# 实 验 报 告


| | |
|---|---|
| 课程名称 | 密码学基础 |
| 学　　院 | 计算机科学技术学院 |
| 专　　业 | 信息安全 |
| 姓　　名 | 冉津豪 |
| 学　　号 | 17307130179 |

开 课 时 间 **2019** 至 **2020** 学年第 二 学期

| 实验项目名　　称 | Cracking Vigenère Cipher | 成绩 | |
|---|---|---|---|

## 一、实验目的

1. Understanding Vigenère Cipher
2. Understanding Kasiski Test
3. Understanding Friedman's Index of Coincidence Test
4. Understanding Frequency Analysis to crack Caesar Cipher.

## 二、实验内容

### 1. Vigenère Cipher

Vigenere 密码使用一个字符串作为密钥，字符串中的每一个字符都作为移位替换密码的密钥并确定一个替换表。设密钥 K=k1k2..kd,明文与密文表中均包含 n 个字母，明文 M=$m_1m_2$... 密文 C=$c_1c_2$....则加密：

$$Ci=(m_i+k_i)\bmod n$$

解密：

$$Mi=(c_i-k_i+n)\%n$$

在通常的 Vigenere 密码中，替换表由 26 个英文字母组成，周期循环为 26，若看成一维表，由于共 26 个字母，每个字母可做一个指定一个替换表，即共 26 个替换表，即总密文表长度为 26X26。

### 2. Kasiski Test

用 Vigenere 密码加密，明文中的相同字母在密文中一般不会对应相同的字母。但是，如果两个相同字母序列间距正好是密钥长度的倍数时，也可能产生相同的密文序列。寻找重复出现的字母序列，并求其长度的过程被称为 Kasiski 试验，即 Kasiski。

Kasiski 法在解密 Vigenere 密码时，利用的是多表体系的弱点：相同的明文字母组，在明文序列中间隔的字母数为 d（d 是密钥的长度）的倍数时，则明文字母组对应的密文字母组也必相同。反之则不一定，但相同的概率很大。如果将密文中相同字母组找出来，并对其间隔的距离进行研究，找出它们的最大公因子，则该因子是密钥长度的概率是较大的。

### 3. Index of Coincidence & Friedman Test

IC（重合指数）是指从长度为 N 的序列中，任意抽到两个字母相同的概率，用来衡量字母频率的不均匀程度。

$$IC = \frac{\sum_{i=1}^{c} n_i(n_i-1)}{N(N-1)}$$

$n_i$为某个字母在序列中出现的次数，c 为字母的数量，在英语中一般为 26。

Friedman Test 使用重合指数来破解维吉尼亚密码的密钥长度：

$$L \approx \frac{\kappa_p-\kappa_r}{IC-\kappa_r} = \frac{0.027N}{IC*(N-1)-0.0385N+0.0655}$$

其中，其余参数由英文语言特征决定。

## 三、实验步骤

1. Computing the chance for the false positive rate of Kasiski test

我们需要计算在长度为 L 的密文中，发生三元字符串偶然碰撞的概率。

$$K(N,r) = 1 - Q(N,r)$$

$$Q(N,r) = \frac{N \cdot (N-1) \cdots (N-r+1)}{N^r} = [1 - \frac{1}{N}] \cdots [1 - \frac{r-1}{N}].$$

长度为 L 的字符串一共有 L-2 个三元组，在 L 较大时，我们近似认为有 L 个三元组，因此 r = L。三元组全集中一共有 $26^3$ 个元素，所以 $N = 26^3$。

$$P = K(N, r) = 1 - \prod_{i=1}^{r-1}(1 - \frac{i}{N}) = 1 - \prod_{i=1}^{L-1}(1 - \frac{i}{26^3})$$

根据论文 ***Kasiski's Test: Couldn't the Repetitions be by Accident***，我们可以得到获得对 K(N, r)的一个大小估计：

**Theorem 3** *For $r$ random character strings of length $t$ over an alphabet of $n$ characters with $r \leq n^{t/2}$ the probability of a repetition is less than $\frac{1}{2}$.*

在这里 r = L, n = 26, t = 3。所以得出结论：

**当密文长度 $L \leq 26^{\frac{3}{2}} \approx 133$ 时，三元组偶然碰撞概率 $K(N, r) < \frac{1}{2}$.**

2. Implementing Vigenère Cipher

根据维吉尼亚密码的原理，实现维吉尼亚密码的加解密。

加密：

$$Ci=(m_i+k_i) \bmod n$$

```
1    def vigenere_encrypt(plaintext, key, alphabet):
2        """Returns ciphertext encrypted by vigenere cipher using key"""
3        ciphertext = ""
4        i = 0
5        final_plaintext = prepare_string(plaintext, alphabet)
6        for m in final_plaintext:
7            if i % len(key) == 0:
8                i = 0
9            ciphertext += i2c((c2i(m, alphabet) + c2i(key[i], alphabet)
10                       -2*c2i('A', alphabet)) % len(alphabet), alphabet)
11           i += 1
12       return ciphertext
```

解密：

$$Mi=(c_i-k_i+n)\%n$$

```
def vigenere_decrypt(ciphertext, key, alphabet):
    """Returns plaintext decrypted by vigenere cipher using key"""
    # WRITE YOUR CODE HERE!
    plaintext = ""
    i = 0
    final_ciphertext = prepare_string(ciphertext, alphabet)
    for m in final_ciphertext:
        if i % len(key) == 0:
            i = 0
        plaintext += i2c((c2i(m, alphabet) + len(alphabet)
                - c2i(key[i], alphabet)
                -2*c2i('A', alphabet)) % len(alphabet), alphabet)
        i += 1
    return plaintext
```

3. Determining the Key Size

我们需要通过进行 Kasiski Test 或者 Friedman Test 来对最有可能的密钥长度进行估计：

1) Kasiski Test

只需完成对每两个重复的三元词组之间的距离统计即可。完成统计后，根据 Kasiski Test 的原理，大多数密文相同的三元组是相同的密钥位加密的结果，所以大多数距离都是密钥长度的整数倍。将重复次数最多的 6 种（根据密文长度及结果可适当调整）距离作为参数，计算它们的公约数，在本次实验中，得出结果为 7，是一个较为理想的结果，作为最有可能的密钥长度。

```
def kasiski_test(ciphertext):  #Code partially provided
    """Finds gcd of most common distances between
    repeated trigraphs"""
    trigraphs = []
    distances = []
    temp = ''
    for i in range(len(ciphertext)-2):
        temp = ciphertext[i:i+3]
        for j in range(i+1, len(ciphertext)-2):
            if temp == ciphertext[j:j+3]:
                trigraphs.append(temp)
                distances.append(j-i)
    dCount = Counter(distances)
    topCount = dCount.most_common(6)
    my_gcd = topCount[0][0]
    for index in range(1, len(topCount)):
        if topCount[index][1] > 1:
            my_gcd = gcd(my_gcd, topCount[index][0])
    return my_gcd
```

2) Friedman Test

根据 IC 的计算方法，计算从长度为 N 的序列中，任意抽到两个字母相同的概率。经过 Friedman Test 之后得到的结果为 4.411864846855899。

由于密文文本的长度限制，我们选择 Kasiski Test 作为猜测结果，即猜测密钥长度 7。

```
def index_of_coincidence(ciphertext, alpha):
    """Caculates index of coincidnece of ciphertext"""
    common = Counter(ciphertext)
    ioc = 0
    sum = 0
    for index in alpha:
        if common[index] > 2:
            sum += common[index]*(common[index]-1)
    n = len(ciphertext)
    ioc = sum/(n*(n-1))
    return ioc

def friedman_test(ciphertext, alpha):
    l = 0
    n = len(ciphertext)
    i = index_of_coincidence(ciphertext, alpha)
    l = n * (0.027)/((n-1)*i + 0.0655 - 0.0385 * n)
    return l
```

4. Determining the Keyword

有了密钥长度之后，我们再将密文按照密钥长度 7 划分为 7 个子序列，保证每个子序列都是由密钥的相同字母加密的。

对于每个子序列，使用 find_likely_letters，根据英文语言习惯，对这个加密的密钥字母进行推测。

find_likely_letters

在函数中对序列出现的每个字母进行频率统计。由于序列是由单个字母加密得到的，所以序列的频率都有一个偏移值，我们要做的就是找到这个偏移值，也就可以找到加密的字母了。我们对得到频率序列与英语语言统计的频率序列做比较，将差值记录下来，然后对频率序列进行类似解密的偏移：$Mi=(c_i-k_i+n)\%n$，用 rotate_list 每次将频率序列头部的项移到序列的最后，再进行频率差值计算。最终得到一个差值列表，差值最小的两项就是最有可能的偏移量得出的频率序列，同时指出该偏移对应的字母。

在此例中得出的密钥为：BITCOIN

```
def find_likely_letters(coset, alpha, eng_freq):
    """Finds the most likely shifts for each coset and prints them"""
    coset_freq = []
    differences = []
    # WRITE YOUR CODE HERE!
    n1 = len(coset)
    n2 = len(alpha)
    count = [0]*n2
```

```
        for m in coset:
            count[c2i(m, alpha)] += 1
        for i in range(n2):
            coset_freq.append(count[i]/n1)
        for i in range(n2):
            differences.append(find_total_difference(coset_freq,
    eng_freq))
            coset_freq = rotate_list(coset_freq)
        firstletter = differences.index(min(differences))
        differences[firstletter] = float("inf")
        secondletter = differences.index(min(differences))
        # WRITE YOUR CODE HERE!
        letter1 = alpha[firstletter]
        letter2 = alpha[secondletter]
        return "the most likely letter is: "
               + letter1 + " followed by: " + letter2
```

5. Encrypt the Plaintext

使用密钥对密文进行解密，得到相应的明文，根据解密内容可以判断解密正确。经过搜索查找，得到这部分明文是 Bitcoin: A Peer-to-Peer Electronic Cash System 的 Introduction。

```
Your cipher text is: DWFOSZPFWGVVMVOBXTBMGIILECURUWKGZGNMUHUHMKDTNUWDRMGHP
Kasiski test gives this as the most likely: 7
 Friedman test gives: 4.411864846855899
Choose the key length you'd like to try: 7
For coset 1, the most likely letter is: B followed by: M.
For coset 2, the most likely letter is: I followed by: M.
For coset 3, the most likely letter is: T followed by: I.
For coset 4, the most likely letter is: C followed by: R.
For coset 5, the most likely letter is: O followed by: S.
For coset 6, the most likely letter is: I followed by: T.
For coset 7, the most likely letter is: N followed by: R.
Type the key you would like to use to decipher: BITCOIN
COMMERCEONTHEINTERNETHASCOMETORELYALMOSTEXCLUSIVELYONFINANCIALINSTITUTIONS
```

*Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model. Completely non-reversible transactions are not really possible, since financial institutions cannot avoid mediating disputes. The cost of mediation increases transaction costs, limiting the minimum practical transaction size and cutting off the possibility for small casual transactions, and there is a broader cost in the loss of ability to make non-reversible payments for nonreversible services. With the possibility of reversal, the need for trust spreads. Merchants must be wary of their customers, hassling them for more information than they would otherwise need. A certain percentage of fraud is accepted as unavoidable. These costs and payment uncertainties can be avoided in person by using physical currency, but no mechanism exists to make payments over a communications channel without a trusted party.*

## 四、实验结果及分析

根据密钥以及尝试解密后的明文，我们可以判断成功破解了这段维吉尼亚密码。下面对实验的相关复杂度进行分析。

1. Vigenère Cipher 加解密
   加解密仅对明文或密文进行一次遍历，所以复杂度为 O(n)。
2. Kasiski Test
   对于每个三元组，都需要对密文余下的部分进行遍历，复杂度为 $O(n^2)$。
3. Friedman Test
   仅需要对密文的每个字母进行一次频率统计，复杂度为 O(n)。
4. Find Likely Letters
   仅需要对密文的每个字母进行一次频率统计，复杂度为 O(n)。

总体而言，“Cracking Vigenère Cipher”整体的算法复杂度取决于 Kasiski Test，而由于 Kasiski Test 和 Friedman Test 的可代替性，当密文长度过长的时候可考虑只进行 Friedman Test。但对于现代计算机而言，复杂度 $O(n^2)$不至于造成过重的计算负担，所以结合两者使用更佳。

## 五、实验总结

通过“Cracking Vigenère Cipher”实验，我们可以更加清楚地了解维吉尼亚密码，用实际的代码实现维吉尼亚密码的整个过程，更重要的是深刻理解了维吉尼亚密码的缺陷，即没有摆脱英语语言习惯，可以用概率来有效破解密码，并通过这个缺陷完成了一次维吉尼亚密码的破解。由此可见，脱离语言习惯对于一种密码的重要性。