刘修铭 2112492 信息安全

1.18

序号	初始向量	密钥流	周期
1	0000	0000...	1
2	0001	0001100011...	5
3	0010	0010100101...	5
4	0011	0011000110...	5
5	0100	0100101001...	5
6	0101	0101001010...	5
7	0110	0110001100...	5
8	0111	0111101111...	5
9	1000	1000110001...	5
10	1001	1001010010...	5
11	1010	1010010100...	5
12	1011	1011110111...	5
13	1100	1100011000...	5
14	1101	1101111011...	5
15	1110	1110111101...	5
16	1111	1111011110...	5

1.21

使用python完成编程，进行问题的求解：

```python
import numpy as np
from collections import defaultdict

# 计算字母频率
def calculate_letter_frequency(text):
    length = len(text)
    letter_count = defaultdict(int)
```

```
    for letter in text:
        letter_count[letter] += 1
    return letter_count, length

# 计算重合指数
def calculate_index_of_coincidence(text, letters):
    letter_count, length = calculate_letter_frequency(text)
    sum_term = sum([letter_count[letter] * (letter_count[letter] - 1) for
letter in letters])
    return sum_term / (length * (length - 1))

# 计算最佳m值
def calculate_optimal_m(text, letters):
    m_values = [0.0]
    for m in range(1, 11):
        print(f'm={m}', end=' ')
        substrings = ["" for _ in range(m)]
        for i, letter in enumerate(text):
            substrings[i % m] += letter
        sum_ic = 0.0
        for substring in substrings:
            ic = calculate_index_of_coincidence(substring, letters)
            sum_ic += ic
            print(f'{ic:.5f}', end=' ')
        print()
        m_values.append(sum_ic / m)
    optimal_m = np.array(m_values).argmax(axis=0)
    print(f'密钥长度 m = {optimal_m}')
    return optimal_m

# 计算密钥
def calculate_key(text, letters, letter_frequencies):
    m = calculate_optimal_m(text, letters)
    substrings = ["" for _ in range(m)]
    for i, letter in enumerate(text):
        substrings[i % m] += letter
    key = []
    for substring in substrings:
        print(substring + ', ', end='')
        letter_count, length = calculate_letter_frequency(substring)
        Mg_values = []
        for g in range(26):
            Mg = 0.0
            for i in range(26):
                Mg += letter_frequencies[i] * letter_count[letters[(i + g) %
26]]
            Mg /= length
            Mg_values.append(Mg)
        k = np.abs(np.array(Mg_values) - np.array(0.065)).argmin(axis=0)
        key.append(k)
        print(f'k = {k}, Mg({k}) = {Mg_values[k]:.5f}')
    return key
```

```
58
59
60  ciphertext =
    "KCCPKBGUFDPHOTYAVINRRTMVGRKDNBVFDETDGILTXRGUDDKOTFMBPVGEGLTGCKORACOCWDNAWCRXIZA
    KFTLEWRPTYCQKYVXCHKFTPONCQQRHJVAJUWETMCMSPKODYHJVDAHCTRISVSKCGCZOODZXGSFRLSWCWSJ
    TBHAFSIASPRJAHKJRJUMVGKMITZHFPDISPZLVLGWTFPLKKEBDPGCEBSHCTJRWXBAFSPEZONRWXCVYCGA
    ONWDDKACKAWBBIKFTIOVKCGGHJVLNHIFFSOESVYCLACNVRWBBIREPBBVFEXOSCDYGZWPFDTKFOIYCWHJ
    VLNHIOIBTKHJVNPIST"
61  letters = [chr(ord('A') + i) for i in range(26)]
62  letter_frequencies = [0.082, 0.015, 0.028, 0.043, 0.127, 0.022, 0.020, 0.061,
    0.070, 0.002, 0.008, 0.040, 0.024, 0.067, 0.075, 0.019, 0.001, 0.060, 0.063,
    0.091, 0.028, 0.010, 0.023, 0.001, 0.020, 0.001]
63
64  # 计算密钥
65  key = calculate_key(ciphertext, letters, letter_frequencies)
66  print(f'Key = {key}')
67
68  # 解密文本
69  def decrypt_text(ciphertext, key, letters):
70      plaintext = ""
71      n = len(key)
72      for i, letter in enumerate(ciphertext):
73          letter_value = ord(letter) - ord('A')
74          k = key[i % n]
75          plaintext += letters[(letter_value - k + 26) % 26]
76      return plaintext.lower()
77  plaintext = decrypt_text(ciphertext, key, letters)
78  print(f'明文: {plaintext}')
```

1. 定义函数 `calculate_letter_frequency(text)`：
   - 计算文本中每个字母的频率和文本长度
   - 返回一个包含字母频率和文本长度的元组
2. 定义函数 `calculate_index_of_coincidence(text, letters)`：
   - 计算文本的重合指数
   - `letters` 参数是字母表，用于计算期望的字母频率
   - 返回文本的重合指数。
3. 定义函数 `split_text(text, m)`：
   - 将文本拆分成 `m` 个子字符串，以便进行维吉尼亚密码的分组解密
   - 返回一个包含子字符串的列表
4. 定义函数 `calculate_optimal_m(text, letters)`：
   - 计算最佳的密钥长度 `m`。在不同的 `m` 值下，计算子字符串的重合指数，并输出结果以帮助选择最佳 `m`
   - 返回最佳的密钥长度 `m = 6`
5. 定义函数 `calculate_key(text, letters, letter_frequencies)`：

- 计算维吉尼亚密码的密钥。根据最佳 `m`，将文本拆分成子字符串，并计算每个子字符串的密钥，输出每个子字符串的密钥和对应的 `Mg` 值
- 返回包含所有子字符串密钥的列表

运行上述代码后，得到密钥：

```
Key = [2, 17, 24, 15, 19, 14]
```

对密文进行解密，得到明文串：

```
ilearnedhowtocalculatetheamountofpaperneededforaroomwheniwasatschoolyoumultiplyth
esquarefootageofthewallsbythecubiccontentsofthefloorandceilingcombinedanddoubleit
youthenallowhalfthetotalfnropeningssuchaswindowsanddoorsthenyouallowtheotherhalff
ormatchingthepatternthenyoudoublethewholethingagaintogiveamarginoferrorandthenyou
orderthepaper
```

对其进行分词，得到如下语句：

```
I learned how to calculate the amount of paper needed for a room when I was at
school. You multiply the square footage of the walls by the cubic contents of the
floor and ceiling combined and then double it. You then allocate half of the
total for openings such as windows and doors, and the other half for matching the
pattern. Afterward, you double the whole measurement again to account for any
margin of error, and then you place the order for the paper.
```