山东大学 计算机科学与技术 学院

云计算技术 课程实验报告

学号：202200130048	姓名： 陈静雯	班级： 6班
实验题目：维吉尼亚密码实现和破解		
实验学时：2	实验日期： 2025.4.16	

实验目的：熟悉维吉尼亚密码算法、了解其安全性。
具体包括：1）实现维吉尼亚密码算法，可以利用任意长度密钥对任意长度句子进行加解密，并在 Docker 中部署；2）通过 Kasiski 和 Friedman 测试方法实现密钥破解。

硬件环境：
计算机一台

软件环境：
Docker，ubuntu，vmware

实验步骤：

1. 实现维吉尼亚密码算法
2. docker 部署
   （1）编写 dockerfile（2）构建并运行容器
3. 运行 vigenere，实现加密和解密
4. 在 docker 中运行 vigenere，实现加密和解密
5. 密钥破解，Kasiski 和 Friedman 测试方法

前置知识：

1. 维吉尼亚密码算法

（1）加密过程

a）**字母转数字：**
- 将明文和密钥中的字母转换为数字（A=0，B=1，...，Z=25）。

b）**扩展密钥：**
- 若密钥长度小于明文，循环重复密钥直至与明文等长。

c）**逐字符加密：**
- 对每个明文字符 **P_i** 和对应密钥字符 **K_j**，计算密文字符：

$$Ci=(Pi+Kj)\bmod 26$$

（2）解密过程

a）**字母转数字：**
- 将密文和密钥中的字母转换为数字。

b）**扩展密钥：**
- 与加密过程相同，循环使用密钥匹配密文长度。

c）**逐字符解密：**
- 对每个密文字符 **C_i** 和对应密钥字符 **K_j**，计算明文字符：

$$Pi=(Ci-Kj)\bmod 26$$

（3）示例：明文 hello world，密钥 KEY

加密过程：

a） 预处理明文

移除非字母字符（如空格），并将所有字母转换为大写：

- 明文：hello world → HELLOWORLD

b） 扩展密钥

密钥 KEY 需要循环扩展至与明文长度（10 个字母）一致：

- 密钥扩展：K E Y K E Y K E Y K

  （对应位置：1-K, 2-E, 3-Y, 4-K, 5-E, 6-Y, 7-K, 8-E, 9-Y, 10-K）

c） 字母转数字

字母映射为数字（A=0, B=1, ..., Z=25）：

- 明文 HELLOWORLD：

  H(7) E(4) L(11) L(11) O(14) W(22) O(14) R(17) L(11) D(3)

- 密钥 KEYKEYKEYKE：

  K(10) E(4) Y(24) K(10) E(4) Y(24) K(10) E(4) Y(24) K(10)

d） 逐字符加密

加密公式：密文字母 = (明文字母 + 密钥字母) mod 26

密文结果：RIJVSUVYJN

解密过程

a） 预处理密文

密文保留字母形式：RIJVSUVYJN

b） 扩展密钥

密钥扩展与加密过程一致：KEYKEYKEYKE

c） 字母转数字

密文 RIJVSUVYJN：

R(17) I(8) J(9) V(21) S(18) U(20) Y(24) V(21) J(9) N(13)

密钥 KEYKEYKEYKE：

K(10) E(4) Y(24) K(10) E(4) Y(24) K(10) E(4) Y(24) K(10)

d） 逐字符解密

解密公式：明文字母 = (密文字母 - 密钥字母) mod 26

明文结果：HELLOWORLD

2. Kasiski

- **核心思想**：通过分析密文中重复出现的字母序列，推测密钥长度。
- **步骤**：
  1. **寻找重复序列**：在密文中查找至少重复出现三次的相同字符序列（例如 3 字符序列）。
  2. **计算距离**：记录这些重复序列的起始位置，并计算它们之间的距离。
  3. **求最大公约数（GCD）**：分析这些距离的因数，推测可能的密钥长度（例如，若多个距离的 GCD 为 6，则密钥长度可能是 6 的因数）。
- **应用场景**：适合密文较长且有明显重复模式的情况。例如，当明文片段被相同的密钥字母多次加密时，Kasiski 方法能有效找到密钥长度的候选值。

3. Friedman
   - **核心思想**：基于重合指数（Index of Coincidence, IC）来估计密钥长度。
   - **步骤**：
     1. **计算重合指数（IC）**：对密文按不同密钥长度分组后，计算每组的 IC 值（即两个随机字母相同的概率）。
     2. **比较 IC 值**：自然语言的 IC 值较高（例如英文约为 0.0667），而随机文本的 IC 值较低（约 0.0385）。若分组正确（即每组使用同一密钥字母加密），分组的 IC 值会接近自然语言。
     3. **选择最佳密钥长度**：选择使平均 IC 值最大的密钥长度。
   - **应用场景**：适合密文较短或重复模式不明显的情况。Friedman 方法通过统计特性弥补 Kasiski 方法的不足。

实验内容：

1. **实现维吉尼亚密码算法**

```python
import argparse

def vigenere_encrypt(plaintext, key):
    key = ''.join([k.upper() for k in key if k.isalpha()])
    if not key:
        return plaintext  # No valid key, return plaintext as is
    key_len = len(key)
    ciphertext = []
    key_index = 0
    for c in plaintext:
        if c.isalpha():
            shift = ord(key[key_index % key_len]) - ord('A')
            if c.isupper():
                new_char = chr((ord(c) - ord('A') + shift) % 26 + ord('A'))
            else:
                new_char = chr((ord(c) - ord('a') + shift) % 26 + ord('a'))
            ciphertext.append(new_char)
            key_index += 1
        else:
            ciphertext.append(c)
    return ''.join(ciphertext)

def vigenere_decrypt(ciphertext, key):
    key = ''.join([k.upper() for k in key if k.isalpha()])
    if not key:
        return ciphertext  # No valid key, return ciphertext as is
    key_len = len(key)
    plaintext = []
    key_index = 0
```

```
    for c in ciphertext:
        if c.isalpha():
            shift = ord(key[key_index % key_len]) - ord('A')
            if c.isupper():
                new_char = chr((ord(c) - ord('A') - shift) % 26 + ord('A'))
            else:
                new_char = chr((ord(c) - ord('a') - shift) % 26 + ord('a'))
            plaintext.append(new_char)
            key_index += 1
        else:
            plaintext.append(c)
    return ''.join(plaintext)

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Vigenère cipher tool')
    parser.add_argument('action', choices=['encrypt', 'decrypt'], help='Action to perform: encrypt or decrypt')
    parser.add_argument('text', help='The text to process')
    parser.add_argument('key', help='The encryption key')
    args = parser.parse_args()

    if args.action == 'encrypt':
        print(vigenere_encrypt(args.text, args.key))
    else:
        print(vigenere_decrypt(args.text, args.key))
```

2. docker 部署

（1）dockerfile

vigenere.py | Dockerfile

```
FROM python:3.9-slim
WORKDIR /app
COPY vigenere.py .
ENTRYPOINT ["python", "vigenere.py"]
```

（2）构建并运行容器

```
orange@orange-VMware-Virtual-Platform:~/vigenere$ docker build -t vigenere .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  4.608kB
Step 1/4 : FROM python:3.9-slim
3.9-slim: Pulling from library/python
8a628cdd7ccc: Pull complete
74018f7cfa8f: Pull complete
a0b0cfc480ce: Pull complete
97d21b95fb00: Pull complete
Digest: sha256:9aa5793609640ecea2f06451a0d6f379330880b413f954933289cf3b27a78567
Status: Downloaded newer image for python:3.9-slim
 ---> 501f96d59d70
Step 2/4 : WORKDIR /app
 ---> Running in 90ace46481a7
 ---> Removed intermediate container 90ace46481a7
 ---> 51fd6f8d20fc
Step 3/4 : COPY vigenere.py .
 ---> 61d780cb2f26
Step 4/4 : ENTRYPOINT ["python", "vigenere.py"]
 ---> Running in cf74719f7dc9
 ---> Removed intermediate container cf74719f7dc9
 ---> fc43380914f3
Successfully built fc43380914f3
Successfully tagged vigenere:latest
orange@orange-VMware-Virtual-Platform:~/vigenere$ docker run -it --rm vigenere encrypt "Hello World" "KEY"
Rijvs Uyvjn
```

3. 运行 vigenere，实现加密和解密

```
orange@orange-VMware-Virtual-Platform:~/vigenere$ python3 vigenere.py encrypt "Hello World" "KEY"

Rijvs Uyvjn
orange@orange-VMware-Virtual-Platform:~/vigenere$ python3 vigenere.py decrypt "Rijvs Uyvjn" "KEY"

Hello World
```

4. 在 docker 中运行 vigenere，实现加密和解密

```
orange@orange-VMware-Virtual-Platform:~/vigenere$ docker run -it --rm vigenere encrypt "Hello World" "KEY"
Rijvs Uyvjn
orange@orange-VMware-Virtual-Platform:~/vigenere$ docker run -it --rm vigenere decrypt "Rijvs Uyvjn" "KEY"
Hello World
```

5. 密钥破解，Kasiski 和 Friedman 测试方法

vigenere.py | Dockerfile | cracker

```python
import re
from collections import Counter
from itertools import cycle
import string

def decrypt_vigenere(ciphertext, key):
    """使用密钥解密密文"""
    key_cycle = cycle(key.upper())
    plaintext = []
    for char in ciphertext:
        if not char.isalpha():
            plaintext.append(char)
            continue
        shift = ord(next(key_cycle)) - ord('A')
        decrypted_char = chr( (ord(char.upper()) - ord('A') - shift) % 26 + ord('A') )
        plaintext.append(decrypted_char)
    return ''.join(plaintext)

def kasiski_examination(ciphertext):
    """Kasiski方法：检测3字符重复子串并推测密钥长度"""
    substrings = {}
    # 扫描3字符重复序列
    for i in range(len(ciphertext) - 3):
        substring = ciphertext[i:i+3]
        substrings.setdefault(substring, []).append(i)

    # 计算重复序列间距
    distances = []
    for positions in substrings.values():
        if len(positions) > 1:
            for i in range(len(positions) - 1):
                distances.append(positions[i+1] - positions[i])

    # 因数分解统计
    factor_counts = Counter()
    for dist in distances:
        for factor in range(2, min(dist, 20) + 1):
            if dist % factor == 0:
                factor_counts[factor] += 1

    # 返回最高频的候选（至少为1）
    return factor_counts.most_common(1)[0][0] if factor_counts else 1

def friedman_test(ciphertext):
    """Friedman测试：基于重合指数推测密钥长度"""
    n = len(ciphertext)
    if n < 2:
        return 1

    # 计算重合指数（IC）
    freq = Counter(ciphertext)
    numerator = sum(f * (f - 1) for f in freq.values())
    denominator = n * (n - 1)
    ic = numerator / denominator if denominator != 0 else 0

    # 处理分母非正的情况
    friedman_denominator = (n - 1) * ic - 0.038 * n + 0.065
```

```
    # 处理分母非正的情况
    friedman_denominator = (n - 1) * ic - 0.038 * n + 0.065
    if friedman_denominator <= 0:
        return 1

    k = (0.027 * n) / friedman_denominator
    return max(1, round(k))

def get_best_key_length(ciphertext):
    """综合Kasiski和Friedman结果选择最佳密钥长度"""
    kasiski_len = kasiski_examination(ciphertext)
    friedman_len = friedman_test(ciphertext)
    print(f"[阶段1] Kasiski推测长度: {kasiski_len}")
    print(f"[阶段1] Friedman推测长度: {friedman_len}")

    # 候选池：Kasiski、Friedman及短密文遍历
    candidates = []
    if kasiski_len > 1:
        candidates.append(kasiski_len)
    candidates.append(friedman_len)
    if len(ciphertext) < 20:
        candidates += list(range(1, 6))

    # 选择IC最接近0.066的候选
    best_len, min_diff = 1, float('inf')
    for L in set(candidates):
        ics = []
```

```
    # 选择IC最接近0.066的候选
    best_len, min_diff = 1, float('inf')
    for L in set(candidates):
        ics = []
        for i in range(L):
            subgroup = ciphertext[i::L]
            if len(subgroup) < 2:
                continue
            freq = Counter(subgroup)
            numerator = sum(f * (f - 1) for f in freq.values())
            denominator = len(subgroup) * (len(subgroup) - 1)
            ic = numerator / denominator if denominator != 0 else 0
            ics.append(ic)
        if not ics:
            continue
        avg_ic = sum(ics) / len(ics)
        ic_diff = abs(avg_ic - 0.066)
        if ic_diff < min_diff:
            best_len, min_diff = L, ic_diff
    return best_len

def break_vigenere_cipher(ciphertext):
    """主破解函数"""
    ciphertext = re.sub(r'[^A-Z]', '', ciphertext.upper())
    print(f"净化后的密文: {ciphertext}")

    key_length = get_best_key_length(ciphertext)
    print(f"[阶段1] 最终使用长度: {key_length}")
```

```
    ciphertext = re.sub(r'[^A-Z]', '', ciphertext.upper())
    print(f"净化后的密文: {ciphertext}")

    key_length = get_best_key_length(ciphertext)
    print(f"[阶段1] 最终使用长度: {key_length}")

    # 标准英文频率表
    english_freq = {
        'A': 0.08167, 'B': 0.01492, 'C': 0.02782, 'D': 0.04253,
        'E': 0.12702, 'F': 0.02228, 'G': 0.02015, 'H': 0.06094,
        'I': 0.06966, 'J': 0.00153, 'K': 0.00772, 'L': 0.04025,
        'M': 0.02406, 'N': 0.06749, 'O': 0.07507, 'P': 0.01929,
        'Q': 0.00095, 'R': 0.05987, 'S': 0.06327, 'T': 0.09056,
        'U': 0.02758, 'V': 0.00978, 'W': 0.02360, 'X': 0.00150,
        'Y': 0.01974, 'Z': 0.00074
    }

    # 分组频率分析
    key = []
    for i in range(key_length):
        nth_letters = ciphertext[i::key_length]
        if not nth_letters:
            key.append('A')
            continue

        best_shift = 0
        min_deviation = float('inf')
        for possible_shift in range(26):
            # 计算位移后的字母分布
            shifted = [
                chr((ord(c) - ord('A') - possible_shift) % 26 + ord('A'))
                for c in nth_letters
            ]
            freq = Counter(shifted)
            total = len(shifted)

            # 计算卡方差异
            deviation = 0
            for char in string.ascii_uppercase:
                expected = english_freq[char] * total
                observed = freq.get(char, 0)
                if expected > 0:
                    deviation += (observed - expected) ** 2 / expected

            if deviation < min_deviation:
                min_deviation = deviation
                best_shift = possible_shift

        key.append(chr(ord('A') + best_shift))

    final_key = ''.join(key)
    print(f"[阶段2] 推测密钥: {final_key}")
    plaintext = decrypt_vigenere(ciphertext, final_key)
    print(f"[最终结果] 解密文本: {plaintext}")
    return plaintext
```

```python
if __name__ == "__main__":
    ciphertext = "LXFOPVEFRNHR"
    print("破解过程：")
    break_vigenere_cipher(ciphertext)
```

```
orange@orange-VMware-Virtual-Platform:~/vigenere$ python3 cracker.py "LXFOPVEFRNHR"
破解过程：
净化后的密文: LXFOPVEFRNHR
[阶段1] Kasiski推测长度: 1
[阶段1] Friedman推测长度: 1
[阶段1] 最终使用长度: 3
[阶段2] 推测密钥: ATN
[最终结果] 解密文本: LESOWIEMENOE
```

结论分析与体会：

1. Kasiski 方法：依赖重复序列的几何分布，适合密文较长的情况。
2. Friedman 方法：基于统计特性（IC 值），适合密文较短或无重复序列的情况。
3. 代码中的策略：优先使用 Kasiski 方法，若失败则切换至 Friedman 方法，确保在不同场景下均能有效推测密钥长度。