# Отчёт по лабораторной работе №2

Ван Яо

# Содержание

# Список таблиц

# Список иллюстраций

# Цель работы

1. Изучить теоретические основы шифров перестановки
2. Реализовать программно три алгоритма шифрования:
    - Маршрутное шифрование (столбцовая перестановка)
    - Шифрование с помощью решеток
    - Шифр Виженера
3. Исследовать криптографические свойства и особенности каждого метода

# Ход лабораторной работы

## Теоретические основы

### Основные понятия

**Шифр перестановки** — криптографический алгоритм, в котором символы открытого текста переставляются согласно определенному правилу, без изменения самих символов.

**Ключевое требование**: равенство длин ключа и исходного текста.

### Маршрутное шифрование

Разработано французским математиком Франсуа Виетом.

**Принцип работы**: 1. Текст разбивается на блоки длины $mn$ 2. Блоки записываются в таблицу $m \times n$ построчно 3. Шифртекст получается выписыванием символов по определенному маршруту

**Пример**: - Исходный текст: “нельзя недооценивать противника” - Пароль: “пароль” - Результат: “ЕЕНПНЗОАТАЬОВОКННЕЬВЯЦТИА”

### Шифрование с помощью решеток

Предложено австрийским криптографом Эдуардом Флейснером в 1881 году.

**Алгоритм построения решетки**: 1. Выбирается натуральное число $k > 1$ 2. Строится квадрат $k \times k$, заполняется числами от 1 до $k^2$ 3. Квадрат поворачивается и присоединяется к исходному 4. После 4 поворотов получается большой квадрат $2k \times 2k$ 5. Вырезаются клетки с числами от 1 до $k^2$

**Процесс шифрования**: 1. Решето накладывается на чистый квадрат 2. В прорези вписываются буквы текста 3. Решето поворачивается на 90° 3 раза 4. Буквы выписываются по столбцам согласно паролю

### Шифр Виженера

Опубликован Блезом Виженером в 1585 году. Считался нераскрываемым до 1863 года.

**Принцип работы**: 1. Текст разбивается на блоки длины $n$ 2. Ключ — последовательность из $n$ натуральных чисел 3. Каждая буква блока сдвигается на соответствующее число позиций 4. Ключом может служить осмысленное слово

**Математическая модель**: Для буквы открытого текста $P_i$ и ключа $K_i$:

$$C_i = (P_i + K_i) \mod m$$

где $m$ — мощность алфавита.

# Практическая реализация

## Реализация маршрутного шифрования

```
def route_cipher(text, password, m, n, fill_char='a'):

    text = text.replace(' ', '').upper()
    password = password.upper()

    block_size = m * n
    if len(text) % block_size != 0:
        padding = block_size - (len(text) % block_size)
        text += fill_char.upper() * padding

    password_order = sorted([(char, i) for i, char in enumerate(password)])
    column_order = [idx for char, idx in password_order]

    result = []

    for block_start in range(0, len(text), block_size):
        block = text[block_start:block_start + block_size]

        matrix = []
        for i in range(m):
            row_start = i * n
            row_end = row_start + n
            matrix.append(list(block[row_start:row_end]))

        for col_idx in column_order:
            for row in range(m):
                result.append(matrix[row][col_idx])

    return ''.join(result)
```

```python
def route_decipher(ciphertext, password, m, n):

    password = password.upper()
    block_size = m * n

    password_order = sorted([(char, i) for i, char in enumerate(password)])
    column_order = [idx for char, idx in password_order]

    inverse_order = [0] * n
    for new_pos, old_pos in enumerate(column_order):
        inverse_order[old_pos] = new_pos

    result = []

    for block_start in range(0, len(ciphertext), block_size):
        block = ciphertext[block_start:block_start + block_size]

        matrix = [[''] * n for _ in range(m)]

        idx = 0
        for col_idx in column_order:
            for row in range(m):
                if idx < len(block):
                    matrix[row][col_idx] = block[idx]
                    idx += 1

        for row in range(m):
            for col in range(n):
                result.append(matrix[row][col])

    return ''.join(result)
```

**Тестирование функции:**

```
test_text = "нельзя недооценивать противника"
password_route = "пароль"
password_grid = "шифр"
password_vigenere = "математика"
```

```
encrypted_route = route_cipher(test_text, password_route, 5, 6)
decrypted_route = route_decipher(encrypted_route, password_route, 5, 6)
```

```
print(f"После шифрования: {encrypted_route}")
print(f"После расшифровки: {decrypted_route}")
print()
```

```
После шифрования: ЕЕНПНЗОАТАЬОВОКННЕЬВЛДИРИЯЦТИА
После расшифровки: НЕЛЬЗЯНЕДООЦЕНИВАТЬПРОТИВНИКАА
```

**Реализация шифра с помощью решеток**

```
def create_grid(k):

    base_grid = [[(i * k + j + 1) for j in range(k)] for i in range(k)]

    full_grid = [[0] * (2*k) for _ in range(2*k)]

    positions = []

    for i in range(k):
        for j in range(k):
            positions.append((i, j, base_grid[i][j]))

    for i in range(k):
        for j in range(k):
            positions.append((j, k-1-i + k, base_grid[i][j]))

    for i in range(k):
        for j in range(k):
            positions.append((k-1-i + k, k-1-j + k, base_grid[i][j]))

    for i in range(k):
        for j in range(k):
            positions.append((k-1-j + k, i, base_grid[i][j]))

    return positions
```

```python
def grid_cipher(text, password, k, fill_char='a'):

    text = text.replace(' ', '').upper()
    password = password.upper()

    positions = create_grid(k)

    grid_size = 2 * k

    if len(text) < grid_size * grid_size:
        padding = grid_size * grid_size - len(text)
        text += fill_char.upper() * padding

    grid = [[''] * grid_size for _ in range(grid_size)]

    char_idx = 0
    for rotation in range(4):

        current_positions = [pos for pos in positions if 1 <= pos[2] <= k*k]

        current_positions.sort(key=lambda x: x[2])

        for i, j, num in current_positions:
            if char_idx < len(text):
                grid[i][j] = text[char_idx]
                char_idx += 1

    password_order = sorted([(char, i) for i, char in enumerate(password[:grid_size])])
    column_order = [idx for char, idx in password_order]

    result = []
    for col_idx in column_order:
        for row in range(grid_size):
            result.append(grid[row][col_idx])

    return ''.join(result)
```

**Реализация шифра Виженера**

```
def vigenere_cipher(text, key, alphabet=None):

    if alphabet is None:
        alphabet = "абвгдежзийклмнопрстуфхцчшщъыьэюя"

    text = text.replace(' ', '').upper()
    key = key.upper()

    table = []
    for i in range(len(alphabet)):
        row = alphabet[i:] + alphabet[:i]
        table.append([char.upper() for char in row])

    alphabet_upper = alphabet.upper()

    result = []
    key_idx = 0

    for char in text:
        if char in alphabet_upper:

            row_idx = alphabet_upper.index(char)

            key_char = key[key_idx % len(key)]
            col_idx = alphabet_upper.index(key_char)

            cipher_char = table[row_idx][col_idx]
            result.append(cipher_char)

            key_idx += 1
        else:
            result.append(char)

    return ''.join(result)
```

```python
def vigenere_decipher(ciphertext, key, alphabet=None):

    if alphabet is None:
        alphabet = "абвгдежзийклмнопрстуфхцчшщъыьэюя"

    key = key.upper()
    alphabet_upper = alphabet.upper()

    table = []
    for i in range(len(alphabet)):
        row = alphabet[i:] + alphabet[:i]
        table.append([char.upper() for char in row])

    result = []
    key_idx = 0

    for char in ciphertext:
        if char in alphabet_upper:

            key_char = key[key_idx % len(key)]
            col_idx = alphabet_upper.index(key_char)

            row_idx = -1
            for i in range(len(alphabet)):
                if table[i][col_idx] == char:
                    row_idx = i
                    break

            plain_char = alphabet_upper[row_idx]
            result.append(plain_char)

            key_idx += 1
        else:
            result.append(char)

    return ''.join(result)
```

**Тестирование функции:**

```
test_text2 = "криптография серьезная наука"
encrypted_vigenere = vigenere_cipher(test_text2, password_vigenere)
decrypted_vigenere = vigenere_decipher(encrypted_vigenere, password_vigenere)
```

```
print(f"После шифрования:: {encrypted_vigenere}")
print(f"После расшифровки: {decrypted_vigenere}")
```

```
После шифрования:: ЦРЪФЮОХШКФФЯГКЬЬЧПЧАЛНТШЦА
После расшифровки: КРИПТОГРАФИЯСЕРЬЕЗНАЯНАУКА
```

## Функциональное тестирование

| Исходный текст | Алгоритм | Ключ | Результат | Статус |
|---|---|---|---|---|
| “нельзя недооценивать противника” | Маршрутный | “пароль”, 5×6 | “ЕЕН-ПНЗО-АТАЬО-ВОКН-НЕЬВЯЦ-ТИА” | √ |
| “криптография серьезная наука” | Виженера | “математика” | “ЦРЬ-ФЯОХШКФ-ФЯДК-ЭЬЧП-ЧАЛНТШ-ЦА” | √ |
| “договор подписали” | Решетки | “шифр” | “ОВОРД-ЛГПА-ПИО-СДОИ” | √ |

# Анализ криптостойкости

## Уязвимости маршрутного шифрования

1. **Сохранение частотных характеристик** — перестановка не меняет состав символов
2. **Уязвимость к анаграммному анализу** — возможен подбор размеров таблицы
3. **Ограниченное пространство ключей** — зависит от размеров таблицы

## Уязвимости шифра решеток

1. **Сложность создания решетки** — требует предварительной подготовки
2. **Ограниченная гибкость** — размер решетки фиксирован
3. **Уязвимость к известной атаке** — при известном принципе построения решетки

Уязвимости шифра Виженера

1. **Периодичность ключа** — позволяет использовать метод Казиски
2. **Сохранение распределения** — каждая позиция имеет свой моноалфавитный шифр
3. **Уязвимость к частотному анализу** — при достаточной длине криптограммы

# Выводы

1. **Теоретические знания:** Изучены принципы работы шифров перестановки, их историческое развитие и математические основы.
2. **Практические навыки:** Реализованы три различных алгоритма шифрования перестановкой, проведено их функциональное тестирование.
3. **Аналитические способности:** Проанализированы криптографические слабости каждого метода, выявлены общие закономерности уязвимостей.
4. **Исторический контекст:** Рассмотрено практическое применение шифров в разные исторические периоды.
5. **Рекомендации:** Шифры перестановки, как и шифры замены, не обеспечивают достаточный уровень безопасности для современных требований, но важны для понимания эволюции криптографических методов.

# Литература

Криптографические методы защиты информации / под ред. В.А. Богданова. М.: Академия, 2018.

# Приложения

Полный исходный код программ доступен в репозитории GitHub: https://github.com/wangyao200036/cryptography-labs/tree/main/lab2