

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Факультет безопасности информационных технологий

Дисциплина:

«Криптографические методы обеспечения информационной безопасности»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

«Анализ исторических шифров»

Выполнил:

Чу Ван Доан, студент группы N3347



(подпись)

Проверил:

Таранов Сергей Владимирович

(отметка о выполнении)

(подпись)

Санкт-Петербург

2025 г.

СОДЕРЖАНИЕ

Содержание.....	2
Введение.....	4
Задание.....	5
Ход работы.....	6
1. Шифр Цезаря.....	6
1.1. Принципы и основные характеристики.....	6
1.2. Формулы шифрования и дешифрования.....	6
1.2.1. Шифрование сообщения.....	7
1.2.2. Дешифрование сообщения.....	9
1.3. Взлом шифра.....	11
1.3.1. Анализ частоты.....	11
1.3.2. Использование известного фрагмента.....	11
1.3.3. Brute-force attack.....	12
2. Substitution Cipher (Шифр с заменой).....	13
2.1. Шифрование сообщения.....	13
2.2. Дешифрование сообщения.....	16
3. Transposition Cipher (Транспозиционный шифр).....	18
3.1. Принцип шифрования Columnar Transposition Cipher.....	19
3.2. Принцип дешифрования Columnar Transposition Cipher.....	21
3.3. Brute-force attack.....	24
3.4. Transposition Crib Analysis.....	27
3.5. Transposition Genetic Analysis.....	29
3.6. Transposition Hill Climbing Analysis.....	33
4. Vigenère Cipher.....	37
4.1. Метод повторения ключа (Key Repetition Method).....	38
4.2. Метод автоключа (Autokey Method).....	39
4.3. Vigenère Encryption (Key Repetition Method).....	39
4.4. Vigenère Decryption (Key Repetition Method).....	40
4.5. Kasiski Examination.....	42
5. Роторная машина Энигма (Enigma Cipher Machine).....	44
5.1. Аппаратная структура Enigma.....	44
5.2. Как шифруется каждая буква.....	45
5.3. Encryption.....	46
5.4. Decryption.....	49
5.5. Turing Bombe.....	52
5.5.1. Цель Bombe.....	53
5.5.2. Как работает Bombe.....	53
Заключение.....	56

ВВЕДЕНИЕ

Цель работы – изучить принципы работы исторических шифров, а также провести их криптоанализ.

Задание

Проанализировать следующие криптографические примитивы:

- 1) Шифр Цезаря, шифры перестановки и замены (как примеры моноалфавитных шифров);
- 2) Шифр Виженера (как пример полиалфавитного шифра);
- 3) Структуру и процесс шифрования в роторной машине Энигма.

1. Шифр Цезаря

1.1. Принципы и основные характеристики

Шифр Цезаря — один из самых известных и простых древних шифров. Принцип шифра Цезаря заключается в сдвиге каждой буквы открытого текста на фиксированное количество позиций в алфавите, после чего заменяется соответствующей буквой. Величина сдвига называется ключом (key) системы. Этот метод получил название в честь Юлия Цезаря, который использовал его для защиты своих секретных посланий. Например, при ключе = 3 (сдвиг на 3 буквы) буква А становится D, В становится Е, С становится F и так далее. Обратный процесс работает аналогично: Х становится U, Y становится V, Z становится W (циклический сдвиг).

Шифр Цезаря фактически представляет собой частный случай шифра подстановки, в котором алфавит шифротекста — это обычный алфавит, но сдвинутый на фиксированное число позиций.

1.2. Формулы шифрования и дешифрования

Шифрование в шифре Цезаря можно описать математически через порядковый номер буквы в алфавите.

Сначала каждой букве присваивается числовое значение:

$$A = 0, B = 1, C = 2, \dots, Z = 25.$$

Пусть X — это числовое значение буквы открытого текста, Y — числовое значение зашифрованной буквы, а k — ключ (сдвиг). Тогда формула шифрования выглядит так:

$$Y = (X + k) \bmod 26$$

Здесь сложение выполняется по модулю 26 (количество букв в алфавите).

Процесс дешифрования выполняется по обратной формуле:

$$X = (Y - k) \bmod 26$$

Иными словами, расшифровка означает сдвиг буквы назад на k позиций в алфавите.

Например, если $k = 3$, то правило шифрования:

$$A \rightarrow D, B \rightarrow E, \dots, X \rightarrow A, Y \rightarrow B, Z \rightarrow C.$$

А для расшифрования:

$D \rightarrow A, E \rightarrow B, \dots, A \rightarrow X, B \rightarrow Y, C \rightarrow Z$.

Иллюстрация шифра Цезаря с шагом сдвига 3. Верхний ряд — это исходный алфавит (открытый текст), а нижний ряд — это алфавит, сдвинутый влево на 3 позиции (Получаем закрытый текст:).

Стрелки показывают, как каждая буква открытого текста заменяется на букву, находящуюся на 3 позиции левее.

Например, буква **Е** (выделена синим) в верхнем ряду шифруется в **В** (выделена синим) в нижнем ряду при использовании ключа 3.

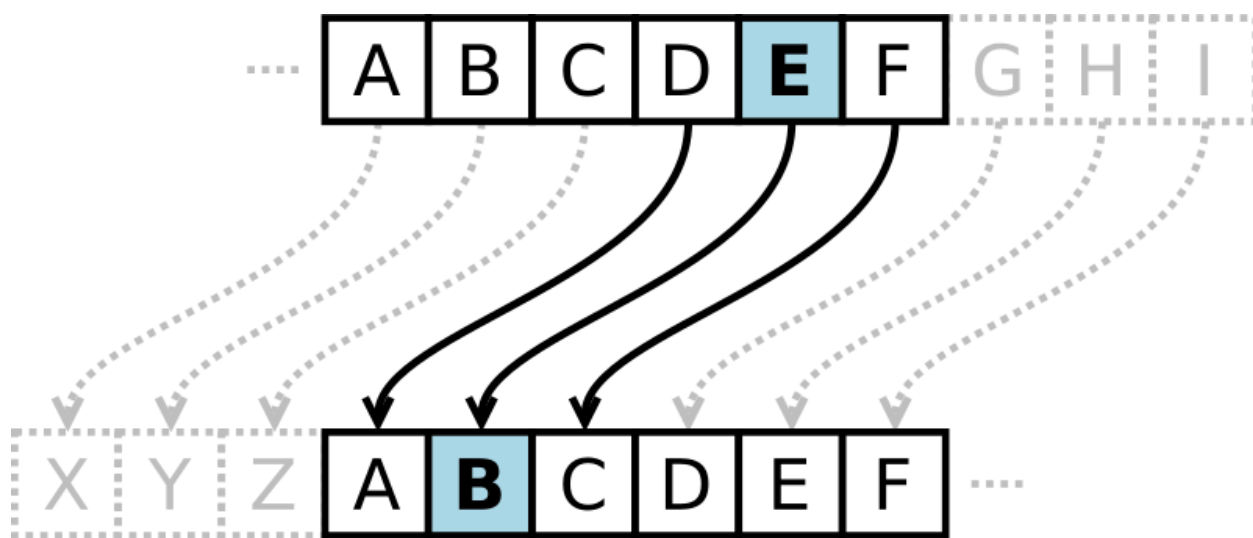


Рисунок 1 – Шифрование текста, шифр Цезаря

1.2.1. Шифрование сообщения

Исходное сообщение: "HELLO"

Сдвиговой шифр с ключом $k = 3$ (Каждая буква заменяется буквой, находящейся на 3 позиции дальше в алфавите):

Открытый текст	Н	Е	Л	Л	О
Шифротекст	К	Н	О	О	Р

Итоговое зашифрованное сообщение: "KHOOR"

- Открытый текст:

Is anyone as foolish as me? I create my own love story, only to end up hurting myself. Is anyone as foolish as me? Ignoring hundreds of paths, I choose to look in only one direction. I wish for just one chance to love you, to be near you, even if it means enduring pain because that is still better than a lifetime of regret. I would rather be a flower spreading its fragrance for you to pick than to wither away unnoticed. That day, the sunlight shimmered so beautifully, and I couldn't help but be captivated by your clear eyes. That day, the rain fell, stinging my eyes, and yet, you stood beside me, warming my heart. Even if you have your own path to follow, I still wish to be the umbrella that shields you from the storm. Tomorrow, I will still love you, but I will keep it locked deep inside, a secret only I will ever know.

- Код:

```
def caesar_encrypt(text, key):
    encrypted = ""
    for char in text:
        if char.isalpha():
            shift = key % 26
            base = ord('A') if char.isupper() else ord('a')
            encrypted += chr((ord(char) - base + shift) % 26 + base)
        else:
            encrypted += char
    return encrypted

def encrypt_file(input_filename, output_filename, key):
    try:
        with open(input_filename, 'r', encoding='utf-8') as infile:
            plaintext = infile.read()

        ciphertext = caesar_encrypt(plaintext, key)

        with open(output_filename, 'w', encoding='utf-8') as outfile:
            outfile.write(ciphertext)

        print(f"Encryption complete. Result saved in '{output_filename}'.")

    except FileNotFoundError:
        print(f"File '{input_filename}' not found.")
    except Exception as e:
        print(f"Error: {e}")

# =====
# Main program
```

```
# =====
if __name__ == "__main__":
    input_file = "input.txt"
    output_file = "caesar_encryption.txt"

    try:
        key = int(input("Enter Caesar key (integer): "))
        encrypt_file(input_file, output_file, key)
    except ValueError:
        print("Please enter a valid integer.")
```

- Получаем закрытый текст:

Lv dqbrqh dv irrolvk dv ph? L fuhdwh pb rzq oryh vwrub, rqob wr hqg xs kxuwqlj pbvhoi. Lv dqbrqh dv irrolvk dv ph? Ljqrulqj kxqguhgv ri sdwkv, L fkrvh wr orrn lq rqob rqh gluhfwlrq. L zlvk iru mxvw rqh fkdqfh wr oryh brx, wr eh qhdu brx, hyhq li lw phdqv hqgxulqj sdlq ehfdxvh wkdw lv vwloo ehwwhu wkdw d olihwlp ri uhjuhw. L zrxog udwkhuh eh d iorzhu vsuhdglqj lww iudjudqfh iru brx wr slfn wkdw wr zlwkhuh dzdb xqqrwlfhg. Wkdw gdb, wkh vxqoljkw vklppuhg vr ehdxwlixoob, dqg L frxogq'w khos exw eh fdswlydwhg eb brxu fohdu hbhv. Wkdw gdb, wkh udlq ihoo, vwlqjlqj pb hbhv, dqg bhw, brx vwrrg ehvlg ph, zduplqj pb khduw. Hyhq li brx kdyh brxu rzq sdwk wr iroorz, L vwloo zlvk wr eh wkh xpeuhood wkdw vklhogv brx iurp wkh vwrub. Wrpruurz, L zloo vwloo oryh brx, exw L zloo nhhs lw orfnhg ghhs lqvlgh, d vhfuhw rqob L zloo hyhu nqrz.

1.2.2. Дешифрование сообщения

Чтобы расшифровать, выполняем обратный сдвиг ($k = -3$):

Шифротекст	K	H	O	O	R
Открытый текст	H	E	L	L	O

Восстановленный открытый текст: "HELLO"

- Дешифрование текста:

Lv dqbrqh dv irrolvk dv ph? L fuhdwh pb rzq oryh vwrub, rqob wr hqg xs kxuwqlj pbvhoi. Lv dqbrqh dv irrolvk dv ph? Ljqrulqj kxqguhgv ri sdwkv, L fkrvh wr orrn lq rqob rqh gluhfwlrq. L zlvk iru mxvw rqh fkdqfh wr oryh brx, wr eh qhdu brx, hyhq li lw phdqv hqgxulqj sdlq ehfdxvh

wkdw lv vwloo ehwwhu wkdq d olihwlpb ri uhjuhw. L zrxog udwkhu eh d iorzhu vsuhdglqj lwv iudjudqfh iru brx wr slfn wkdq wr zlwkh dzdb xqqrwlfhg. Wkdw gdb, wkh vxqoljkw vklppuhg vr ehdxwlxoob, dqg L frxogq'w khos exw eh fdswlydwhg eb brxu fohdu hbhv. Wkdw gdb, wkh udlq ihoo, vwlqjlqj pb hbhv, dqg bhw, brx vwrrg ehvlg ph, zduplqj pb khduw. Hyhq li brx kdyh brxu rzq sdwk wr iroorz, L vwloo zlvk wr eh wkh xpeuhod wkdw vklhogv brx iurp wkh vwrup. Wrpruurz, L zloo vwloo oryh brx, exw L zloo nhhs lw orfnhg ghhs lqvlgh, d vhfuhw rqob L zloo hyhu nqrz.

```
- Код:
def caesar_decrypt(text, key):
    decrypted = ""
    for char in text:
        if char.isalpha():
            shift = key % 26
            base = ord('A') if char.isupper() else ord('a')
            decrypted += chr((ord(char) - base - shift) % 26 + base)
        else:
            decrypted += char
    return decrypted

def decrypt_file(input_filename, output_filename, key):
    try:
        with open(input_filename, 'r', encoding='utf-8') as infile:
            ciphertext = infile.read()

            plaintext = caesar_decrypt(ciphertext, key)

            with open(output_filename, 'w', encoding='utf-8') as outfile:
                outfile.write(plaintext)

            print(f"Decryption complete. Result saved in '{output_filename}'.")

    except FileNotFoundError:
        print(f"File '{input_filename}' not found.")
    except Exception as e:
        print(f"Error: {e}")

# =====
# Main program
# =====
if __name__ == "__main__":
    input_file = "caesar_encryption.txt"
```

```

output_file = "caesar_decryption.txt"

try:
    key = int(input("Enter Caesar decryption key (integer): "))
    decrypt_file(input_file, output_file, key)
except ValueError:
    print("Please enter a valid integer.")

```

- Текст был расшифрован

Is anyone as foolish as me? I create my own love story, only to end up hurting myself. Is anyone as foolish as me? Ignoring hundreds of paths, I choose to look in only one direction. I wish for just one chance to love you, to be near you, even if it means enduring pain because that is still better than a lifetime of regret. I would rather be a flower spreading its fragrance for you to pick than to wither away unnoticed. That day, the sunlight shimmered so beautifully, and I couldn't help but be captivated by your clear eyes. That day, the rain fell, stinging my eyes, and yet, you stood beside me, warming my heart. Even if you have your own path to follow, I still wish to be the umbrella that shields you from the storm. Tomorrow, I will still love you, but I will keep it locked deep inside, a secret only I will ever know.

1.3. Взлом шифра

1.3.1. Анализ частоты

Шифр Цезаря не изменяет частоту появления букв, а только заменяет их значениями. В английском языке:

- E — самая часто встречающаяся буква.
- T, A, O, I, N тоже довольно распространены.

Предположим, что в зашифрованном тексте встречается "KHOOR". Если у нас есть длинный текст, можно подсчитать частоту букв и заметить:

- O встречается дважды → возможно, это одна из распространённых букв, например, E или L.
- H, K, R встречаются реже.

Если предположить, что "O" соответствует "L", то можно попробовать сдвинуть текст на 3 позиции назад, что даст слово "HELLO".

1.3.2. Использование известного фрагмента

Если заранее известно, что в сообщении содержится "HELLO", можно сравнить его с "KHOOR" и заметить закономерность:

- H → K (сдвиг на 3)

- $E \rightarrow H$ (сдвиг на 3)
- $L \rightarrow O$ (сдвиг на 3)
- $L \rightarrow O$ (сдвиг на 3)
- $O \rightarrow R$ (сдвиг на 3)

Определив шаблон сдвига, можно заключить, что ключ $k = 3$, и расшифровать остальной текст.

1.3.3. Brute-force attack

Brute-force (атака перебором) — это метод перебора всех возможных ключей до тех пор, пока не будет найден осмысленный открытый текст.

Мы последовательно пробуем все ключи от 1 до 25. Для каждого ключа расшифровываем шифротекст, сдвигая буквы назад на соответствующее количество шагов.

Поскольку количество возможных ключей составляет всего 25, этого недостаточно, чтобы создать серьёзные трудности. Поэтому шифр Цезаря не защищён от атаки перебором (brute-force).

- Код:

```
def caesar_decrypt(text, key):
    decrypted = ""
    for char in text:
        if char.isalpha():
            shift = key % 26
            base = ord('A') if char.isupper() else ord('a')
            decrypted += chr((ord(char) - base - shift) % 26 + base)
        else:
            decrypted += char
    return decrypted

def brute_force_attack(input_filename):
    try:
        with open(input_filename, 'r', encoding='utf-8') as infile:
            ciphertext = infile.read()

        print("Brute-force Caesar Cipher results:\n")
        for key in range(1, 26):
            decrypted = caesar_decrypt(ciphertext, key)
            print(f"[Key {key:2}] {decrypted}")

    except FileNotFoundError:
        print(f"File '{input_filename}' not found.")
```

```

except Exception as e:
    print(f"Error: {e}")

# =====
# Main program
# =====

if __name__ == "__main__":
    input_file = "caesar_encryption.txt"
    brute_force_attack(input_file)

```

[Key 3] Is anyone as foolish as me? I create my own love story, only to end up hurting myself. Is anyone as foolish as me? Ignoring hundreds of paths, I choose to look in only one direction. I wish for just one chance to love you, to be near you, even if it means enduring pain because that is still better than a lifetime of regret. I would rather be a flower spreading its fragrance for you to pick than to wither away unnoticed. That day, the sunlight shimmered so beautifully, and I couldn't help but be captivated by your clear eyes. That day, the rain fell, stinging my eyes, and yet, you stood beside me, warming my heart. Even if you have your own path to follow, I still wish to be the umbrella that shields you from the storm. Tomorrow, I will still love you, but I will keep it locked deep inside, a secret only I will ever know.

Рисунок 2 – Результат Brute-force attack

2. Substitution Cipher (Шифр с заменой)

Каждый символ открытого текста (plaintext) заменяется другим символом по определённому правилу (таблице подстановки).

Сохраняется длина строки, и положения символов не изменяются.

2.1. Шифрование сообщения

Каждая буква открытого текста заменяется единственной другой буквой.

Используется таблица замены (substitution table) для отображения каждой буквы.

Это взаимнооднозначное отображение (биекция): каждая буква заменяется только одной уникальной буквой, без повторов.

Предположим, используется следующая таблица замены:

Исходная буква	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Заменённая	Q	W	E	R	T	Y	U	I	O	P	A	S	D	F	G	H	J	K	L	Z	X	C	V	B	N	M

Это случайная перестановка алфавита.

Исходный текст: HELLO

Шифруем по буквам:

- $H \rightarrow I$
- $E \rightarrow T$
- $L \rightarrow S$
- $L \rightarrow S$
- $O \rightarrow G$

Результат шифрования: ITSSG

- Открытый текст:

Is anyone as foolish as me? I create my own love story, only to end up hurting myself. Is anyone as foolish as me? Ignoring hundreds of paths, I choose to look in only one direction. I wish for just one chance to love you, to be near you, even if it means enduring pain because that is still better than a lifetime of regret. I would rather be a flower spreading its fragrance for you to pick than to wither away unnoticed. That day, the sunlight shimmered so beautifully, and I couldn't help but be captivated by your clear eyes. That day, the rain fell, stinging my eyes, and yet, you stood beside me, warming my heart. Even if you have your own path to follow, I still wish to be the umbrella that shields you from the storm. Tomorrow, I will still love you, but I will keep it locked deep inside, a secret only I will ever know.

- Key: IJKLMNOPQRSTUVWXYZABCDEFGH

- Код:

```
def load_substitution_key(filename):
    try:
        with open(filename, 'r', encoding='utf-8') as f:
            line = f.readline().strip().upper()
            if len(line) != 26 or not line.isalpha():
                raise ValueError("Invalid key! Must be 26 unique letters.")
            return line
    except FileNotFoundError:
        print(f"File '{filename}' not found.")
        return None
    except Exception as e:
        print(f"Error reading key: {e}")
        return None

def substitution_encrypt(text, key_map):
    result = ""
    for char in text:
        if char.isalpha():
            is_upper = char.isupper()
            idx = ord(char.upper()) - ord('A')
```

```

        sub_char = key_map[idx]
        result += sub_char if is_upper else sub_char.lower()
    else:
        result += char
    return result

def encrypt_file(input_file, key_file, output_file):
    key_map = load_substitution_key(key_file)
    if not key_map:
        return

    try:
        with open(input_file, 'r', encoding='utf-8') as f:
            plaintext = f.read()

            ciphertext = substitution_encrypt(plaintext, key_map)

            with open(output_file, 'w', encoding='utf-8') as f:
                f.write(ciphertext)

            print(f"Encryption complete. Result saved in '{output_file}'.")

    except FileNotFoundError:
        print(f"File '{input_file}' not found.")
    except Exception as e:
        print(f"Error processing file: {e}")

# =====
# Main program
# =====
if __name__ == "__main__":
    input_file = "input.txt"
    key_file = "substitution_table_key.txt"
    output_file = "Substitution_Cipher_encryption.txt"

    encrypt_file(input_file, key_file, output_file)

```

- Получаем закрытый текст::

Qa ivgwvm ia nwwtqap ia um? Q kzmibm ug wev twdm abwzg, wvtg bw mvl cx pczbqvo ugamtn. Qa ivgwvm ia nwwtqap ia um? Qovwzqvo pcvlzmia wn xibpa, Q kpwwam bw twws qv wvtg wvm lqzmkbqvw. Q eqap nwz rcab wvm kpivkm bw twdm gwc, bw jm vmiz gwc, mdmv qn qb umiva mvlczqvo xiqv jmkicam bpib qa abqtt jmbbmz bpiv i tqnmbqum wn zmozmb. Q

ewctl zibpmz jm i ntwemz axzmlqvo qba nziozivkm nwz gwc bw xqks bpiv bw eqbpmz ieig cvvwbqkml. Bpib lig, bpm acvtqopb apquumzml aw jmicbqnettg, ivl Q kwctlv'b pmtx jcb jm kixbqdbml jg gwc ktmiz mgma. Bpib lig, bpm ziqv nmmt, abqvoqvo ug mgma, ivl gmb, gwc abwwl jmaqlm um, eizuqvo ug pmizb. Mdmv qn gwc pidm gwc wev xibp bw nwtwe, Q abqtt eqap bw jm bpm cujzmtti bpib apqmtla gwc nzwu bpm abwzu. Bwuwzzwe, Q eqtt abqtt twdm gwc, jcb Q eqtt smmx qb twksml lmmx qvaqlm, i amkzmb wvtg Q eqtt mdmz svwe.

2.2. Дешифрование сообщения

Используется обратная таблица замены: каждая зашифрованная буква сопоставляется с исходной.

Создаём таблицу расшифровки (обратную к таблице шифрования):

Зашифр. буква	Q	W	E	R	T	Y	U	I	O	P	A	S	D	F	G	H	J	K	L	Z	X	C	V	B	N	M
Исх. буква	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

Получаем закрытый текст:: ITSSG

Расшифровка по буквам:

- $I \rightarrow H$
- $T \rightarrow E$
- $S \rightarrow L$
- $S \rightarrow L$
- $G \rightarrow O$

Результат расшифровки: HELLO

- Дешифрование текста:

Qa ivgwvm ia nwwtqap ia um? Q kzmibm ug wev twdm abwzg, wvtg bw mvl cx pczbqvo ugamtn. Qa ivgwvm ia nwwtqap ia um? Qovwzqvo pcvlzmla wn xibpa, Q kpwwam bw twws qv wvtg wvm lqzmbqvw. Q eqap nwz rcab wvm kpivkm bw twdm gwc, bw jm vmiz gwc, mdmv qn qb umiva mvlczqvo xiqv jmkicam bpib qa abqtt jmbbmz bpiv i tqnmbqum wn zmozmb. Q ewctl zibpmz jm i ntwemz axzmlqvo qba nziozivkm nwz gwc bw xqks bpiv bw eqbpmz ieig cvvwbqkml. Bpib lig, bpm acvtqopb apquumzml aw jmicbqnettg, ivl Q kwctlv'b pmtx jcb jm kixbqdbml jg gwc ktmiz mgma. Bpib lig, bpm ziqv nmmt, abqvoqvo ug mgma, ivl gmb, gwc abwwl jmaqlm um, eizuqvo ug pmizb. Mdmv qn gwc pidm gwc wev xibp bw nwtwe, Q abqtt

eqap bw jm bpm cujzmtti bpib apqmtla gwc nzwu bpm abwzu. Bwuwzzwe, Q eqtt abqtt twdm
gwc, jcb Q eqtt smmx qb twksml lmmx qvaqlm, i amkzmb wvtg Q eqtt mdmz svwe.

- Key: IJKLMNOPQRSTUVWXYZABCDEFGH

- Код:

```
def load_substitution_key(filename):
    try:
        with open(filename, 'r', encoding='utf-8') as f:
            line = f.readline().strip().upper()
            if len(line) != 26 or not line.isalpha():
                raise ValueError("Invalid key! Must be 26 unique letters.")
            return line
    except FileNotFoundError:
        print(f"File '{filename}' not found.")
        return None
    except Exception as e:
        print(f"Error reading key: {e}")
        return None

def build_reverse_key_map(key_map):
    reverse_map = [''] * 26
    for i, char in enumerate(key_map):
        idx = ord(char) - ord('A')
        reverse_map[idx] = chr(ord('A') + i)
    return reverse_map

def substitution_decrypt(text, reverse_map):
    result = ""
    for char in text:
        if char.isalpha():
            is_upper = char.isupper()
            idx = ord(char.upper()) - ord('A')
            plain_char = reverse_map[idx]
            result += plain_char if is_upper else plain_char.lower()
        else:
            result += char
    return result

def decrypt_file(input_file, key_file, output_file):
    key_map = load_substitution_key(key_file)
    if not key_map:
        return

    reverse_map = build_reverse_key_map(key_map)
```



```

try:
    with open(input_file, 'r', encoding='utf-8') as f:
        ciphertext = f.read()

    plaintext = substitution_decrypt(ciphertext, reverse_map)

    with open(output_file, 'w', encoding='utf-8') as f:
        f.write(plaintext)

    print(f"Decryption complete. Result saved in '{output_file}'.")

except FileNotFoundError:
    print(f"File '{input_file}' not found.")
except Exception as e:
    print(f"Error processing file: {e}")

# =====
# Main program
# =====
if __name__ == "__main__":
    input_file = "Substitution_Cipher_encryption.txt"
    key_file = "substitution_table_key.txt"
    output_file = "Substitution_Cipher_decryption.txt"

    decrypt_file(input_file, key_file, output_file)

```

```

< chu >> cat Substitution_Cipher_decryption.txt
Is anyone as foolish as me? I create my own love story, only to end up hurting myself. Is anyone as foolish as me? Ignoring hun
dreds of paths, I choose to look in only one direction. I wish for just one chance to love you, to be near you, even if it mean
s enduring pain because that is still better than a lifetime of regret. I would rather be a flower spreading its fragrance for
you to pick than to wither away unnoticed. That day, the sunlight shimmered so beautifully, and I couldn't help but be captiva
ted by your clear eyes. That day, the rain fell, stinging my eyes, and yet, you stood beside me, warming my heart. Even if you
have your own path to follow, I still wish to be the umbrella that shields you from the storm. Tomorrow, I will still love you,
but I will keep it locked deep inside, a secret only I will ever know.

```

Рисунок 3 – Результат

3. Transposition Cipher (Транспозиционный шифр)

Transposition Cipher - это метод шифрования, при котором порядок символов в открытом тексте изменяется по определённому правилу (обычно на основе ключа), но сами символы не заменяются.

Columnar Transposition Cipher - это самая распространённая разновидность шифра перестановки и классический пример, часто используемый в курсах по криптографии.

Columnar Transposition Cipher шифрует сообщение следующим образом:

- Открытый текст записывается в виде таблицы по строкам.
- Затем столбцы упорядочиваются в соответствии с порядком, заданным ключом.
- Наконец, символы считываются по столбцам в новом порядке, чтобы сформировать шифротекст.

3.1. Принцип шифрования Columnar Transposition Cipher

Исходный текст: "WE ARE DISCOVERED RUN"

Ключ: "4312567"

Это означает, что есть 7 столбцов.

Исходная таблица:

1	2	3	4	5	6	7
W	E	A	R	E	D	I
S	C	O	V	E	R	E
D	R	U	N	-	-	-

Затем мы считываем символы по столбцам в порядке, заданном ключом, чтобы зашифровать сообщение.

4	3	1	2	5	6	7
R	A	W	E	E	D	I
V	O	S	C	E	R	E
N	U	D	R	-	-	-

В результате получаем шифротекст: RVNAOUWSD ECR EE DR IE

- Открытый текст:

Is anyone as foolish as me? I create my own love story, only to end up hurting myself. Is anyone as foolish as me? Ignoring hundreds of paths, I choose to look in only one direction. I wish for just one chance to love you, to be near you, even if it means enduring pain because that is still better than a lifetime of regret. I would rather be a flower spreading its fragrance for you to pick than to wither away unnoticed. That day, the sunlight shimmered so beautifully, and I couldn't help but be captivated by your clear eyes. That day, the rain fell, stinging my eyes, and yet, you stood beside me, warming my heart. Even if you have your own path to follow, I still wish to be

the umbrella that shields you from the storm. Tomorrow, I will still love you, but I will keep it locked deep inside, a secret only I will ever know.

- Key: 4213

- Код:

```
import math

# Read plaintext from input.txt
def load_plaintext(filename="input.txt"):
    with open(filename, 'r', encoding='utf-8') as f:
        return f.read().replace('\n', ' ').replace(' ', '').upper()

# Read key from Columnar_Transposition_Cipher_key.txt
def load_key(filename="Columnar_Transposition_Cipher_key.txt"):
    with open(filename, 'r', encoding='utf-8') as f:
        key_line = f.read().strip()
        if ' ' in key_line:
            key = list(map(int, key_line.split()))
        else:
            key = [int(c) for c in key_line]
        return key

# Columnar Transposition Cipher encryption
def columnar_encrypt(plaintext, key):
    num_cols = len(key)
    num_rows = math.ceil(len(plaintext) / num_cols)

    # Add padding if needed
    padded_len = num_cols * num_rows
    plaintext += 'X' * (padded_len - len(plaintext))

    # Create grid row by row
    grid = []
    index = 0
    for _ in range(num_rows):
        grid.append(plaintext[index:index + num_cols])
        index += num_cols

    # Read columns in sorted key order
    ciphertext = ''
    sorted_key = sorted((num, idx) for idx, num in enumerate(key))
    for _, col_idx in sorted_key:
        for row in grid:
            ciphertext += row[col_idx]
```

```

    return ciphertext

# Save ciphertext to file
def save_ciphertext(ciphertext,
filename="Columnar_Transposition_Cipher_encryption.txt"):
    with open(filename, 'w', encoding='utf-8') as f:
        f.write(ciphertext)

# =====
# Main program execution
# =====
if __name__ == "__main__":
    plaintext = load_plaintext("input.txt")
    key = load_key("Columnar_Transposition_Cipher_key.txt")
    ciphertext = columnar_encrypt(plaintext, key)
    save_ciphertext(ciphertext,
"Columnar_Transposition_Cipher_encryption.txt")
    print("Encryption complete. Result saved in
'Columnar_Transposition_Cipher_encryption.txt'")

```

- Получаем закрытый текст::

ANFISIAYLSYLEPTMLSOSLA?OGDSA,OTOOOIT.SRTCCLYTNYEIMSUGNATHIERNFM
REWDHEOSAGFREYOKNIRYOEHANHNIROUU,ILTPBPABUEYTDTAESGMENTUOSM
ANHTEYAOWTFOSLHEUETSLOOERORILLVUTLELEES,COILRWSOSLA?EMNERNOU
RGEIYAOHENNNDPSHEONYDCNIOSENOE,ER,NTNDNICETTBEAIIFRILTLRENSGCR
TCAWEANCTDTUGHESAFYDU'LTAVDOLE.T,RF,NGYAEOOEEWIYRVFHYOAOLILSB
ERATEYRHOTR,LIOOUIETKENEETYLEONEOSMCTOOT,YNHIYFANFISIRHROTIOOK
NNRIIHJOHEOOEOVFEEERPBUHSLTTAAEETOREAWPDIRAFOPTTTAUTDAYELTMEB
TLACDHBETTYRAEHAHILTIYSD,SDIERGE.NOVUNHOWTWTTMLHHDUMSMMOWS
LE,ILPODPIARNWEK.IYAOHEREWVOOTDUNS.NEOSMGIUEFHCSLILEEOWFUNATV
UBAUEIANIAESASLTHLTOG.UARFERITANOUIHOWNI.T,SISMDEILNONEUCIEYCRS
AYENLINE,YYTBD,MMAEIUERPTL,IIOHBLAISFTT.OWITLYBWKICDIDSELIVN

3.2. Принцип дешифрования Columnar Transposition Cipher

Входные данные:

Шифротекст: RVNAOUWSDECREEDRIE

Ключ: "4312567"

Вычисление количества строк

Шифротекст содержит 18 символов.

Количество столбцов = длина ключа = 7

Количество строк = $\lceil 18 / 7 \rceil = 3$

Разделите шифротекст на столбцы в соответствии с порядком, заданным ключом.

4	R	V	N
3	A	O	U
1	W	S	D
2	E	C	R
5	E	E	-
6	D	R	-
7	I	E	-

Считываем построчно → Исходный текст

1	2	3	4	5	6	7
W	E	A	R	E	D	I
S	C	O	V	E	R	E
D	R	U	N	-	-	-

Исходный текст: WE ARE DISCOVERED RUN

- Дешифрование текста:

ANFISIAYLSYLEPTMLSOSLA?OGDSA,OTOOOIT.SRTCCLYTNYEIMSUGNATIERNFM
REWDHEOSAGFREYOKNIRYOEHAAHNIROUU,ILTPBPABUEYTDTAESGMENTUOSM
ANHTEYAOWTFOSLHEUETSLOOERORILLVUTLELEES,COILRWSOSLA?EMNERNOU
RGEIYAOHENNNDPSHEONYDCNIOSENOE,ER,NTNDNICETTBEAIIFRILTLRENSGCR
TCAWEANCTDTUGHESAFYDU'LTAVDOLE.T,RF,NGYAEOOEEWIYRVFHYOAOLILSB
ERATEYRHOTR,LIOOUIETKENEETYLEONEOSMCTOOT,YNHIYFANFISIRHROTIOOK
NNRIIHJOHEOOEOVFEEERPBUHSLTTAEETOREAWPDIRAFOPTTTAUTDAYELTMEB
TLACDHBETTYRAEHAHILTIYSD,SDIERGE.NOVUNHOWTWTTMLHHDUMSMMOWS
LE,ILPODPIARNWEK.IYAOHEREWVOOTDUNS.NEOSMGIUEFHCSLILEEOWFUNATV
UBAUEIANIAESASLTHLTOG.UARFERITANOUIHOWNI.T,SISMDEILNONEUCIEYCRS
AYENLINE,YYTBD,MMAEIUERPTL,IIOHBLAISFTT.OWITLYBWKICDIDSELIVN

- Key: 4213
- Код:

```
import math
```

```
# Read ciphertext from file
```

```
def
```

```
load_ciphertext(filename="Columnar_Transposition_Cipher_encryption.txt"):
```

```

with open(filename, 'r', encoding='utf-8') as f:
    return f.read().replace('\n', '').strip()

# Read key from file
def load_key(filename="Columnar_Transposition_Cipher_key.txt"):
    with open(filename, 'r', encoding='utf-8') as f:
        key_line = f.read().strip()
        if ' ' in key_line:
            key = list(map(int, key_line.split()))
        else:
            key = [int(c) for c in key_line]
    return key

# Columnar Transposition Cipher decryption
def columnar_decrypt(ciphertext, key):
    num_cols = len(key)
    num_rows = math.ceil(len(ciphertext) / num_cols)

    # Calculate total number of cells and padding
    total_cells = num_cols * num_rows
    padding = total_cells - len(ciphertext)

    # Determine the length of each column
    col_lengths = [num_rows] * num_cols
    sorted_key = sorted((num, idx) for idx, num in enumerate(key))
    for i in range(padding):
        # The last columns in sorted_key will have one character less
        col_num = sorted_key[-(i + 1)][1]
        col_lengths[col_num] -= 1

    # Slice the ciphertext into columns based on the correct order
    cols = [''] * num_cols
    index = 0
    for num, idx in sorted_key:
        length = col_lengths[idx]
        cols[idx] = ciphertext[index:index + length]
        index += length

    # Read row by row to reconstruct plaintext
    plaintext = ''
    for row in range(num_rows):
        for col in range(num_cols):
            if row < len(cols[col]):

```

```

        plaintext += cols[col][row]

    return plaintext

# Save decrypted plaintext to file
def save_plaintext(plaintext,
filename="Columnar_Transposition_Cipher_Decryption.txt"):
    with open(filename, 'w', encoding='utf-8') as f:
        f.write(plaintext)

# =====
# Main program execution
# =====
if __name__ == "__main__":
    ciphertext =
load_ciphertext("Columnar_Transposition_Cipher_encryption.txt")
    key = load_key("Columnar_Transposition_Cipher_key.txt")
    plaintext = columnar_decrypt(ciphertext, key)
    save_plaintext(plaintext,
"Columnar_Transposition_Cipher_Decryption.txt")
    print("Decryption complete. Result saved in
'Columnar_Transposition_Cipher_Decryption.txt'")

$ chu >> cat Columnar_Transposition_Cipher_Decryption.txt
ISANYONEASFOOLISHASME?ICREATEMYOWNLOVESTORY,ONLYTOENDUPHURTINGMYSELF.ISANYONEASFOOLISHASME?IGNORINGHUNDREDSOFPATHS,ICHOOSETOLOO
KINONLYONEDIRECTION.IWISHFORJUSTONECHANCETOLOVEYOU,TOBENEARYOU,EVENIFITMEANSENDURINGPAINBECAUSETHATISSTILLBETTERTHANALIFETIMEOF
REGRET.IWOULDRATHERBEAFLOWERSPREADINGITSFRAGRANCEFORYOUTOPICKTHANTOWITHERAWAYUNNOTICED.THATDAY,THESUNLIGHTSHIMMEREDSOBEAUTIFULL
Y,ANDICOULDN'THELPBUTBECAPTIVATEDBYYOURCLEAREYES.THATDAY,THERAINFELL,STINGINGMYEYES,ANDYET,YOUSTOODBESIDEME,WARMINGMYHEART.EVEN
IFYOUHAVEYOUROWNPATHTOFOLLOW,ISTILLWISHTOBETHEUMBRELLATHATSHIELDSYOUFROMTHESTORM.TOMORROW,IWILLSTILLLOVEYOU,BUTIWILLKEEPITLOCKE
DDEEPINSIDE,ASECRETONLYIWILLEVERKNOW.

```

Рисунок 4 – Результат

3.3. Brute-force attack

Перебор всех возможных перестановок ($n!$ вариантов, если известна длина ключа).

Возможно только при небольшой длине ключа ($n < 8$).

- Открытый текст:

Is anyone as foolish as me? I create my own love story, only to end up hurting myself. Is anyone as foolish as me? Ignoring hundreds of paths, I choose to look in only one direction. I wish for just one chance to love you, to be near you, even if it means enduring pain because that is still better than a lifetime of regret. I would rather be a flower spreading its fragrance for you to pick than to wither away unnoticed. That day, the sunlight shimmered so beautifully, and I couldn't help but be captivated by your clear eyes. That day, the rain fell, stinging my eyes, and yet, you stood beside me, warming my heart. Even if you have your own path to follow, I still wish to be the umbrella that shields you from the storm. Tomorrow, I will still love you, but I will keep it locked deep inside, a secret only I will ever know.

```

- Key: 4213
- Код:
import itertools
import math

# Read ciphertext from file
def
load_ciphertext(filename="Columnar_Transposition_Cipher_encryption.txt"):
    with open(filename, 'r', encoding='utf-8') as f:
        return f.read().replace('\n', '').strip()

# Decrypt using a specific permutation (key)
def columnar_decrypt(ciphertext, permutation):
    num_cols = len(permutation)
    num_rows = math.ceil(len(ciphertext) / num_cols)
    total_cells = num_cols * num_rows
    padding = total_cells - len(ciphertext)

    # Calculate column lengths
    col_lengths = [num_rows] * num_cols
    for i in range(padding):
        col_idx = permutation.index(num_cols - i) # columns are 1-based
        col_lengths[col_idx] -= 1

    # Slice ciphertext into columns based on the permutation order
    cols = [''] * num_cols
    index = 0
    sorted_perm = sorted(range(num_cols), key=lambda x: permutation[x])
    for col_pos in sorted_perm:
        length = col_lengths[col_pos]
        cols[col_pos] = ciphertext[index:index + length]
        index += length

    # Reconstruct plaintext row by row
    plaintext = ''
    for row in range(num_rows):
        for col in range(num_cols):
            if row < len(cols[col]):
                plaintext += cols[col][row]

    return plaintext

# Try all permutations for key lengths from 2 to 6 (values from 1 to n)

```



```

def brute_force_columnar(ciphertext, min_key_len=2, max_key_len=6):
    results = []
    for key_len in range(min_key_len, max_key_len + 1):
        base = list(range(1, key_len + 1)) # keys from 1 to n
        for perm in itertools.permutations(base):
            try:
                plaintext = columnar_decrypt(ciphertext, list(perm))
                results.append((perm, plaintext))
            except Exception:
                continue
    return results

# Save results to file
def save_results(results,
filename="Columnar_Transposition_Brute_Force_Results.txt"):
    with open(filename, 'w', encoding='utf-8') as f:
        for perm, text in results:
            f.write(f"Key permutation {perm}:\n{text}\n\n")

# =====
# Main program execution
# =====
if __name__ == "__main__":
    ciphertext =
load_ciphertext("Columnar_Transposition_Cipher_encryption.txt")
    results = brute_force_columnar(ciphertext, min_key_len=2,
max_key_len=6)
    save_results(results)
    print("Brute-force complete. Results saved in
'Columnar_Transposition_Brute_Force_Results.txt'")

78
79 Key permutation (4, 1, 2, 3):
80 IASNYNOEAFS00ILSHSAMEI?CRAETEYMWLNOVSETOYR,OLNYTEONDPUHUTRINMGYSLEF.SIANOYNESAFOLIOSAHSM?EIGNORINGHNUDRNESODFAPTH,SICOHOSTEOLOOKIONNLOYNEIDRETICIO.NIWSIHFRJOU
81
82 Key permutation (4, 1, 3, 2):
83 IANSYNEOAFOSIISLHMAEIC?RATEEYOMWLNONVSTE0Y,ROLYNTENODPHUUTIRNMGYSLEF.SAINONYESFAOLIOSAHSM?IEGORNIGHNUDRNESODFATPH,ISCOOHOSTELOKOIONNLOYNEIRDETICO.INWISHFRJOU
84
85 Key permutation (4, 2, 1, 3):
86 ISANYONEASFOOLISHASME?ICREATEMYOWNLOVESTORY,ONLYTOENDUPHURTINGMYSELF.ISANYONEASFOOLISHASME?IGNORINGHUNDREDSOFAPTHS,ICHOOSETOLOOKINONLYONEDIRECTION.IWISHFORJOU
87
88 Key permutation (4, 2, 3, 1):
89 ISNAYOENASOFOLSIHAMSE?CIRETAEMOYWNOLVETSOR,YONYLTONEUDHPURITNGYMSLEF.IASNYNOEAFS00ILSHSAMEI?GNROINHGUNRDE05FPTAHSI,CHOOSEOTLOKOINNOLYNOEDRIECITONI.WIHSFOJRJOU
90
91 Key permutation (4, 3, 1, 2):
92 INASYENOA0FSOSILHMSAECI?RTAEEOYMWOLNVTSE0,YROYLNTNEODHPUUTIRNMGYSLEF.ASINNOYEFSAOIL0SSAHMI?EGRONIGHNUDRNES0DFTAPHI,SCOOHSOTELK00INONLNOYERIDEITCOI.NWHSIFJRJOU
93

```

Рисунок 5 – Результат Brute-force attack

3.4. Transposition Crib Analysis

Используя "crib" (предполагаемый известный фрагмент открытого текста, например, THE, AND), можно восстановить перестановку.

- Дешифрование текста:

```
ANFISIAYLSYLEPTMLSOSLA?OGDSA,OTOOOIT.SRTCCLYTNYEIMSUGNATIERNFM
REWDHEOSAGFREYOKNIRYOEHANHIROUU,ILTPBPABUEYTDTAESGMENTUOSM
ANHTEYAOWTFOSLHEUETSLOOERORILLVUTLELEES,COILRWSOSLA?EMNERNOU
RGEIYAOHENNNDPSHEONYDCNIOSENOE,ER,NTNDNICETTBEAIFRILTBLRENSGCR
TCAWEANCTDTUGHESAFYDU'LTAVDOLE.T,RF,NGYAEOOEEWIYRVFHYOAOLILSB
ERATEYRHOTR,LIOOUIETKENEETYLEONEOSMCTOOT,YNHIYFANFISIRHROTIOOK
NNRIIHJOHEOOOEOVFEEERPBUHSLTTAAEEETOREAWPDIRAFOPTTTAUTDAYELTMEB
TLACDHBETTYRAEHAHILTIYSD,SDIERGE.NOVUNHOWTWTTMLHHDUMSMMOWS
LE,ILPODPIARNWEK.IYAOHEREWVOOTDUNS.NEOSMGIUEFHCSLILEEOWFUNATV
UBAUEIANIAESASLTHLTOG.UARFERITANOUIHOWNI.T,SISMDEILNONEUCIEYCRS
AYENLINE,YYTBD,MMAEIUERPTL,IIOHBLAISFTT.OWITLYBWKICDIDSELIVN
```

- подозреваемое слово в открытом тексте: hurting
- Код:

```
import itertools
import math

# Read ciphertext from file
def
load_ciphertext(filename="Columnar_Transposition_Cipher_encryption.txt"):
    with open(filename, 'r', encoding='utf-8') as f:
        return f.read().replace('\n', '').strip()

# Decrypt using a key permutation
def columnar_decrypt(ciphertext, permutation):
    num_cols = len(permutation)
    num_rows = math.ceil(len(ciphertext) / num_cols)
    total_cells = num_cols * num_rows
    padding = total_cells - len(ciphertext)

    col_lengths = [num_rows] * num_cols
    for i in range(padding):
        col_idx = permutation.index(num_cols - i)
        col_lengths[col_idx] -= 1

    cols = [''] * num_cols
    index = 0
```

```

sorted_perm = sorted(range(num_cols), key=lambda x: permutation[x])
for col_pos in sorted_perm:
    length = col_lengths[col_pos]
    cols[col_pos] = ciphertext[index:index + length]
    index += length

plaintext = ''
for row in range(num_rows):
    for col in range(num_cols):
        if row < len(cols[col]):
            plaintext += cols[col][row]
return plaintext

# Crib-based analysis
def crib_attack(ciphertext, crib, min_key_len=2, max_key_len=6):
    results = []
    for key_len in range(min_key_len, max_key_len + 1):
        base = list(range(1, key_len + 1))
        for perm in itertools.permutations(base):
            try:
                plaintext = columnar_decrypt(ciphertext, list(perm))
                if crib.lower() in plaintext.lower():
                    results.append((perm, plaintext))
            except Exception:
                continue
    return results

# Save results to file
def save_results(results,
filename="Columnar_Transposition_Crib_Results.txt"):
    with open(filename, 'w', encoding='utf-8') as f:
        for perm, text in results:
            f.write(f"Key permutation {perm}:\n{text}\n\n")

# =====
# Main program execution
# =====
if __name__ == "__main__":
    ciphertext =
load_ciphertext("Columnar_Transposition_Cipher_encryption.txt")
    crib = input("🔍 Enter the suspected word or phrase in the plaintext
(crib): ").strip()
    results = crib_attack(ciphertext, crib, min_key_len=2, max_key_len=6)

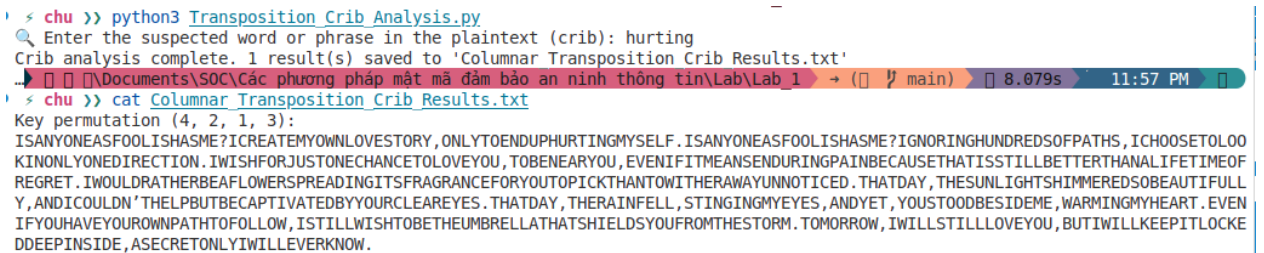
```

```

save_results(results)

print(f"Crib analysis complete. {len(results)} result(s) saved to
'Columnar_Transposition_Crib_Results.txt'")

```



```

chu >> python3 Transposition_Crib_Analysis.py
Enter the suspected word or phrase in the plaintext (crib): hurting
Crib analysis complete. 1 result(s) saved to 'Columnar_Transposition_Crib_Results.txt'
chu >> cat Columnar_Transposition_Crib_Results.txt
Key permutation (4, 2, 1, 3):
ISANYONEASFOOLISHASME?ICREATEMYOWNLOVESTORY, ONLYTOENDUPHURTINGMYSELF. ISANYONEASFOOLISHASME?IGNORINGHUNDREDSOF PATHS, ICHOOSETOLOO
KINONLYONEDIRECTION. IWISHFORJUSTONECHANCETOLOVEYOU, TOBENEARYOU, EVENIFITMEANSSENDURINGPAINBECAUSE THATISSTILLBETTERTHANALIFETIMEOF
REGRET. IWOULDRATHERBEAFLOWERSPREADINGITSFRAGRANCEFORYOUTOPICKTHANTOWITHERAWAYUNNOTICED. THATDAY, THESUNLIGHTSHIMMERDSOBEAUTIFULL
Y, ANDICOULDNTHELPHUTBECAPTIVATEDBYYOURCLEAREYES. THATDAY, THERAINFELL, STINGINGMYEYES, ANDYET, YOUSTOODBESIDEME, WARMINGMYHEART. EVEN
IFYOUHAVEYOUROWNPATHTOFOLLOW, ISTILLWISHTOBETHEUMBRELLATHATSHIELDSYOUFROMTHESTORM. TOMORROW, IWILLSTILLLOVEYOU, BUTIWILLKEEPITLOCKE
DDEEPIINSIDE, ASECRETONLYIWILLEVERKNOW.

```

Рисунок 6 – Crib-based attack

3.5. Transposition Genetic Analysis

Метод Transposition Genetic Analysis — это метод атаки, основанный на использовании генетического алгоритма для расшифровки сообщений, зашифрованных с помощью транспозиционного шифра — типа шифра, в котором изменяется порядок символов, но не сами символы.

Генетический алгоритм — это алгоритм оптимизации, моделирующий процесс естественного отбора. Он использует такие концепции, как:

- Генотип (индивид): В данном случае — это ключ транспозиции (например: [3,1,4,2])
- Приспособленность (*fitness*): Оценивает «осмысленность» расшифрованного текста.
- Селекция, скрещивание (*crossover*), мутация.

Этапы атаки с помощью генетического анализа:

1. Формализация задачи: Ключ представлен как перестановка. Например, при длине блока 6 ключ может быть [2, 4, 1, 6, 5, 3].
2. Инициализация популяции: Создаются случайные ключи (каждый ключ — это особь/индивид).
3. Попытка расшифровки: Каждый ключ применяется для расшифровки шифртекста, разбивая его на блоки и применяя обратную перестановку.

4. Оценка приспособленности (fitness): На основе частоты букв, биграмм или языковых моделей оценивается, насколько расшифрованный текст "осмыслен".
5. Селекция и скрещивание: Выбираются лучшие особи (с наивысшим fitness), из которых создаётся новое поколение с помощью скрещивания и мутаций (например, обмен местами двух элементов в ключе).
6. Повторение: С каждым поколением особи (ключи) эволюционируют, давая всё более "читаемые" расшифровки.
7. Завершение: Алгоритм завершается, когда достигнута достаточно высокая приспособленность, либо по достижении заданного числа поколений.

-> Вероятность нахождения точного исходного сообщения этим методом не гарантирована. Полученный результат может быть приближённым, но иногда очень близким к исходному сообщению.

- Дешифрование текста:

ANFISIAYLSYLEPTMLSOSLA?OGDSA,OTOOOIT.SRTCCLYTNYEIMSUGNATIIERNFM
 REWDHEOSAGFREYOKNIRYOEHANHIROUU,ILTPBPABUEYTDTAESGMENTUOSM
 ANHTEYAOWTFOSLHEUETSLOOERORILLVUTLELEES,COILRWSOSLA?EMNERNOU
 RGEIYAOHENNNDPSHEONYDCNIOSENOE,ER,NTNDNICETTBEAIFRILTBLRENSGCR
 TCAWEANCTDTUGHESAFYDU'LTAVDOLE.T,RF,NGYAEOOEEWIYRVFHYOAOLILSB
 ERATEYRHOTR,LIOOUIETKENEETYLEONEOSMCTOOT,YNHIYFANFISIRHROTIOOK
 NNRIIHJOHEOOEOVFEERPBUHSLTTAEETOREAWPDIRAFOPTTTAUTDAYELTMEB
 TLACDHBETTYRAEHAHILTIYSD,SDIERGE.NOVUNHOWTWTTMLHHDUMSMMOWS
 LE,ILPODPIARNWEK.IYAOHEREWVOOTDUNS.NEOSMGIUEFHCSLILEEOWFUNATV
 UBAUEIANIAESASLTHLTOG.UARFERITANOUIHOWNI.T,SISMDEILNONEUCIEYCRS
 AYENLINE,YYTBD,MMAEIUERPTL,IIOHBLAISFTT.OWITLYBWKICDIDSELIVN

- Код:

```
import math
import random

# Read ciphertext from file
def
load_ciphertext(filename="Columnar_Transposition_Cipher_encryption.txt"):
    with open(filename, "r", encoding="utf-8") as f:
        return f.read().replace("\n", "").strip()

# Decrypt using a key permutation
```

```

def columnar_decrypt(ciphertext, key):
    num_cols = len(key)
    num_rows = math.ceil(len(ciphertext) / num_cols)
    total_cells = num_cols * num_rows
    padding = total_cells - len(ciphertext)

    col_lengths = [num_rows] * num_cols
    for i in range(padding):
        col_idx = key.index(num_cols - i)
        col_lengths[col_idx] -= 1

    cols = [''] * num_cols
    index = 0
    sorted_key = sorted(range(num_cols), key=lambda x: key[x])
    for col_pos in sorted_key:
        length = col_lengths[col_pos]
        cols[col_pos] = ciphertext[index:index + length]
        index += length

    plaintext = ''
    for row in range(num_rows):
        for col in range(num_cols):
            if row < len(cols[col]):
                plaintext += cols[col][row]

    return plaintext

# Fitness function: count how many common English words appear
def fitness(text, common_words):
    words = text.lower().split()
    return sum(1 for word in words if word in common_words)

# Mutate a permutation
def mutate(key):
    new_key = key[:]
    a, b = random.sample(range(len(key)), 2)
    new_key[a], new_key[b] = new_key[b], new_key[a]
    return new_key

# Select the best individuals from the population
def select(population, fitnesses, num):
    sorted_pop = sorted(zip(population, fitnesses), key=lambda x: x[1],
reverse=True)

```

```

        return [x[0] for x in sorted_pop[:num]]

# Generate initial population
def initial_population(size, key_len):
    base = list(range(1, key_len + 1))
    return [random.sample(base, len(base)) for _ in range(size)]

# Genetic algorithm for attacking the cipher
def genetic_attack(ciphertext, common_words, key_len=6, generations=1000,
pop_size=100):
    population = initial_population(pop_size, key_len)
    best_plaintext = ""
    best_score = -1

    for gen in range(generations):
        fitnesses = [fitness(columnar_decrypt(ciphertext, k), common_words)
for k in population]
        best_gen_score = max(fitnesses)
        if best_gen_score > best_score:
            best_score = best_gen_score
            best_plaintext = columnar_decrypt(ciphertext,
population[fitnesses.index(best_gen_score)])
        selected = select(population, fitnesses, pop_size // 2)
        next_gen = selected[:]
        while len(next_gen) < pop_size:
            parent = random.choice(selected)
            child = mutate(parent)
            next_gen.append(child)
        population = next_gen

    return best_plaintext

# =====
# Main program execution
# =====
if __name__ == "__main__":
    ciphertext =
load_ciphertext("Columnar_Transposition_Cipher_encryption.txt")

# Basic set of common English words
common_words = {
    "the", "be", "to", "of", "and", "a", "in", "that", "have", "i",
    "it", "for", "not", "on", "with", "he", "as", "you", "do", "at",

```

```

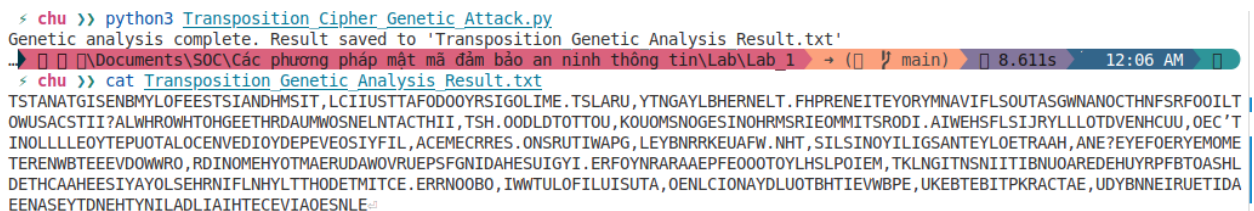
        "this", "but", "his", "by", "from", "they", "we", "say", "her",
        "she", "or", "an", "will", "my", "one", "all", "would", "there",
        "their", "what", "so", "up", "out", "if", "about", "who", "get",
        "which", "go", "me"
    }

    result = genetic_attack(ciphertext, common_words, key_len=6,
generations=1000, pop_size=100)

    with open("Transposition_Genetic_Analysis_Result.txt", "w",
encoding="utf-8") as f:
        f.write(result)

    print("Genetic analysis complete. Result saved to
'Transposition_Genetic_Analysis_Result.txt'")

```



```

< chu >> python3 Transposition Cipher Genetic Attack.py
Genetic analysis complete. Result saved to 'Transposition Genetic Analysis Result.txt'
-> [ ] [ ] \Documents\50C\Các phương pháp mật mã đảm bảo an ninh thông tin\Lab\Lab 1 -> ( [ ] [ ] main) [ ] 8.611s 12:06 AM [ ]
< chu >> cat Transposition Genetic Analysis Result.txt
TSTANATGISENBMYLOFEESTSIANDHMSIT,LCIIUSTTAFODOOYRSIGOLIME.TSLARU,YTNGAYLBHERNELT.FHPRENEITEYORYMNAVIFLSOUTASGWNANOCETHNFSRF00ILT
OWUSACSTII?ALWHR0WHT0HGEETHRDAUMWOSNELNTACTHII,TSH.OODLDTOTTOU,KOUOMSN0GESINOHRMSRIEOMMITSRODI.AIWEHSFLSIJRYLLLOTDVENHCUU,OEC'T
INOLLLEOYTEPUOTALOCENVEDIOYDEPEVEOSIYFIL,ACEMECCRRES.ONSROUTIWPAG,LEYBNRRKEUAFW.NHT,SILSINOYILIGSANTEYLOETRAAH,ANE?EYEF0ERYEMOME
TERENWBTEEEVD0W0R0,RDINOMEHYOTMAERUDAWOV0RUEPSFGNIDAHE0IYI.ERFOYNRARA0EPFE000TOYLSLPOIEM,TKLNGITNSNIITIBNU0AREDEHU0RPF0TOASHL
DETHCAHEESIYAYOLSEHRNIFLNHYLTTHODETMITCE.ERRNO0BO,IWWTULOFILUISUTA,OENLClONAYDLUOTBHTIEVWBPE,UKEBTBITPKRACTAE,UDYBNN0EIRUETIDA
EENASEYTDNEHTYNILADLIAHTECEVIA0ESNLE=

```

Рисунок 7 – Genetic Algorithm attack

->Данный метод криптоанализа не успешно справился с расшифровкой закрытый текст

3.6. Transposition Hill Climbing Analysis

Это эвристический метод атаки, использующий алгоритм восхождения на холм (hill climbing) для поиска наилучшего ключа перестановки при расшифровке транспозиционного шифра.

Как и генетический анализ, метод hill climbing используется для взлома транспозиционных шифров, когда ключ неизвестен. Но вместо эволюции целой популяции (как в GA), hill climbing постепенно улучшает один текущий ключ, проверяя его "соседей".

Принцип работы Hill Climbing

- Начинаем с случайного ключа (перестановки).
- Расшифровываем ciphertext этим ключом.
- Оцениваем результат с помощью функции приспособленности (fitness function) — обычно основанной на частоте биграмм/триграмм/языковой модели.

- Генерируем "соседние ключи", меняя местами 2 позиции в текущем ключе.
- Если новый ключ даёт лучший результат → принимаем его.
- Повторяем, пока не достигнем локального максимума.

Подробные шаги:

1. Инициализация: Генерируем случайную перестановку (например: [2, 4, 1, 3], если длина блока = 4).
2. Пробная расшифровка: Делим ciphertext на блоки и применяем обратную перестановку с текущим ключом.
3. Оценка Fitness: Используем частоты биграмм, частоты символов, или библиотеки типа nltk для оценки "осмысленности" текста.
4. Генерация соседей: Создаём множество соседей, меняя местами 2 символа в ключе.
5. Сравнение и обновление
 - Если сосед даёт более высокий fitness — обновляем ключ.
 - Иначе — оставляем текущий или прекращаем, если улучшений нет после N итераций.

Преимущества:

- Быстрее, чем brute-force при длинных ключах.
- Просто реализуется.
- Эффективен без словаря — только на статистике языка.

Недостатки:

- Возможность застревания в локальном максимуме.
 - Плохо работает на коротких текстах — сложнее оценить "осмысленность".
- Дешифрование текста:

ANFISIAYLSYLEPTMLSOSLA?OGDSA,OTOOOIT.SRTCCLYTNYEIMSUGNATIERNFM
 REWDHEOSAGFREYOKNIRYOEHAAHNHIROUU,ILTPBPABUEYTDTAESGMENTUOSM
 ANHTEYAOWTFOSLHEUETSLOOERORILLVUTLELEES,COILRWSOSLA?EMNERNOU
 RGEIYAOHENNNDPSHEONYDCNIOSENOE,ER,NTNDNICETTBEAIIFRILTBLRENSGCR

TCAWEANCTDTUGHESAFYDU'LTAVDOLE.T,RF,NGYAEOOEEWIYRVFHYOAOLILSB
 ERATEYRHOTR,LIOOUIETKENEETYLEONEOSMCTOOT,YNHIYFANFISIRHROTIOOK
 NNRIIHJOHEOOEOVFEERPBUHSLTTAAEEETOREAWPDIRAFOPTTTAUTDAYELTMEB
 TLACDHBETTYRAEHAHILTIYSD,SDIERGE.NOVUNHOWTWTTMLHHDUMSMMOWS
 LE,ILPODPIARNWEK.IYAOHEREWVOOTDUNS.NEOSMGIEUFHCSLILEEOWFUNATV
 UBAUEIANIAESASLTHLTOG.UARFERITANOUIHOWNL.T,SISMDEILNONEUCIEYCRS
 AYENLINE,YYTBD,MMAEIUERPTL,IIOHBLAISFTT.OWITLYBWKICDIDSELIVN

```
- Код:
import math
import random

# Read ciphertext from file
def
load_ciphertext(filename="Columnar_Transposition_Cipher_encryption.txt"):
    with open(filename, "r", encoding="utf-8") as f:
        return f.read().replace("\n", "").strip()

# Decrypt using a key permutation
def columnar_decrypt(ciphertext, key):
    num_cols = len(key)
    num_rows = math.ceil(len(ciphertext) / num_cols)
    total_cells = num_cols * num_rows
    padding = total_cells - len(ciphertext)

    col_lengths = [num_rows] * num_cols
    for i in range(padding):
        col_idx = key.index(num_cols - i)
        col_lengths[col_idx] -= 1

    cols = [''] * num_cols
    index = 0
    sorted_key = sorted(range(num_cols), key=lambda x: key[x])
    for col_pos in sorted_key:
        length = col_lengths[col_pos]
        cols[col_pos] = ciphertext[index:index + length]
        index += length

    plaintext = ''
    for row in range(num_rows):
        for col in range(num_cols):
            if row < len(cols[col]):
                plaintext += cols[col][row]
```

```

    return plaintext

# Evaluate plaintext based on a simple dictionary
def fitness(text, common_words):
    words = text.lower().split()
    return sum(1 for word in words if word in common_words)

# Hill Climbing Attack
def hill_climb(ciphertext, common_words, key_len=6, max_iterations=5000):
    current_key = random.sample(range(1, key_len + 1), key_len)
    current_plaintext = columnar_decrypt(ciphertext, current_key)
    current_score = fitness(current_plaintext, common_words)

    for _ in range(max_iterations):
        new_key = current_key[:]
        a, b = random.sample(range(key_len), 2)
        new_key[a], new_key[b] = new_key[b], new_key[a]
        new_plaintext = columnar_decrypt(ciphertext, new_key)
        new_score = fitness(new_plaintext, common_words)

        if new_score > current_score:
            current_key = new_key
            current_score = new_score
            current_plaintext = new_plaintext

    return current_plaintext, current_key, current_score

# =====
# Main program execution
# =====
if __name__ == "__main__":
    ciphertext =
    load_ciphertext("Columnar_Transposition_Cipher_encryption.txt")

    # Basic English word set
    common_words = {
        "the", "be", "to", "of", "and", "a", "in", "that", "have", "i",
        "it", "for", "not", "on", "with", "he", "as", "you", "do", "at",
        "this", "but", "his", "by", "from", "they", "we", "say", "her",
        "she", "or", "an", "will", "my", "one", "all", "would", "there",
        "their", "what", "so", "up", "out", "if", "about", "who", "get",
        "which", "go", "me"
    }

```

```

    result_text, final_key, score = hill_climb(ciphertext, common_words,
key_len=6, max_iterations=5000)

    with open("Transposition_Hill_Climbing_Result.txt", "w",
encoding="utf-8") as f:
        f.write(f"Key used: {final_key}\nScore: {score}\nDecrypted
Text:\n{result_text}")

    print("Hill Climbing completed. Result saved to
'Transposition_Hill_Climbing_Result.txt'")

```

```

chu >> python3 Transposition Cipher Hill Climbing Attack.py
Hill Climbing completed. Result saved to 'Transposition Hill Climbing Result.txt'
chu >> cat Transposition Hill Climbing Result.txt
Key used: [6, 2, 4, 3, 5, 1]
Score: 0
Decrypted Text:
ASNTTASGETINLMOBYFTESESIHNMADSLTCI.ITUTISA000FDYGSORIL.MTIESUA, LRYANYTGLRHNBEETHL.PEEIRNTRYEOMIAFNVLTOASUSAWNGNONTFCHSOFIROLU
OSTWAISICT?HLRAWOHHWTGHERETDWUOAMSNETNLAITICH, .SOTHOTLODDT, OKTUOSONUMOIENGOSRRHMIMOIEMTDRISO.EIHAWSILJFSRLLLOYLTNVHDECOUEU, CNT
0'ILELOLLYUEOTPTCLEAONIEOVYEEVDPEYSFOIIC, ELAMRCEERSSOR.NUAIPTWGYLB, ENERURKANWHF.TLSS, IIIOLNYINSTGAEELTYOR, AAHNY?EEEEFYEEORMTME
OERBNTEWEDEOEVW, RRWODMNEIOHMOAYTEAUWRDOERPVSIGDFNAUEIHSGEIRY.FRYAONRPAFAEETOOOYLHPLSO, ETIMKINTLGNINTSIIONABURHDUEEYBPTRFOLSDA
HEAHATCHIEYESASOEYLHFNLRINTYTHLHTDMOEI.CETERONBROOWIT, WUIOLLFUTSAIU, LECONIYNDOALBOHUTTWEBIVPK, EEUBIETTBPCKTAAAD, YEUBINRNEUDTAEI
ESNEEAYEDHTNLTNAYIDIIHLATVCEEANELOSE

```

Рисунок 8 – Hill Climbing attack

-> Данный метод криптоанализа не справился с расшифровкой закрытый текст.

4. Vigenère Cipher

Шифр Виженера — это разновидность многотабличного шифра замены (полиалфавитного шифрования), изобретённая в XVI веке.

Вместо использования одного ключа, как в шифре Цезаря (где каждый символ сдвигается на фиксированное число позиций), шифр Виженера использует ключевую фразу, при которой каждый символ сдвигается на разное количество позиций, в зависимости от соответствующей буквы ключа.

Каждый символ шифруется следующим образом: $C_i = (P_i + K_i) \bmod 26$

Формула расшифровки: $P_i = (C_i - K_i + 26) \bmod 26$

Обозначение	Значение
C_i	Символ на позиции i в шифротексте (<i>ciphertext</i>)
P_i	Символ на позиции i в открытом тексте (<i>plaintext</i>)

K_i	Символ на позиции i в ключе (<i>key</i>) (повторяется, если короче текста)
$mod\ 26$	Остаток от деления по модулю 26 (алфавит A–Z = 0–25)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Рисунок 9 – Vigenère square

4.1. Метод повторения ключа (Key Repetition Method)

- Используется, когда ключ короче открытого текста.
- Ключ повторяется циклически, пока не станет одинаковой длины с текстом.

Преимущества: Очень простой в реализации. Часто используется в классических шифрах.

Недостатки: Уязвим к атакам Касиски или Фридмана, так как ключ повторяется циклически.

4.2. Метод автоключа (Autokey Method)

- Вместо повторения, ключ расширяется путём добавления самого открытого текста.
- Ключ "естественно удлиняется" за счёт содержимого сообщения.

Преимущества: Сложнее для атакующих, так как меньше повторений. Снижает вероятность частотного анализа по блокам.

Недостатки: Если противник знает часть открытого текста, он может восстановить ключ. Обе стороны должны использовать один и тот же алгоритм генерации ключа.

4.3. Vigenère Encryption (Key Repetition Method)

- Открытый текст:

Is anyone as foolish as me? I create my own love story, only to end up hurting myself. Is anyone as foolish as me? Ignoring hundreds of paths, I choose to look in only one direction. I wish for just one chance to love you, to be near you, even if it means enduring pain because that is still better than a lifetime of regret. I would rather be a flower spreading its fragrance for you to pick than to wither away unnoticed. That day, the sunlight shimmered so beautifully, and I couldn't help but be captivated by your clear eyes. That day, the rain fell, stinging my eyes, and yet, you stood beside me, warming my heart. Even if you have your own path to follow, I still wish to be the umbrella that shields you from the storm. Tomorrow, I will still love you, but I will keep it locked deep inside, a secret only I will ever know.

- Key: secretkey

- Код:

```
def read_file(filename):  
    with open(filename, 'r', encoding='utf-8') as f:  
        return f.read()  
  
def write_file(filename, content):  
    with open(filename, 'w', encoding='utf-8') as f:  
        f.write(content)  
  
def repeat_key(key, length):  
    key = key.upper()  
    return (key * (length // len(key) + 1))[:length]  
  
def vigenere_encrypt(plaintext, key):  
    plaintext = ''.join(filter(str.isalpha, plaintext)).upper()
```

```

key = repeat_key(key, len(plaintext))
ciphertext = ''

for p, k in zip(plaintext, key):
    p_val = ord(p) - ord('A')
    k_val = ord(k) - ord('A')
    c_val = (p_val + k_val) % 26
    ciphertext += chr(c_val + ord('A'))

return ciphertext

if __name__ == "__main__":
    # Read data
    plaintext = read_file('input.txt')
    key = read_file('Vigenere_key.txt').strip()

    # Encrypt
    ciphertext = vigenere_encrypt(plaintext, key)

    # Save result
    write_file('Vigenere_encryption.txt', ciphertext)

    print("Encryption complete. Result saved to Vigenere_encryption.txt")

```

- Получаем закрытый текст:

```

AWCECHXIYKJQFPBCLYKQGZGKOERWQAFAGVSTWWVFVRYRJQXQVRWETFMVV
ZRZWCQWPHZWTXCMFICJJHYPGKLCJQXSKLGVKEKAERBJJFJSYZERZWKTLHYW
CLSNFSDSRMFPAFRXNMPWGVZSGSAGKLHFVCEWRGRGTLTXGCLSNFZXISSLSDV
RXKVGWGYGMIGSJGLQGRRLORBMVKEKIKMLTIERYLOXFSXKJWMSPJTIVKIKDLYF
ENZJXDMKWSHIIZBIRAAQLPWBERZITSITPPMOITJTKOEBARIZXLPVYYVCEGXPS
QSWKSISGILLCEXHGMZRITRATYILFSVZGXNXFSXFRCMRIQMRNZKADWFAQOVV
XNWMTICLXPYJDCCEHBMSSDHPKLXVTZMXDVGTVZGXNEVVHUICMMVECITBIW
WWVYEMNEWLLGIEBXJCDPUKMGQMLYQAVCXCELVCCKCHEWRGSFSILSHCEIYR
VFSREECJVEKDITWRKWCHELYNIAFYKYALHEVYXHPSJDSYZWMSPJOMUYXHLIR
ZIWDFKOPJSXJRXLRMCDHUPSNPVMEXJVWVYVKLSOFVKYAGOMNCWMSPJDSX
VCHEFSLMYZPEUICHMVCSVUIBVIGGMGCMWBWEUVGKXMFPAZABVPCNITBRHG

```

4.4. Vigenère Decryption (Key Repetition Method)

- закрытый текст:

```

AWCECHXIYKJQFPBCLYKQGZGKOERWQAFAGVSTWWVFVRYRJQXQVRWETFMVV
ZRZWCQWPHZWTXCMFICJJHYPGKLCJQXSKLGVKEKAERBJJFJSYZERZWKTLHYW

```

CLSNFSDSRMFPAFRXNMPWGVZSGSAGKLHFVCEWRGRGTLTXGCLSNFZXISSLSDV
RXKVWGYGMIGSJGLQGRRLORBMVKEKIKMLTIERYLOXFSXKJWMSPJTIVKIKDLYF
ENZJXDMKWSHIIZBIRAAQLPWBERZITSITPPMOITJTKOEBARIZXLPVYYVCEGXPS
QSWKSISGILLCEXHGMZRITRATIYLFVSZGXNXFSXFRMQRIMRNZKADWFAQOVV
XNWMTICLXBPYJDCCEHBMSSDHPKLXVTZMXDVGTZXGNEVVHUICMMVECTBIW
WWVYEMNEWLLGIEBXJCDPUKMGQMLYQAVCXCELVCCKCHEWRGSFSILSHCEIYR
VFSREECJVEKDITWRKWCHELYNIAFYKYALHEVYXHPSJDSYZWMSPJOMUYXHLIR
ZIWDFKOPJSXJRXLRMCDHUPSNPVMEXJVWVMYVKLSOFVKYAGOMNCWMSPJDSX
VCHEFSLMYZPEUICHMVCSVUIBVIGGMGCMBWEUVGKXMFPAZABVPCNITBRHG

- Key: secretkey

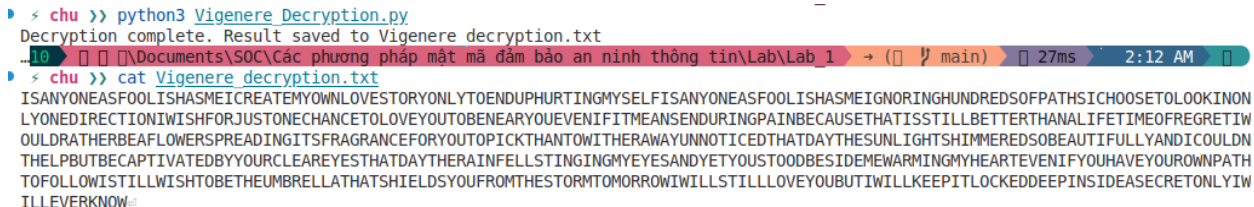
- Код:

```
def read_file(filename):  
    with open(filename, 'r', encoding='utf-8') as f:  
        return f.read()  
  
def write_file(filename, content):  
    with open(filename, 'w', encoding='utf-8') as f:  
        f.write(content)  
  
def repeat_key(key, length):  
    key = key.upper()  
    return (key * (length // len(key) + 1))[:length]  
  
def vigenere_decrypt(ciphertext, key):  
    ciphertext = ''.join(filter(str.isalpha, ciphertext)).upper()  
    key = repeat_key(key, len(ciphertext))  
    plaintext = ''  
  
    for c, k in zip(ciphertext, key):  
        c_val = ord(c) - ord('A')  
        k_val = ord(k) - ord('A')  
        p_val = (c_val - k_val + 26) % 26  
        plaintext += chr(p_val + ord('A'))  
  
    return plaintext  
  
if __name__ == "__main__":  
    # Read data  
    ciphertext = read_file('Vigenere_encryption.txt')  
    key = read_file('Vigenere_key.txt').strip()  
  
    # Decrypt  
    plaintext = vigenere_decrypt(ciphertext, key)
```



```
# Save result
write_file('Vigenere_decryption.txt', plaintext)

print("Decryption complete. Result saved to Vigenere_decryption.txt")
```



```
> < chu >> python3 Vigenere_Decryption.py
Decryption complete. Result saved to Vigenere_decryption.txt
...10> < chu >> cat Vigenere_decryption.txt
ISANYONEASFOOLISHASMEICREATEMYOWNLOVESTORYONLYTOENDUPHURTINGMYSELFISANYONEASFOOLISHASMEIGNORINGHUNDREDSOFPATHSICHOOSETOLOOKINON
LYONEDIRECTIONIWISHFORJUSTONECHANCETOLOVEYOUTOBENEARYOUEVENIFITMEANS ENDURINGPAINBECAUSETHATISSTILLBETTERTHANALIFETIMEOFREGRETIW
OULD RATHERBEAFLOWERSPREADINGITSFRAGRANCEFORYOUTOPICKTHANTOWITHERAWAYUNNOTICEDTHATDAYTHESUNLIGHTSHIMMEREDSOBEAUTIFULLYANDICOULDNT
HELPPBUTBECAPTIVATEDBYYOURCLEAR EYES THATDAYTHERAINFELLSTINGINGMYEYESANDYETYOUSTOOD BESIDEME WARMINGMYHEART EVENIFYOUHAVEYOUR OWNPATH
TO FOLLOWISTILLWISHTOBETHEUMBRELLATHATSHIELDSYOUFROMTHESTORMTOMORROWIWILLSTILLLOVEYOU BUTIWILLKEEPITLOCKEDDEEPIINSIDEASECRETONLYIW
ILLEVERKNOW
```

Рисунок 10 – Результат

4.5. Kasiski Examination

- Анализ Казиски (Kasiski Examination) — метод, предложенный Фридрихом Казиски в 1863 году. Это статистическая атака, которая позволяет:
- Определить длину ключа в шифре Виженера — крайне важный шаг для последующего взлома с помощью частотного анализа каждой колонки (аналогично шифру Цезаря).

Если в открытом тексте встречаются повторяющиеся фрагменты, и они попадают на одинаковую позицию в цикле ключа, то зашифрованные фрагменты тоже будут повторяться.

Пример:

- Открытый текст: ATTACKATDAWN
- Ключ: LEMONLEMONLE

Если последовательность "АСК" повторяется дважды и совпадает по позиции в цикле ключа, то и шифротекст, соответствующий ей (например, "XYZ"), тоже будет повторяться.

Этапы атаки:

Шаг 1: Поиск повторяющихся последовательностей

- Просканировать шифротекст и найти повторяющиеся группы букв (обычно длиной 3 символа и более).
- Зафиксировать расстояние между повторами (в символах).

Шаг 2: Поиск общих делителей

- Для каждого расстояния найти все общие делители.
- Те, что появляются чаще всего — возможные длины ключа (так как повторы часто приходятся на конец цикла ключа).

Шаг 3: Предположить длину ключа

- Выбрать несколько наиболее вероятных значений длины ключа на основе статистики делителей.

Шаг 4: Частотный анализ по колонкам

- Разделить шифротекст на колонки (по количеству предполагаемых символов ключа).
- Применить частотный анализ для каждой колонки отдельно (как в шифре Цезаря).

Простой пример:

Шифротекст (пример):

ZICVTWQNGRZGVTWAVZHCQYGLMGJ

- Последовательность "VTW" встречается дважды: позиции 4 и 10 → расстояние = 6
- "G" также на позициях 7 и 13 → расстояние = 6

→ Общие делители расстояний: 1, 2, 3, 6

→ Возможная длина ключа = 6

Комбинация с другими методами:

После определения длины ключа с помощью метода Казиски обычно применяются:

- Частотный анализ для каждой колонки (по аналогии с Цезарем)
- Метод перебора с использованием словаря популярных коротких ключей (если известна структура открытого текста)

5. Роторная машина Энигма (Enigma Cipher Machine)

Машина "Энигма" — это электромеханическое шифровальное устройство, которое в основном использовалось нацистской Германией во время Второй мировой войны для шифрования военных сообщений.

Основной принцип работы: Каждый раз, когда нажимается клавиша, электрический ток проходит через цепочку механизмов перестановки и возвращает другую букву. При этом роторы поворачиваются, изменяя всю систему перестановки для следующего символа.

5.1. Аппаратная структура Enigma

"Энигма" состоит из 5 основных компонентов:

1. Клавиатура (Keyboard)
 - 26 клавиш, соответствующих буквам A–Z
 - Оператор вводит сообщение, по одной букве за раз.
2. Коммутационная панель (Plugboard / Steckerbrett)
 - Кабели соединяют пары букв, выполняя перестановку до попадания сигнала в роторы
 - Например, A ↔ M, T ↔ G и т.д.
 - Максимум 13 пар может быть подключено
 - Plugboard значительно повышает случайность шифрования
3. Роторы (Walze)
 - Сердце Enigma: каждый ротор — вращающийся диск с внутренней таблицей перестановок (алфавит)
 - Имеет 26 положений (по числу букв)
 - При каждом нажатии клавиши правый ротор поворачивается на 1 шаг, как счётчик
 - При полном обороте ротора, он передаёт вращение следующему (через "зубец" — notch)
 - Каждый ротор похож на шифр Цезаря, но с изменяемым сдвигом.

4. Отражатель (Reflector / UKW – Umkehrwalze)

- Специальный диск, отражающий ток обратно через роторы
- Делает шифр обратимым: одна и та же настройка подходит для шифрования и расшифровки
- Важно: буква никогда не может зашифроваться сама в себя

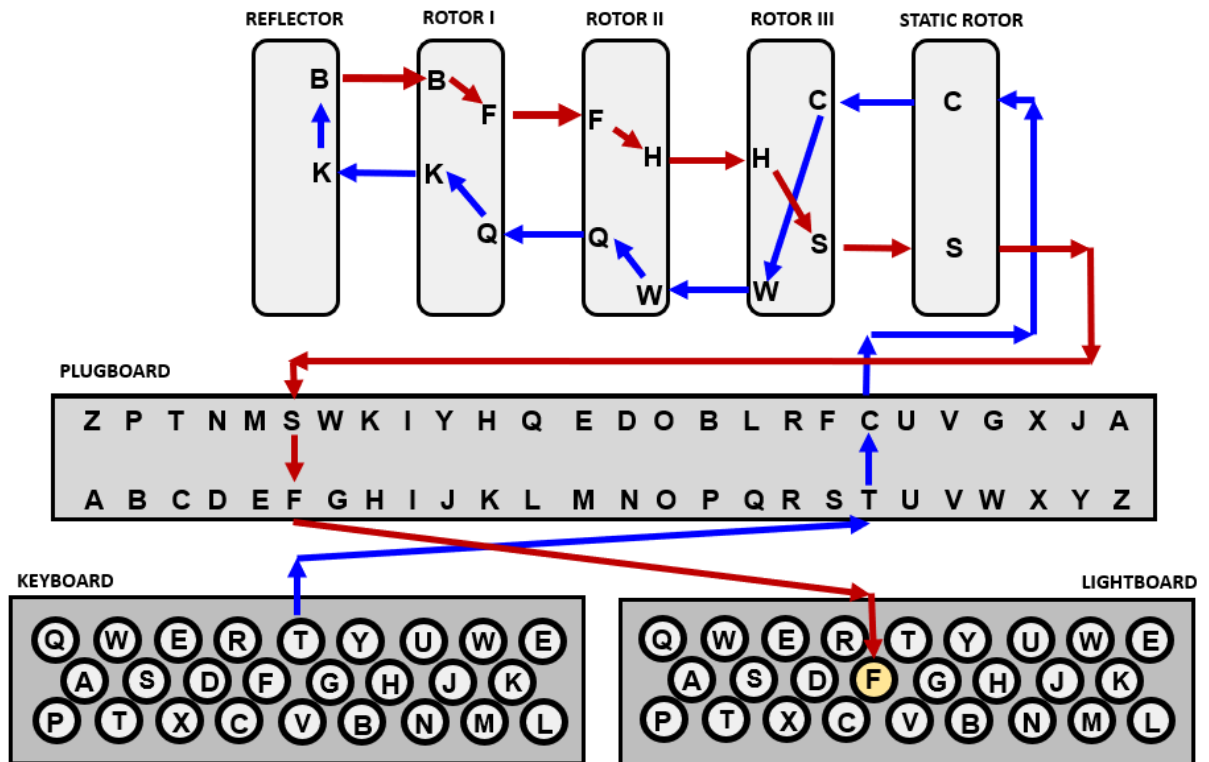
5. Световая панель (Lampboard)

- При шифровании загорается лампочка с зашифрованной буквой
→ Оператор записывает полученную букву (ciphertext)

5.2. Как шифруется каждая буква

Когда оператор нажимает клавишу:

1. Электрический ток проходит через plugboard (выполняется перестановка при наличии пары)
2. Проходит через роторы справа налево, с учётом текущих положений
3. Попадает в отражатель (reflector)
4. Возвращается обратно через роторы слева направо
5. Снова проходит через plugboard
6. В итоге загорается лампа с зашифрованной буквой
7. После этого правый ротор поворачивается на 1 шаг
→ Если он достиг "notch", двигается средний ротор, и т.д.



The path taken by a letter through an Enigma machine as it is encrypted

Рисунок 11 – Пример работы машины Enigma

5.3. Encryption

```
import string

ALPHABET = string.ascii_uppercase

# Sample rotor wirings (Enigma I)
ROTORS = {
    'I': ('EKMFLGDQVZNTOWYHXUSPAIBRCJ', 'Q'),
    'II': ('AJDKSIRUXBLHWTMCQGZNPYFVOE', 'E'),
    'III': ('BDFHJLCPRTXVZNYEIWGAKMUSQO', 'V'),
}

# Sample reflector (UKW-B)
REFLECTOR = dict(zip(ALPHABET, 'YRUHQSLDPXNGOKMIEBFZCWVJAT'))

def rotate(s):
    return s[1:] + s[0]

class Plugboard:
    def __init__(self, wiring_pairs=None):
```

```

        self.wiring = dict(zip(ALPHABET, ALPHABET)) # Default: no swapping

    if wiring_pairs:
        for a, b in wiring_pairs:
            self.wiring[a] = b
            self.wiring[b] = a

    def swap(self, c):
        return self.wiring.get(c, c)

class Rotor:
    def __init__(self, wiring, notch, position='A'):
        self.wiring = wiring
        self.notch = notch
        self.position = position

    def encode_forward(self, c):
        idx = (ALPHABET.index(c) + ALPHABET.index(self.position)) % 26
        return self.wiring[idx]

    def encode_backward(self, c):
        idx = (self.wiring.index(c) - ALPHABET.index(self.position)) % 26
        return ALPHABET[idx]

    def step(self):
        self.position = rotate(self.position)[0]
        return self.position == self.notch

class Enigma:
    def __init__(self, rotor_order, rotor_positions, plugboard_pairs=None):
        self.plugboard = Plugboard(plugboard_pairs)
        self.rotors = []
        for rotor_name, pos in zip(rotor_order, rotor_positions):
            wiring, notch = ROTORS[rotor_name]
            self.rotors.append(Rotor(wiring, notch, pos))

    def encrypt_letter(self, c):
        if c not in ALPHABET:
            return c

        # Rotor stepping
        rotate_next = self.rotors[2].step()
        if rotate_next:

```

```

        rotate_next = self.rotors[1].step()
        if rotate_next:
            self.rotors[0].step()

    # Plugboard pass #1
    c = self.plugboard.swap(c)

    # Forward through rotors
    for rotor in reversed(self.rotors):
        c = rotor.encode_forward(c)

    # Reflector
    c = REFLECTOR[c]

    # Backward through rotors
    for rotor in self.rotors:
        c = rotor.encode_backward(c)

    # Plugboard pass #2
    c = self.plugboard.swap(c)

    return c

def encrypt_message(self, message):
    message = ''.join(filter(str.isalpha, message)).upper()
    return ''.join(self.encrypt_letter(c) for c in message)

# ==== File Helpers ====

def read_file(filename):
    with open(filename, 'r', encoding='utf-8') as f:
        return f.read()

def write_file(filename, content):
    with open(filename, 'w', encoding='utf-8') as f:
        f.write(content)

# ==== Main ====

if __name__ == "__main__":
    # === Enigma machine configuration ===
    rotor_order = ['I', 'II', 'III']
    rotor_positions = ['A', 'A', 'A'] # Initial rotor positions

```

```

# Plugboard configuration - list of letter swap pairs
plugboard_pairs = [
    ('A', 'M'),
    ('T', 'G'),
    ('L', 'P'),
    ('O', 'K'),
    ('E', 'R')
]

# Read plaintext from file
plaintext = read_file('input.txt')

# Encrypt
enigma = Enigma(rotor_order, rotor_positions, plugboard_pairs)
ciphertext = enigma.encrypt_message(plaintext)

# Write to file
write_file('enigma_encryption.txt', ciphertext)

print("Encryption complete. Result saved to enigma_encryption.txt")

```

```

D < chu >> python3 Enigma_Cipher_encryption.py
Encryption complete. Result saved to enigma encryption.txt
D < chu >> cat enigma_encryption.txt
WPEOXNOAEPGNNTWPZEPUAWJBAELAXNIOTNQAPLNBXNOTXLNAOKMSZMBLWOFUXPATGWPEOXNOAEPGNNTWPZEPUAWFONBWOZMOKBAKPNSELZPWJZNNPALNTNNDWONO
TXNOAKWBAJLWNOWIWPZGNBCMPLNOAJZEOJALNTNQAXNMLNRAOABXNMAQAOWGWLUAEOPAOKMBWOFSEWORAJEMPALZELWPPLWTTRALLABLZEOETWGLWUANGBAFBALWI
NMTKBELZABRAEGTNIABPSBAEKWOFWLPGBEFBEOJAGNBXNMLNSWJDLZEOJNIWLZABEIEXMoonLWJAKLZELKEXLZAPMOTWFZLPZWUUBAKPNRAEMLWGMTTTEOKWJNMTKO
LZATSRMLRAJESLWQELAKRXNMBJTAEBAXAPLZELKEXLZABEWOGATTPLOWFOWFUXAXAPEOKXALXNMPLNNKRAPWKAUAIEBUWOFUXZAEBLAQAOXNMZEXAXNMBNIOSELZ
LNGNTTNIWPLWTTIWPZLNRLZAMURBATELZELPZWATKPNMGBNULZAPLNBULNUNBNIIWTTPLWTTTNTQAXNMRMLWIWTTDAASWLTNJDAKKAASWOPWKAEPJABALNOTXWI
WTTAQABDONI

```

Рисунок 12 – Encryption

5.4. Decryption

```

import string

ALPHABET = string.ascii_uppercase

# Sample rotor wirings (Enigma I)
ROTORS = {
    'I': ('EKMFLGDQVZNTOWYHXUSPAIBRCJ', 'Q'),
    'II': ('AJDKSIRUXBLHWTMCQGZNPYFVOE', 'E'),
    'III': ('BDFHJLCPRTXVZNYEIWGAKMUSQO', 'V'),
}

# Sample reflector (UKW-B)

```



```

REFLECTOR = dict(zip(ALPHABET, 'YRUHQSLDPXNGOKMIEBFZCWVJAT'))

def rotate(s):
    return s[1:] + s[0]

class Plugboard:
    def __init__(self, wiring_pairs=None):
        self.wiring = dict(zip(ALPHABET, ALPHABET)) # Default: no
substitution
        if wiring_pairs:
            for a, b in wiring_pairs:
                self.wiring[a] = b
                self.wiring[b] = a

    def swap(self, c):
        return self.wiring.get(c, c)

class Rotor:
    def __init__(self, wiring, notch, position='A'):
        self.wiring = wiring
        self.notch = notch
        self.position = position

    def encode_forward(self, c):
        idx = (ALPHABET.index(c) + ALPHABET.index(self.position)) % 26
        return self.wiring[idx]

    def encode_backward(self, c):
        idx = (self.wiring.index(c) - ALPHABET.index(self.position)) % 26
        return ALPHABET[idx]

    def step(self):
        self.position = rotate(self.position)[0]
        return self.position == self.notch

class Enigma:
    def __init__(self, rotor_order, rotor_positions, plugboard_pairs=None):
        self.plugboard = Plugboard(plugboard_pairs)
        self.rotors = []
        for rotor_name, pos in zip(rotor_order, rotor_positions):
            wiring, notch = ROTORS[rotor_name]
            self.rotors.append(Rotor(wiring, notch, pos))

```

```

def encrypt_letter(self, c):
    if c not in ALPHABET:
        return c

    # Rotor stepping
    rotate_next = self.rotors[2].step()
    if rotate_next:
        rotate_next = self.rotors[1].step()
        if rotate_next:
            self.rotors[0].step()

    # Plugboard pass #1
    c = self.pluginboard.swap(c)

    # Through rotors (forward)
    for rotor in reversed(self.rotors):
        c = rotor.encode_forward(c)

    # Reflector
    c = REFLECTOR[c]

    # Through rotors (backward)
    for rotor in self.rotors:
        c = rotor.encode_backward(c)

    # Plugboard pass #2
    c = self.pluginboard.swap(c)

    return c

def encrypt_message(self, message):
    message = ''.join(filter(str.isalpha, message)).upper()
    return ''.join(self.encrypt_letter(c) for c in message)

# ==== File Helpers ====

def read_file(filename):
    with open(filename, 'r', encoding='utf-8') as f:
        return f.read()

def write_file(filename, content):
    with open(filename, 'w', encoding='utf-8') as f:
        f.write(content)

```

```

# ===== Main =====

if __name__ == "__main__":
    # === Must match encryption settings ===
    rotor_order = ['I', 'II', 'III']
    rotor_positions = ['A', 'A', 'A'] # Must match the starting positions
    during encryption

    plugboard_pairs = [
        ('A', 'M'),
        ('T', 'G'),
        ('L', 'P'),
        ('O', 'K'),
        ('E', 'R')
    ]

    # Read ciphertext from encrypted file
    ciphertext = read_file('enigma_encryption.txt')

    # Reinitialize Enigma machine with the same settings
    enigma = Enigma(rotor_order, rotor_positions, plugboard_pairs)

    # Decrypt (in Enigma, encryption and decryption use the same function)
    plaintext = enigma.encrypt_message(ciphertext)

    # Write output to file
    write_file('enigma_decryption.txt', plaintext)

    print("Decryption complete. Result saved to enigma_decryption.txt")

```

```

$ chu >> python3 enigma_decryption.py
Decryption complete. Result saved to enigma_decryption.txt
$ chu >> cat enigma_decryption.txt
ISANYONEASFOOLISHASMEICREATEMYOWNLOVESTORYONLYTOENDUPHURTINGMYSELFISANYONEASFOOLISHASMEIGNORINGHUNDREDSOFPATHSICHOOSETOLOOKINON
LYONEDIRECTIONIWISHFORJUSTONECHANCETOLOVEYOUOBENEARYOUEVENIFITMEANSENDURINGPAINBECAUSETHATISSTILLBETTERTHANALIFETIMEOFREGRETIW
OULDRAATHERBEAFLOWERSPREADINGITSFRAGRANCEFORYOUTOPICKTHANTOWITHERAWAYUNNOTICEDTHATDAYTHESUNLIGHTSHIMMEREDSOBEAUTIFULLYANDICOULDN
THELPBUTBECAPTIVATEDBYYOURCLEAREYESTHATDAYTHERAINFELLSTINGINGMYEYESANDYETYOUSTOODBESIDEME WARMINGMYHEARTEVENIFYOUHAVEYOUROWNPATH
TOFOLLOWISTILLWISHTOBETHEUMBRELLATHATSHIELDSYOUFROMTHESTORMTOMORROWIWILLSTILLLOVEYOU BUTIWILLKEEPITLOCKEDDEEPIINSIDEASECRETONLYIW
ILLEVERKNOW

```

Рисунок 13 – Decryption

5.5. Turing Bombe

Turing Bombe — это электромеханическое устройство, разработанное Аланом Тьюрингом (совместно с Гордоном Уэлчманом) в 1940 году в Bletchley Park,

Великобритания. Оно было создано для автоматизации процесса взлома Enigma, которая ежедневно имела миллионы миллионов возможных настроек ключа.

5.5.1. Цель Bombe

Определить ежедневную конфигурацию ключа Enigma, включая:

- Порядок роторов
- Начальные позиции роторов
- Подключения на коммутационной панели (Plugboard)

5.5.2. Как работает Bombe

- Bombe не перебирает случайные ключи вслепую. Вместо этого она опирается на предположения (cribs) — фрагменты открытого текста, которые предположительно содержатся в шифротексте.
- Подробные этапы атаки Turing Bombe:

1. Crib (предполагаемый фрагмент)

Crib — это предполагаемый фрагмент открытого текста (например, "WEATHER", "HEILHITLER", "ATTACKATDAWN"), который, возможно, находится в определённой позиции шифротекста. Сдвигая crib вдоль ciphertext и пробуя разные положения, можно построить логическую сеть (menu).

2. Построение "Menu" из crib

На основе crib + ciphertext криптоаналитики строят меню: цепочку логических связей между буквами открытого текста и шифротекста, проходящих через plugboard и роторы.

Пример:

Открытый текст: W E A T H E R

Шифротекст: X Q B M L Q G

Получаем логические связи: $W \rightarrow X$, $E \rightarrow Q$, $A \rightarrow B$ и т.д.

3. Запуск Bombe — перебор тысяч комбинаций

Bombe действует как сеть механических машин, моделирующих множество вариантов Enigma одновременно:

- Она моделирует обратное шифрование: от ciphertext к предполагаемому plaintext
- Перебирает все комбинации роторов (порядок + начальные позиции)
- При этом делает предположение о паре plugboard (например $A \leftrightarrow G$) и проверяет, возникают ли противоречия

Если противоречий нет, это может быть верный ключ.

4. Отбрасывание неверных ключей (Contradictions)

Если в логической сети возникает противоречие, например:

Буква не может одновременно быть и A, и G

→ комбинация отвергается

5. Отправка потенциального ключа на проверку

Если Bombe не обнаруживает противоречий, она останавливается и сообщает “hit” — возможный верный ключ.

Тогда проверяющий оператор (обычно женщины Wrens):

- Вводят найденную конфигурацию в настоящую машину Enigma
- Пробуют расшифровать сообщение
- Если текст осмысленный, ключ подтверждается

Сводка атаки Bombe:

Этап	Описание
1. Выбор crib	Предположить фрагмент открытого текста
2. Сопоставление с шифром	Построить логическое "меню"
3. Моделирование Enigma на Bombe	Перебор роторов и начальных позиций

4. Проверка противоречий	Противоречие \rightarrow исключение, нет противоречия \rightarrow возможно верно
5. Ручная проверка	Проверка на настоящей машине Enigma

ЗАКЛЮЧЕНИЕ

В процессе выполнения данной лабораторной работы мы изучили основные принципы работы исторических шифров, включая шифры Цезаря, Виженера, транспозиции и замены. Мы также провели их криптоанализ, в результате чего смогли восстановить открытый текст и секретный ключ для шифров Цезаря, Виженера, транспозиции и замены, хотя не во всех случаях методы криптоанализа оказались успешными.

Кроме того, мы выполнили криптоанализ машины Энигма, используя различные подходы, и смогли расшифровать исходный текст, когда были установлены точные настройки роторной машины. Длина сообщения, вероятно, сыграла важную роль в успешном проведении криптоанализа. Однако можно заключить, что шифрование с использованием машины Энигма обладает более высокой устойчивостью к криптоанализу по сравнению с другими рассмотренными методами шифрования.

В результате выполнения работы поставленная цель была достигнута, а все задачи успешно выполнены.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Очень.
2. Важный.
3. Источник.