherText, newKeys, GROUPSIZE, ROUND);
    //restore the original text
    plainText = plainText.substr(0, plainText.length() - diff);
    std::cout << "The plainText is:\n" << binaryToString(plainText) <<std::endl;
    //check the result
    if((binaryToString(plainText) == binaryToString(rawbinaryText)))
        std::cout << "success!" << std::endl;
    else
        std::cout << "failed!" << std::endl;
    return 0;
}
```

五、实验过程中遇到的问题及解决情况

1. **问题：**频度分析法中最后一个 conJ 单词有多种可能的情况，难以判别。

解决：通过逐个查字典分析，Vietcong 表示“越共”的意思，才得以解决。

2. **问题：**使用 random 方法生成随机数时发现每次生成的都一样，经查询，疑似编译器的问题。

解决：一次性生成 2048 位密钥，而不是每次生成 128 位密钥，然后进行截断

处理即可保证每轮的密钥不同。

3. **问题**：加密后的密文转化为字符字符串并输出后，发现代码后续部分没有执行。

解决：该问题未解决，最后采取的是分两次运行。猜测是乱码引起？

六、实验结果及分析和（或）源程序调试过程

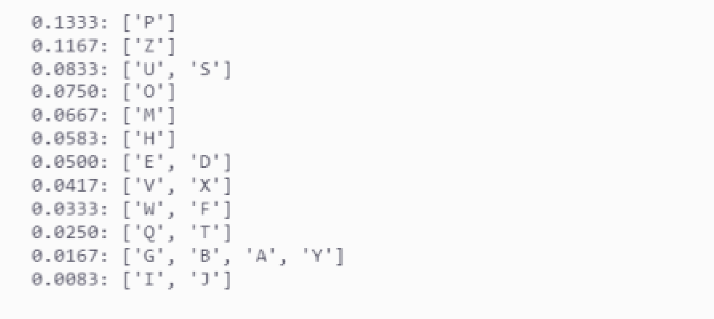
1. 频度分析法解密文本结果分析

1.1 解密结果为:

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

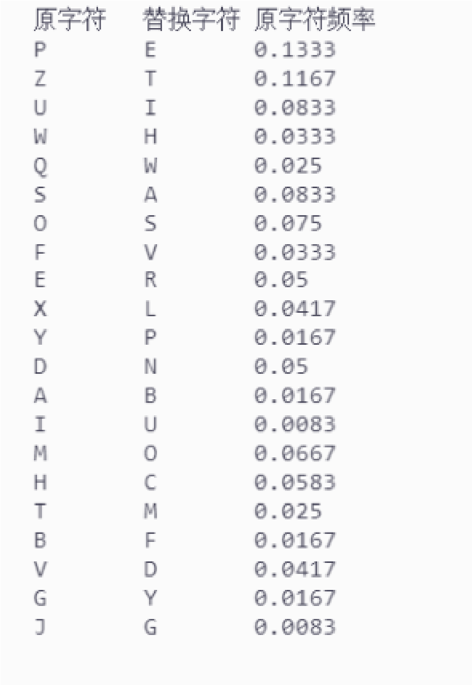
具有实际意义，由此可见密文解密正确。

1.2 得到的原始词频统计:



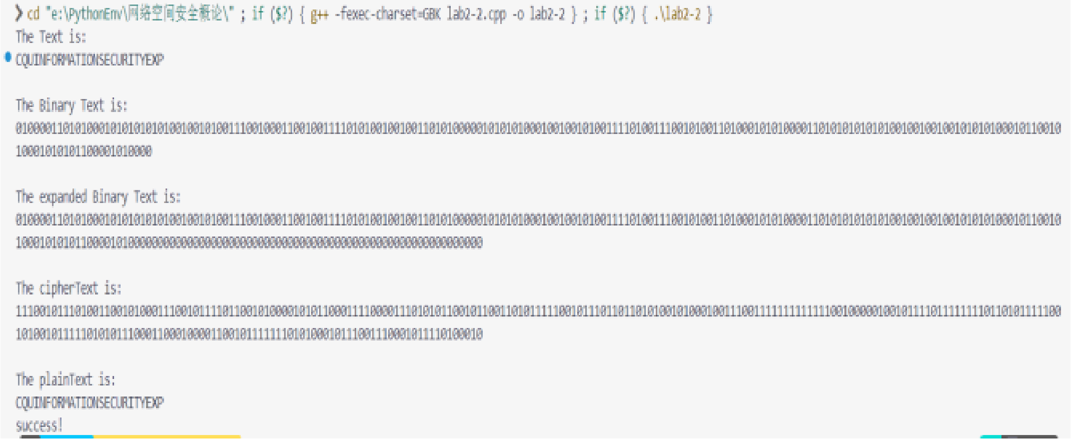
```
0.1333: ['P']
0.1167: ['Z']
0.0833: ['U', 'S']
0.0750: ['O']
0.0667: ['M']
0.0583: ['H']
0.0500: ['E', 'D']
0.0417: ['V', 'X']
0.0333: ['W', 'F']
0.0250: ['Q', 'T']
0.0167: ['G', 'B', 'A', 'Y']
0.0083: ['I', 'J']
```

1.3 对应的替换表为:



```
原字符    替换字符  原字符频率
P         E         0.1333
Z         T         0.1167
U         I         0.0833
W         H         0.0333
Q         W         0.025
S         A         0.0833
O         S         0.075
F         V         0.0333
E         R         0.05
X         L         0.0417
Y         P         0.0167
D         N         0.05
A         B         0.0167
I         U         0.0083
M         O         0.0667
H         C         0.0583
T         M         0.025
B         F         0.0167
V         D         0.0417
G         Y         0.0167
J         G         0.0083
```

2.Feistel 加密解密



```
❯ cd "e:\PythonEnv\网络空间安全概论\" ; if ($?) { g++ -fexec-charset=GBK lab2-2.cpp -o lab2-2 } ; if ($?) { .\lab2-2 }
The Text is:
CQUINFORMATIONSECURITYEXP

The Binary Text is:
010000110101000101010101010010010100111001000110010011110101001001001101010000010101010001001001010011110100111001010011010001010100001101010101010100100100100101010100010110010
10001010101100001010000

The expanded Binary Text is:
010000110101000101010101010010010100111001000110010011110101001001001101010000010101010001001001010011110100111001010011010001010100001101010101010100100100100101010100010110010
1000101010110000101000000000000000000000000000000000000000000000000000000000000

The cipherText is:
111001011101001100101000111001011110110010100001010110001111000011101010110010110011010111110010111011011010100101000100111001111111111111110010000001001011110111111110110101111100
1010010111111010101110001100010000110010111111010100010111001110001011110100010

The plainText is:
CQUINFORMATIONSECURITYEXP
success!
```

由控制台输出可知，经加密后解密得到的文本与原始文本一致，由此验证了实验结果的正确性。