# MainPage/Info Security/Encript Lab 1.1

Университет ИТМО
Факультет ФПИ и КТ

## Отчёт по лабораторной работе 1

# «Основы шифрования данных»

по дисциплину

**«Информационная безопасность»**

Студент: Чжоу Хунсян
Группа: P33131
Преподаватель:

Санкт-Петербург 2024

# Цель работы

Изучение основных принципов шифрования информации, знакомство с широко известными алгоритмами шифрования, приобретение навыков их программной реализации.

# Варианты заданий

> Вариант: 23 % 10 = 3

Реализовать шифрование и дешифрацию файла с использованием метода биграмм. Ключевое слово вводится.

# Исходный код

```
# function for generating key matrix
def generate_key_matrix(key):
    alphabet = "ABCDEFGHIKLMNOPQRSTUVWXYZ"  # combine 'J' and 'I' together
    key = key.upper()
    key = ''.join(sorted(set(key.upper()), key=lambda x: key.index(x)))  # remove repeat letter
    key_matrix = []
    key_matrix_row = []
    used = set()

    for letter in key:
        if letter not in used and letter != 'J':  # ignore 'J'
            key_matrix_row.append(letter)
            used.add(letter)
        if key_matrix_row.__len__() == 5:
            key_matrix.append(key_matrix_row)
            key_matrix_row = []

    for letter in alphabet:
        if letter not in used:
            key_matrix_row.append(letter)
            used.add(letter)
        if key_matrix_row.__len__() == 5:
            key_matrix.append(key_matrix_row)
            key_matrix_row = []

    return key_matrix


# function to find position of letter in key matrix
def find_position(key_matrix, letter):
    for row in range(5):
        for col in range(5):
            if key_matrix[row][col] == letter.upper():
                return row, col


# function to judge whether a character is an english letter
def is_english_letter(char):
    return 'A' <= char <= 'Z' or 'a' <= char <= 'z'
```

```
# function to transform text pair matrix to string
def text_pairs_to_string(text_pairs):
    text = ""
    for text_pair in text_pairs:
        for char in text_pair:
            text += char
    return text


# function to print matrix
def print_matrix(matrix):
    for row in matrix:
        print(row)


# function to get english letters from pairs
def get_letter_from_pair(text_pair):
    pair_letter = []
    for letter in text_pair:
        if is_english_letter(letter):
            pair_letter.append(letter)
    return pair_letter


# function to preprocess_text
# preprocess
def preprocess_text(text):
    text = text.replace("J", "I").replace("j", "i")  # replace J with I and remove spaces
    processed_text_pairs = []
    pair = []
    pair_letter = []
    i = 0

    while i < len(text):
        pair.append(text[i])
        if is_english_letter(text[i]):
            pair_letter.append(text[i])
        if pair_letter.__len__() == 2:
            if pair_letter[0] == pair_letter[1]: # if there are two same letters in one pair, th
                pair.remove(pair_letter[1])
                pair_letter.remove(text[i])
                pair.append('X')
```

```
                processed_text_pairs.append(pair)
                pair_letter = []
                pair = []
                i -= 1
            else:
                processed_text_pairs.append(pair)
                pair_letter = []
                pair = []
        if i == len(text) - 1 and pair_letter.__len__() == 1: # if in the end the length of text
            pair.append('X')
            processed_text_pairs.append(pair)

        i += 1

    return processed_text_pairs


# encrypt process
def encrypt(text, key_matrix):
    encrypted_text_pairs = []
    preprocessed_text_pairs = preprocess_text(text)  # preprocess plaint text

    for preprocessed_text_pair in preprocessed_text_pairs:
        preprocessed_pair_letter = get_letter_from_pair(preprocessed_text_pair)
        row1, col1 = find_position(key_matrix, preprocessed_pair_letter[0])
        row2, col2 = find_position(key_matrix, preprocessed_pair_letter[1])

        encrypted_pair_letter = ['', '']
        if row1 == row2:  # if same row
            encrypted_pair_letter[0] = key_matrix[row1][(col1 + 1) % 5]
            encrypted_pair_letter[1] = key_matrix[row2][(col2 + 1) % 5]
        elif col1 == col2:  # if same col
            encrypted_pair_letter[0] = key_matrix[(row1 + 1) % 5][col1]
            encrypted_pair_letter[1] = key_matrix[(row2 + 1) % 5][col2]
        else:  # if square
            encrypted_pair_letter[0] = key_matrix[row1][col2]
            encrypted_pair_letter[1] = key_matrix[row2][col1]

        # make the capitalization the same as plaint text
        if preprocessed_pair_letter[0].islower():
            encrypted_pair_letter[0] = encrypted_pair_letter[0].lower()
        if preprocessed_pair_letter[1].islower():
            encrypted_pair_letter[1] = encrypted_pair_letter[1].lower()
```

```
        # set encrypted text pair matrix
        encrypted_text_pair = []
        letter_count = 0
        for letter in preprocessed_text_pair:
            if is_english_letter(letter):
                encrypted_text_pair.append(encrypted_pair_letter[letter_count])
                letter_count += 1
            else:
                encrypted_text_pair.append(letter)
        encrypted_text_pairs.append(encrypted_text_pair)

    return encrypted_text_pairs


# decrypt process
def decrypt(encrypted_text_pairs, key_matrix):
    decrypted_text_pairs = []

    for encrypted_text_pair in encrypted_text_pairs:
        encrypted_pair_letter = get_letter_from_pair(encrypted_text_pair)
        row1, col1 = find_position(key_matrix, encrypted_pair_letter[0])
        row2, col2 = find_position(key_matrix, encrypted_pair_letter[1])

        decrypted_pair_letter = ['', '']
        if row1 == row2:  # if same row
            decrypted_pair_letter[0] = key_matrix[row1][(col1 - 1) % 5]
            decrypted_pair_letter[1] = key_matrix[row2][(col2 - 1) % 5]
        elif col1 == col2:  # if same col
            decrypted_pair_letter[0] = key_matrix[(row1 - 1) % 5][col1]
            decrypted_pair_letter[1] = key_matrix[(row2 - 1) % 5][col2]
        else:  # if square
            decrypted_pair_letter[0] = key_matrix[row1][col2]
            decrypted_pair_letter[1] = key_matrix[row2][col1]

        if encrypted_pair_letter[0].islower():
            decrypted_pair_letter[0] = decrypted_pair_letter[0].lower()
        if encrypted_pair_letter[1].islower():
            decrypted_pair_letter[1] = decrypted_pair_letter[1].lower()

        decrypted_text_pair = []
        letter_count = 0
        for letter in encrypted_text_pair:
```

```
            if is_english_letter(letter):
                decrypted_text_pair.append(decrypted_pair_letter[letter_count])
                letter_count += 1
            else:
                decrypted_text_pair.append(letter)
        decrypted_text_pairs.append(decrypted_text_pair)

    decrypted_text_pairs = remove_x(decrypted_text_pairs)

    return decrypted_text_pairs


# function for removing additional 'X' in result during preprocess and encryption
def remove_x(decrypted_text_pairs):
    res_text_pairs = []
    for i in range(len(decrypted_text_pairs) - 1):
        text_pair = decrypted_text_pairs[i]
        text_pair_next = decrypted_text_pairs[i + 1]
        pair_letter = get_letter_from_pair(text_pair)
        pair_letter_next = get_letter_from_pair(text_pair_next)

        if pair_letter[0] == pair_letter_next[0] and pair_letter[1] == 'X':
            text_pair.remove('X')

        res_text_pairs.append(text_pair)

    if decrypted_text_pairs[-1][-1] == 'X':
        decrypted_text_pairs[-1].remove('X')
        res_text_pairs.append(decrypted_text_pairs[-1])

    return res_text_pairs


# read from file
def read_file(filename):
    with open(filename, 'r') as file:
        return file.read()


# write into file
def write_file(filename, content):
    with open(filename, 'w') as file:
        file.write(content)
```

```
# def compare_strings(str1, str2):
#     # 确保两个字符串的长度一致
#     if len(str1) != len(str2):
#         raise ValueError("String lengths do not match")
#
#     # 逐字符比较并统计差异字符数量
#     diff_count = 0
#     for char1, char2 in zip(str1, str2):
#         if char1 != char2:
#             diff_count += 1
#
#     return diff_count


# main function
if __name__ == "__main__":
    key_input = input("Please input keyword: ")
    # key_input = "SECRETKEY"
    matrix = generate_key_matrix(key_input)

    print("Generated key matrix: ")
    print_matrix(matrix)

    # 从read plain text from file
    plaintext = read_file('plaintext.txt')
    print(f"Plain text: \n{plaintext}")

    # encrypt text
    text_pairs_encrypted = encrypt(plaintext, matrix)
    write_file('encrypted.txt', text_pairs_to_string(text_pairs_encrypted))

    print(f"Encrypted text: \n{text_pairs_to_string(text_pairs_encrypted)}")

    # decrypt text
    text_pairs_decrypted = decrypt(text_pairs_encrypted, matrix)
    write_file('decrypted.txt', text_pairs_to_string(text_pairs_decrypted))

    print(f"Decrypted text: \n{text_pairs_to_string(text_pairs_decrypted)}")
```

# Результаты работы программы

Пример Usage

```
C:\Users\Tolia\AppData\Local\Programs\Python\Python310\python.exe "C:\Users\Tolia\Documents\GitH
Please input keyword: SECRETKEY
Generated key matrix:
['S', 'E', 'C', 'R', 'T']
['K', 'Y', 'A', 'B', 'D']
['F', 'G', 'H', 'I', 'L']
['M', 'N', 'O', 'P', 'Q']
['U', 'V', 'W', 'X', 'Z']
Plain text:
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut lab
Encrypted text:
Hqtcp fmrsu aqhqt elr kocs, awmecrscszc bblxpergpn yfls, ety aq rgsknpl dsnqpb poelblbvms zd qbc
Decrypted text:
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut lab

Process finished with exit code 0
```

# Вывод

В ходу лабораторной работы, я выучил несколько подходы для щифрования и дешифрования текст в файле.