

Правительство Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»
(НИУ ВШЭ)

Московский институт электроники и математики им. А.Н.Тихонова

ОТЧЁТ
О ПРАКТИЧЕСКОЙ РАБОТЕ №1
по дисциплине «Основы криптографии и стеганографии»
ПОДСТАНОВОЧНЫЕ ШИФРЫ

Студент БИБ252

Мельников В.К.
«___» 2026 г.

Руководитель
Заведующий кафедрой информационной
безопасности киберфизических систем
канд. техн. наук, доцент

О.О. Евсютин
«___» 2026 г.

СОДЕРЖАНИЕ

1 Задание на практическую работу	3
2 Краткая теоретическая часть	4
2.1 Описание шифров	4
2.2 Методы криptoанализа шифров	5
3 Примеры шифрования	7
4 Программная реализация шифров	10
5 Примеры криptoанализа	15
6 Выводы о проделанной работе	25
7 Список используемых источников	26

1 Задание на практическую работу

Цель данной работы заключается в освоении навыков программной реализации и криптоанализа простых подстановочных шифров.

В рамках практической работы необходимо выполнить следующее:

- Написать программную реализацию аффинного шифра, аффинного рекуррентного шифра, а также же шифра простой подстановки.
- Изучить методы криптоанализа моноалфавитных шифров, а так же провести криптоанализ данных шифров.
- Подготовить отчёт о проделанной работе.

2 Краткая теоретическая часть

2.1 Описание шифров

Шифр простой замены — простейший пример подстановочного шифра, в котором каждому символу открытого текста соответствует один символ зашифрованного текста. Сообщение рассматривается как последовательность символов алфавита A , мощности m , а ключ k формируется как произвольная перестановка алфавита.

$$k = \begin{pmatrix} x_1 & x_2 & \dots & x_m \\ y_1 & y_2 & \dots & y_m \end{pmatrix} \quad (1)$$

В данном случае x_1, x_2, \dots, x_m — это символы открытого текста, а y_1, y_2, \dots, y_m — символы зашифрованного текста.

Зашифрование осуществляется заменой каждого символа открытого текста на соответствующий ему символ зашифрованного текста и может быть записано как

$$E_k(x) = (k(x_1), \dots, k(x_m)) \quad (2)$$

Расшифрование осуществляется обратной заменой и может быть записано как

$$D_k(y) = (k^{-1}(y_1), \dots, k^{-1}(y_m)), \quad (3)$$

$$k^{-1} = \begin{pmatrix} y_1 & y_2 & \dots & y_m \\ x_1 & x_2 & \dots & x_m \end{pmatrix} \quad (4)$$

Аффинный шифр — это частный случай шифра замены, который использует математическую функцию для преобразования символов, называемую аффинное преобразование.

Данный шифр реализует замену символов открытого текста с использованием операций в арифметике остатков. Символы алфавита A мощностью m представляются элементами множества Z_m , а ключ k состоит из двух чисел a и b , принадлежащих Z_m , таких что a и m взаимно простые.

Ключ может быть представлен как перестановка (1), в котором

$$y_i = (a \cdot x_i + b) \pmod{m}, i \in \{1, 2, \dots, m\} \quad (5)$$

Расшифрование осуществляется с помощью обратного аффинного преобразования, которое может быть записано как

$$x_i = (a^{-1} \cdot (y_i - b)) \pmod{m}, i \in \{1, 2, \dots, m\} \quad (6)$$

Аффинный рекуррентный шифр — это усиление аффинного шифра, который использует рекуррентные функции для генерации ключей.

Алгоритм шифрования основан на вычислении нового ключа для каждого символа открытого текста на основе предыдущих символов и ключей. Для этого необходимо задать

два ключевые пары: $k_1 = (a_1, b_1)$ и $k_2 = (a_2, b_2)$, а последующие ключевые пары будут вычисляться по формуле:

$$k_i = ((a_{i-1} \cdot a_{i-2}) \pmod{m}, (b_{i-1} + b_{i-2}) \pmod{m}), i \in \{3, 4, \dots, m\} \quad (7)$$

Алгоритм зашифрования, расшифрования и формулы для вычисления ключей аналогично аффинному шифру, но с использованием рекуррентных ключей.

2.2 Методы криptoанализа шифров

Криptoанализ шифра простой замены. Для каждого алфавита мощности m может быть подобрано $m!$ различных ключей, что делает полный перебор ключей неэффективным и нецелесообразным. Однако, шифр простой замены является моноалфавитным шифром, что означает, что каждый символ открытого текста всегда заменяется одним и тем же символом зашифрованного текста. Это делает его уязвимым к частотному анализу, который заключается в изучении частоты появления символов в зашифрованном тексте и сопоставлении их с частотой появления символов в языке, на котором написан открытый текст. Например, в русском языке буква «о» встречается чаще всего, следовательно, символ, который встречается чаще всего в зашифрованном тексте, вероятно, соответствует букве «о». Аналогично, можно сопоставить другие часто встречающиеся символы с наиболее частыми буквами в языке.

Криptoанализ аффинного шифра. Для каждого алфавита мощности m необходимо подобрать два числа a и b , где $\gcd(a, m) = 1$, а $b \in \{0, 1, \dots, m - 1\}$. Чтобы найти количество взаимно простых чисел можно воспользоваться функцией Эйлера $\varphi(m)$, которая определяет количество чисел от 1 до $m - 1$, которые являются взаимно простыми с m , где

$$\varphi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right), \quad n > 1 \quad (8)$$

p — простое число и пробегает все значения, участвующие в разложении n на простые множители

Так, для русского алфавита мощности $33 = 3^1 \cdot 11^1$

$$\varphi(33) = 33 \cdot \left(1 - \frac{1}{3}\right) \cdot \left(1 - \frac{1}{11}\right) = 20 \quad (9)$$

В таком случае количество всех возможных ключей равно $33 \cdot 20 = 660$. Такое количество различных ключей позволяет сделать полный перебор, все современные компьютеры способны выполнить его меньше чем за секунду. Однако, существуют и другие способы криptoанализа. Аффинный шифр является моноалфавитным шифром, что делает его уязвимым к частотному анализу, аналогично шифру простой замены.

Криptoанализ аффинного рекуррентного шифра. Аффинный рекуррентный шифр является более сложным, чем аффинный шифр, из-за использования рекуррентных ключей, что делает его полиалфавитным шифром, а следовательно, более устойчивым к частотному

анализу. Так, разные символы открытого текста могут быть зашифрованы одними и теми же символами зашифрованного текста и наоборот, что затрудняет использование частотного анализа.

Разберем сложность полного перебора. Для этого нужно верно подобрать четыре числа: a_1 , b_1 , a_2 и b_2 . Числа a_1 и a_2 должны быть взаимно простыми с мощностью алфавита m , а числа b_1 и b_2 могут принимать любые значения от 0 до $m - 1$. Таким образом, количество возможных ключей равно $\varphi(m)^2 \cdot m^2$. Для русского алфавита это будет равно $20^2 \cdot 33^2 = 435600$. Такое количество ключей делает полный перебор сложным, но не невозможным для современных компьютеров. Однако он является уязвимым к криптоанализу на основе открытого текста.

3 Примеры шифрования

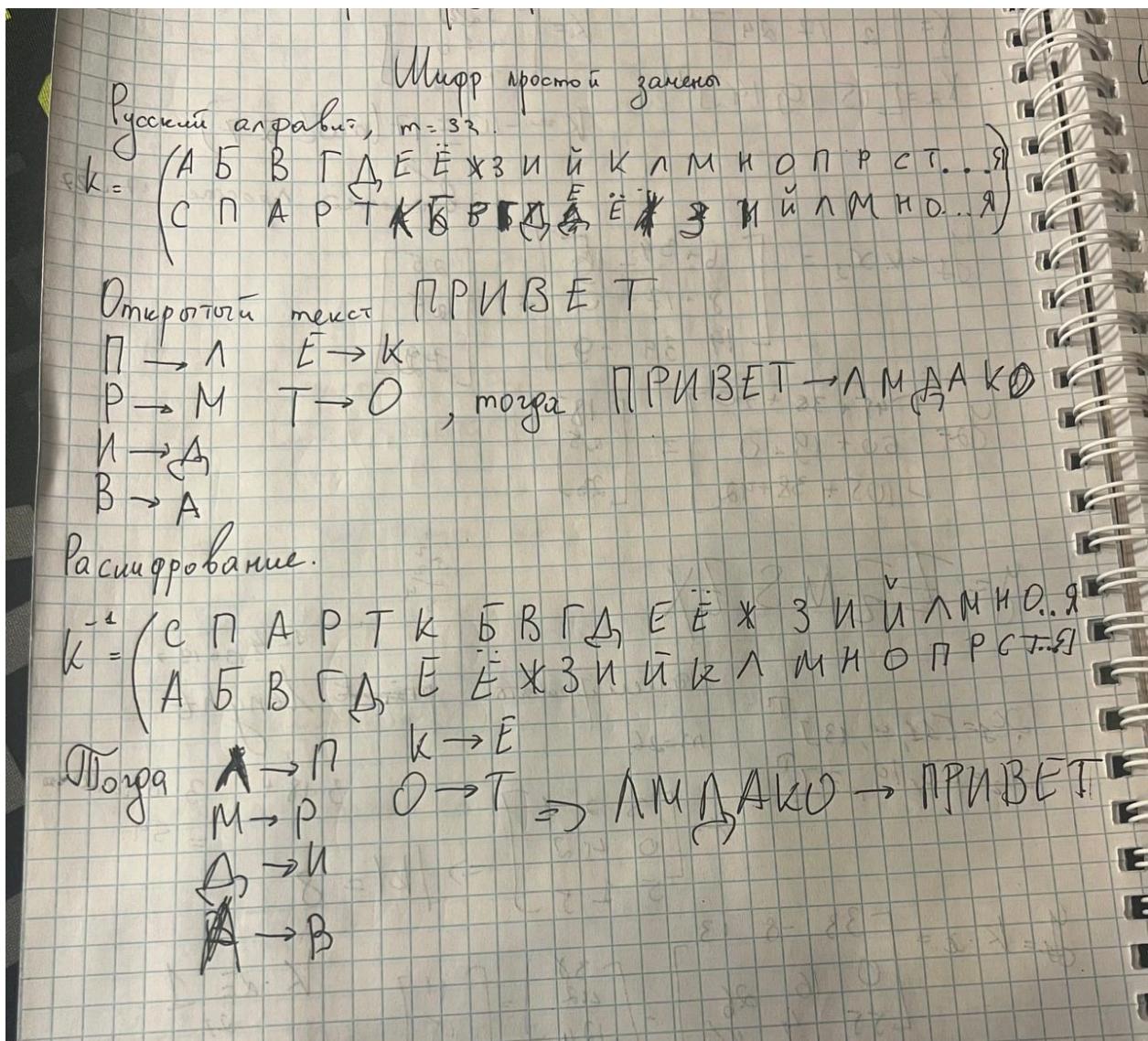


Рисунок 1 — Пример шифрования и расшифрования текста с помощью шифра простой замены

Русский алфавит, $m = 33$

$$d = (4, 7), \text{ НОД}(4, 33) = 1$$

Формула $y_i = (a \cdot x_i + b) \bmod 33$
Шифруем ПРИВЕТ

$$a = 16$$

$$b = 7$$

$$x_1 = 9$$

$$x_2 = 2$$

$$x_3 = 5$$

$$x_4 = 10$$

$$y_1 = (4 \cdot 16 + 7) \bmod 33 = 71 \bmod 33 = 5$$

$$y_2 = (4 \cdot 17 + 7) \bmod 33 = 75 \bmod 33 = 9.$$

$$y_3 = (4 \cdot 8 + 7) \bmod 33 = 43 \bmod 33 = 10$$

$$y_4 = (2 \cdot 9 + 7) \bmod 33 = 15$$

$$y_5 = (5 \cdot 2 + 7) \bmod 33 = 27.$$

$$y_6 = (3 \cdot 4 + 7) \bmod 33 = 83 \bmod 33 = 17$$

$$5 = e; 9 = u; 10 = y; 0 = i; 15 = r; 27 = t; 17 = p.$$

ПРИВЕТ. ЕИЮРЫР

Расшифрование:

$$\text{Формула } x_i = (a^{-1} \cdot (y_i - b)) \bmod 33$$

$$a^{-1} = 25$$

по расч. алг. Евклида

$$\begin{array}{r} 33 \\ 25 \overline{) 8} \\ 25 \overline{) 3} \\ 25 \overline{) 3} \\ \hline 0 \end{array}$$

$$25 \equiv -32 \bmod 33$$

$$-8 + 33 = 25.$$

$$x_1 = (25 \cdot (5 - 7)) \bmod 33 = -50 \bmod 33 = 16 = n$$

$$x_2 = (25 \cdot (9 - 7)) \bmod 33 = 50 \bmod 33 = 17 = p$$

$$x_3 = (25 \cdot (10 - 7)) \bmod 33 = 75 \bmod 33 = 9 = u$$

$$x_4 = (25 \cdot (15 - 7)) \bmod 33 = 200 \bmod 33 = 2 = b$$

$$x_5 = (25 \cdot (27 - 7)) \bmod 33 = 500 \bmod 33 = 5 = e$$

$$x_6 = (25 \cdot (17 - 7)) \bmod 33 = 250 \bmod 33 = 17 = t$$

Расшифровка
ПРИВЕТ

Рисунок 2 — Пример шифрования и расшифрования текста с помощью аффинного шифра

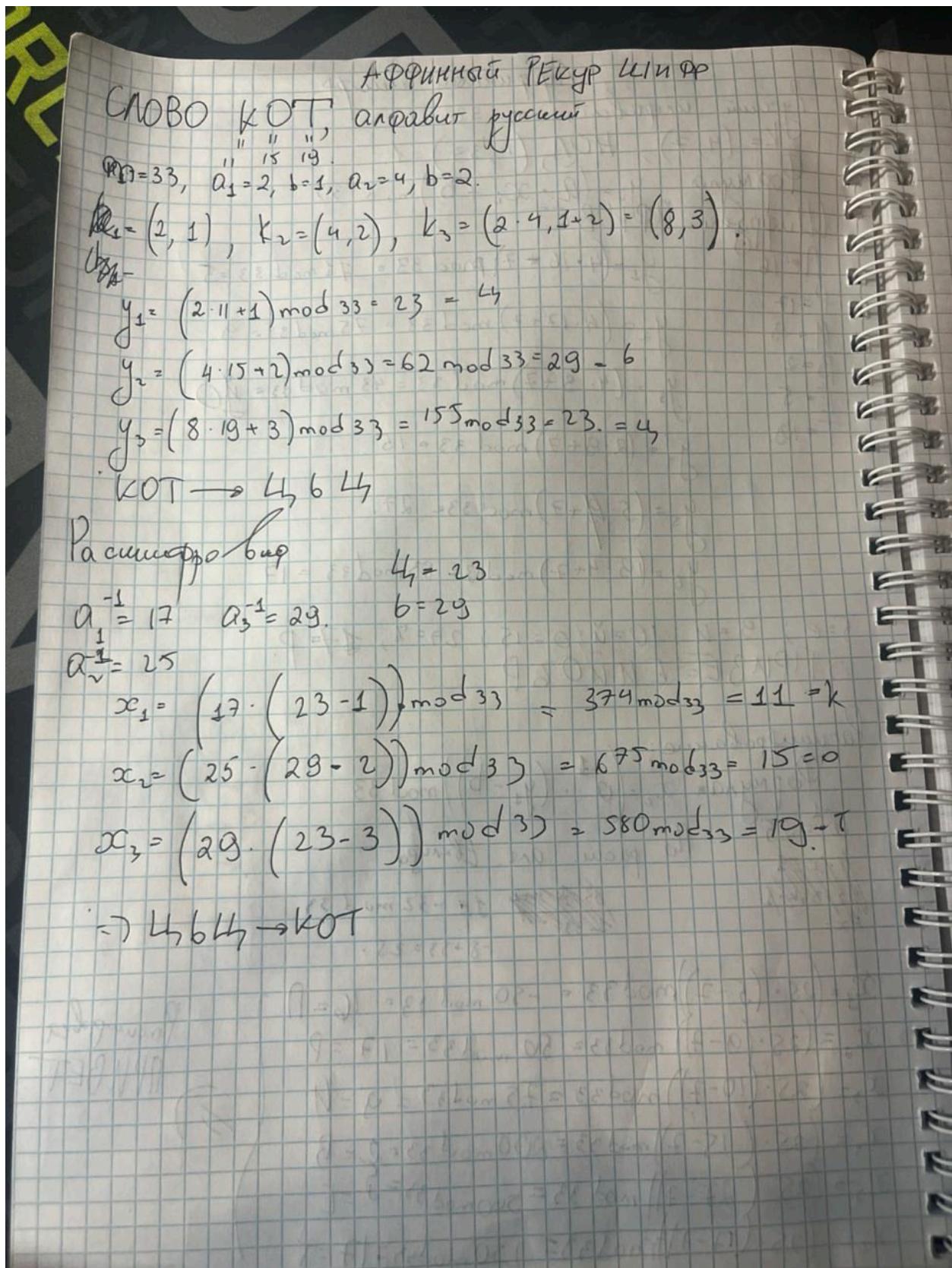


Рисунок 3 — Пример шифрования и расшифрования текста с помощью аффинного рекуррентного шифра

На этом примере (см. Рисунок 3) видно, что только аффинный рекуррентный шифр обеспечивает шифрование разных символов открытого текста в одни и те же символы зашифрованного текста, что делает его более устойчивым к частотному анализу.

4 Программная реализация шифров

Программная реализация шифров была выполнена на языке Python. Реализованы функции записи шифротекста или расшифрованного текста в отдельный файл (см. Рисунок 4), а так же исключение для ошибки записи файла. Написана функция вычисления НОД для двух чисел (см. Рисунок 5) для проверки валидности ключа.

Реализовано 6 функций для расшифровки и шифровки текста тремя способами: простой замены, аффинного шифра и аффинного рекуррентного шифра. Каждая функция принимает на вход открытый текст, ключ и алфавит, а возвращает зашифрованный текст или расшифрованный текст в зависимости от функции. Учтены некоторые исключения, такие как несоответствие мощности алфавита и некорректный ключ (см. Рисунки 6,7,8,9).

Пользователь из консоли может выбрать способ шифрования, ввести открытый текст, ключ и алфавит, а также указать имя файла для сохранения результата. Результат будет записан в указанный файл, в случае если файл с таким именем существует, он будет перезаписан (см. Рисунки 10, 11).

```
Explain Code | Generate Tests | Ask Sourcery | You, 5 minutes ago | 1 author (You)
class SaveFileException(Exception):
    """Ошибка сохранения файла"""
    pass

Explain Code | Generate Tests | Generate Docstrings | Ask Sourcery
def save_in_file(text):
    ans = input("Сохранить содержимое вывода в отдельный файл? [y/n]: ")
    if ans == 'y':
        filename = input("Введите название файла: ")
        if not filename.endswith(".txt"):
            filename += ".txt"
        try:
            with open(filename, "w") as f:
                f.write(text)
        except:
            raise SaveFileException("Ошибка записи файла")
```

Рисунок 4 — Реализация функции записи в файл и ее исключения

```
Explain Code | Generate Tests | Ask Sourcery
@staticmethod
def gcd(a, b):
    """Алгоритм Евклида"""
    while b != 0:
        a, b = b, a % b
    return a
```

Рисунок 5 — Реализация функции вычисления НОД

```

Explain Code | Generate Tests | Generate Docstrings | Ask Sourcery
def inverse(self, a):
    return pow(a, -1, self.m)

Explain Code | Generate Tests | Generate Docstrings | Ask Sourcery
def simple_substitution_encrypt(self, text, key):
    if len(key) != self.m:
        raise KeyValidityError(f'Длина ключа ({key}) не соответствует мощности алфавита ({self.m})')
    if sorted(key) != sorted(self.alphabet):
        raise CharacterMismatch("Один или более символов не указаны в алфавите или наоборот")
    result = ''
    for char in text:
        if char in [" ", ".", ","]:
            result += char
            continue
        index = self.alphabet.find(char)
        result += key[index]
    return result

Explain Code | Generate Tests | Generate Docstrings | Ask Sourcery
def simple_substitution_decrypt(self, text, key):
    if len(key) != self.m:
        raise KeyValidityError(f'Длина ключа ({key}) не соответствует мощности алфавита ({self.m})')
    if sorted(key) != sorted(self.alphabet):
        raise CharacterMismatch("Один или более символов не указаны в алфавите или наоборот")
    result = ''
    for char in text:
        if char in [" ", ".", ","]:
            result += char
            continue
        index = key.find(char)
        result += self.alphabet[index]
    return result

```

Рисунок 6 — Реализация функций шифрования и расшифрования текста с помощью шифра простой замены

```

Explain Code | Generate Tests | Generate Docstrings | Ask Sourcery
def affine_cipher_encrypt(self, text, key_a, key_b):
    key_a = int(key_a)
    key_b = int(key_b)
    if self.gcd(key_a, self.m) != 1:
        raise KeyValidityError("Первое число ключа и мощность алфавита не взаимно просты")
    result = ''
    for char in text:
        if char in [" ", ".", ","]:
            result += char
            continue
        x = self.alphabet.find(char)
        y = key_a * x + key_b
        result += self.alphabet[y % self.m]
    return result

Explain Code | Generate Tests | Generate Docstrings | Ask Sourcery
def affine_cipher_decrypt(self, text, key_a, key_b):
    key_a = int(key_a)
    key_b = int(key_b)
    if self.gcd(key_a, self.m) != 1:
        raise KeyValidityError("Первое число ключа и мощность алфавита не взаимно просты")
    result = ''
    for char in text:
        if char in [" ", ".", ","]:
            result += char
            continue
        y = self.alphabet.find(char)
        x = (y - key_b) * self.inverse(key_a)
        result += self.alphabet[x % self.m]
    return result

```

Рисунок 7 — Реализация функций шифрования и расшифрования текста с помощью аффинного шифра

```

Explain Code | Generate Tests | Generate Docstrings | Ask Sourcery
def affine_recurrent_cipher_encrypt(self, text, key_1a, key_1b, key_2a, key_2b):
    key_1a, key_1b, key_2a, key_2b = int(key_1a), int(key_1b), int(key_2a), int(key_2b)

    if self.gcd(key_1a, self.m) != 1 or self.gcd(key_2a, self.m) != 1:
        raise KeyValidityError("Первое число ключа и мощность алфавита не взаимно просты")
    result = ''

    x = self.alphabet.find(text[0])
    y = key_1a * x + key_1b
    result += self.alphabet[y % self.m]
    x = self.alphabet.find(text[1])
    y = key_2a * x + key_2b
    result += self.alphabet[y % self.m]

    prev2_a, prev2_b = key_1a, key_1b #i - 2 элемент
    prev1_a, prev1_b = key_2a, key_2b #i - 1 элемент

    for i in range(2, len(text)):
        if text[i] in [" ", ".", ","]:
            result += text[i]
            continue
        current_a = (prev1_a * prev2_a) % self.m
        current_b = (prev1_b + prev2_b) % self.m
        x = self.alphabet.find(text[i])
        y = current_a * x + current_b
        result += self.alphabet[y % self.m]
        prev2_a, prev2_b = prev1_a, prev1_b
        prev1_a, prev1_b = current_a, current_b

    return result

```

Рисунок 8 — Реализация функции шифрования текста с помощью аффинного рекуррентного шифра

```

Explain Code | Generate Tests | Generate Docstrings | Ask Sourcery
def affine_recurrent_cipher_decrypt(self, text, key_1a, key_1b, key_2a, key_2b):
    key_1a, key_1b, key_2a, key_2b = int(key_1a), int(key_1b), int(key_2a), int(key_2b)

    if self.gcd(key_1a, self.m) != 1 or self.gcd(key_2a, self.m) != 1:
        raise KeyValidityError("Первое число ключа и мощность алфавита не взаимно просты")
    result = ''
    result += self.alphabet[((self.alphabet.find(text[0]) - key_1b) * self.inverse(key_1a)) % self.m]
    result += self.alphabet[((self.alphabet.find(text[1]) - key_2b) * self.inverse(key_2a)) % self.m]

    prev2_a, prev2_b = key_1a, key_1b #i - 2 элемент
    prev1_a, prev1_b = key_2a, key_2b #i - 1 элемент

    for i in range(2, len(text)):
        if text[i] in [" ", ".", ","]:
            result += text[i]
            continue

        current_a = (prev1_a * prev2_a) % self.m
        current_b = (prev1_b + prev2_b) % self.m
        y = self.alphabet.find(text[i])
        x = (y - current_b) * self.inverse(current_a)
        result += self.alphabet[x % self.m]
        prev2_a, prev2_b = prev1_a, prev1_b
        prev1_a, prev1_b = current_a, current_b

    return result

```

Рисунок 9 — Реализация функции расшифрования текста с помощью аффинного рекуррентного шифра

```

if __name__ == "__main__":
    encrypt_or_decrypt = input("Вы хотите зашифровать или расшифровать? 1 - зашифровка, 2 - расшифровка \n")
    res = ''
    if encrypt_or_decrypt == "1":
        encrypt_way = input("Каким способом вы хотите зашифровать текст? \n 1 - шифр простой замены \n 2 - аффинный шифр \n 3 - аффинный рекурентный шифр \n")
        if encrypt_way == "1":
            user_alphabet = input("Введите алфавит: ")
            app = SimpleCiphers(user_alphabet)
            text = input("Текст, который вы хотите зашифровать: ")
            key = input("Введите ключ: ")
            res = app.simple_substitution_encrypt(text, key)
        if encrypt_way == "2":
            user_alphabet = input("Введите алфавит: ")
            app = SimpleCiphers(user_alphabet)
            text = input("Введите текст, который хотите зашифровать ")
            key_a = input("Введите первое число ключа. Оно должно быть взаимно простым с мощностью алфавита ")
            key_b = input("Введите второе число ключа ")
            res = app.affine_cipher_encrypt(text, key_a, key_b)
        if encrypt_way == "3":
            user_alphabet = input("Введите алфавит: ")
            app = SimpleCiphers(user_alphabet)
            text = input("Введите текст, который хотите зашифровать ")
            key_1a = input("Введите первое число первого ключа. Оно должно быть взаимно простым с мощностью алфавита ")
            key_1b = input("Введите второе число первого ключа ")
            key_2a = input("Введите первое число второго ключа. Оно должно быть взаимно простым с мощностью алфавита ")
            key_2b = input("Введите второе число второго ключа ")
            res = app.affine_recurrent_cipher_encrypt(text, key_1a, key_1b, key_2a, key_2b)

```

Рисунок 10 — Реализация главной функции, которая позволяет пользователю выбрать способ шифрования или расшифрования, ввести открытый текст, ключ и алфавит, а также указать имя файла для сохранения результата [часть 1]

```

elif encrypt_or_decrypt == '2':
    encrypt_way = input("Каким способом вы хотите расшифровать текст? \n 1 - шифр простой замены \n 2 - аффинный шифр \n 3 - аффинный рекурентный шифр \n")
    if encrypt_way == "1":
        user_alphabet = input("Введите алфавит: ")
        app = SimpleCiphers(user_alphabet)
        text = input("Шифртекст, который вы хотите расшифровать: ")
        key = input("Введите ключ: ")
        res = app.simple_substitution_decrypt(text, key)
    if encrypt_way == "2":
        user_alphabet = input("Введите алфавит: ")
        app = SimpleCiphers(user.alphabet)
        text = input("Шифртекст, который хотите расшифровать: ")
        key_a = input("Введите первое число ключа. Оно должно быть взаимно простым с мощностью алфавита: ")
        key_b = input("Введите второе число ключа: ")
        res = app.affine_cipher_decrypt(text, key_a, key_b)
    if encrypt_way == "3":
        user.alphabet = input("Введите алфавит: ")
        app = SimpleCiphers(user.alphabet)
        text = input("Шифртекст, который хотите расшифровать: ")
        key_1a = input("Введите первое число первого ключа. Оно должно быть взаимно простым с мощностью алфавита ")
        key_1b = input("Введите второе число первого ключа ")
        key_2a = input("Введите первое число второго ключа. Оно должно быть взаимно простым с мощностью алфавита ")
        key_2b = input("Введите второе число второго ключа ")
        res = app.affine_recurrent_cipher_decrypt(text, key_1a, key_1b, key_2a, key_2b)

if res:
    print(f'{res}')
    save_in_file(res)

```

Рисунок 11 — Реализация главной функции, которая позволяет пользователю выбрать способ шифрования или расшифрования, ввести открытый текст, ключ и алфавит, а также указать имя файла для сохранения результата [часть 2]

```

$ (.venv) dvvd@DebuX:~/Documents/Crypto$ /home/dvvd/Documents/Crypto/.venv/bin/python /home/dvvd/Documents/Crypto/SimpleCiphers/simpleCiphers.py
Вы хотите зашифровать или расшифровать? 1 - зашифровка, 2 - расшифровка
1
Каким способом вы хотите зашифровать текст?
1 - шифр простой замены
2 - аффинный шифр
3 - аффинный рекурентный шифр
1
Введите алфавит: абвгдёйжзийлмнопртсуфхчшщыъэю
Текст, который вы хотите зашифровать: привет
Введите ключ: спарткбвгдёйжзийлмноуфхчшщыъэоя
лидако
Сохранить содержимое вывода в отдельный файл? [y/n]: n
$ (.venv) dvvd@DebuX:~/Documents/Crypto$ 

```

Рисунок 12 — Пример зашифровки шифром простой замены

```

Сохранить содержимое вывода в отдельный файл? [у/н]: н
(.venv) dvvd@DebuX:~/Documents/Crypto$ /home/dvvd/Documents/Crypto/.venv/bin/python /home/dvvd/Documents/Crypto/SimpleCiphers/simpleCiphers.py
Вы хотите зашифровать или расшифровать? 1 - зашифровка, 2 - расшифровка
2
Каким способом вы хотите расшифровать текст?
1 - шифр простой замены
2 - аффинный шифр
3 - аффинный рекуррентный шифр
1
Введите алфавит: абвгдёжзийклмнопрстуфхцчшыъэюя
Шифтекст, который вы хотите расшифровать: лмдако
Введите ключ: спарткбвгдёжзийлмноуфхцчшыъэюя
привет
Сохранить содержимое вывода в отдельный файл? [у/н]: н

```

Рисунок 13 — Пример расшифрования шифра простой замены

```

(.venv) dvvd@DebuX:~/Documents/Crypto$ /home/dvvd/Documents/Crypto/.venv/bin/python /home/dvvd/Documents/Crypto/SimpleCiphers/simpleCiphers.py
Вы хотите зашифровать или расшифровать? 1 - зашифровка, 2 - расшифровка
1
Каким способом вы хотите зашифровать текст?
1 - шифр простой замены
2 - аффинный шифр
3 - аффинный рекуррентный шифр
2
Введите алфавит: абвгдёжзийклмнопрстуфхцчшыъэюя
Введите текст, который хотите зашифровать привет
Введите первое число ключа. Оно должно быть взаимно простым с мощностью алфавита 4
Введите второе число ключа 7
еййօբ
Сохранить содержимое вывода в отдельный файл? [у/н]: н

```

Рисунок 14 — Пример шифрования аффинным шифром

```

(.venv) dvvd@DebuX:~/Documents/Crypto$ /home/dvvd/Documents/Crypto/.venv/bin/python /home/dvvd/Documents/Crypto/SimpleCiphers/simpleCiphers.py
Вы хотите зашифровать или расшифровать? 1 - зашифровка, 2 - расшифровка
2
Каким способом вы хотите расшифровать текст?
1 - шифр простой замены
2 - аффинный шифр
3 - аффинный рекуррентный шифр
2
Введите алфавит: абвгдёжзийклмнопрстуфхцчшыъэюя
Введите шифтекст, который хотите расшифровать: ейյօբ
Введите первое число ключа. Оно должно быть взаимно простым с мощностью алфавита: 4
Введите второе число ключа: 7
привет
Сохранить содержимое вывода в отдельный файл? [у/н]: н

```

Рисунок 15 — Пример расшифрования аффинного шифра

```

(.venv) dvvd@DebuX:~/Documents/Crypto$ /home/dvvd/Documents/Crypto/.venv/bin/python /home/dvvd/Documents/Crypto/SimpleCiphers/simpleCiphers.py
Вы хотите зашифровать или расшифровать? 1 - зашифровка, 2 - расшифровка
1
Каким способом вы хотите зашифровать текст?
1 - шифр простой замены
2 - аффинный шифр
3 - аффинный рекуррентный шифр
3
Введите алфавит: абвгдёжзийклмнопрстуфхцчшыъэюя
Введите текст, который хотите зашифровать кот
Введите первое число первого ключа. Оно должно быть взаимно простым с мощностью алфавита 2
Введите второе число первого ключа 1
Введите первое число второго ключа. Оно должно быть взаимно простым с мощностью алфавита 4
Введите второе число второго ключа 2
цц
Сохранить содержимое вывода в отдельный файл? [у/н]: н

```

Рисунок 16 — Пример ширования аффинным рекуррентным шифром

```

(.venv) dvvd@DebuX:~/Documents/Crypto$ /home/dvvd/Documents/Crypto/.venv/bin/python /home/dvvd/Documents/Crypto/SimpleCiphers/simpleCiphers.py
Вы хотите зашифровать или расшифровать? 1 - зашифровка, 2 - расшифровка
2
Каким способом вы хотите расшифровать текст?
1 - шифр простой замены
2 - аффинный шифр
3 - аффинный рекуррентный шифр
3
Введите алфавит: абвгдёжзийклмнопрстуфхцчшыъэюя
Введите шифтекст, который хотите расшифровать: цц
Введите первое число первого ключа. Оно должно быть взаимно простым с мощностью алфавита 2
Введите второе число первого ключа 1
Введите первое число второго ключа. Оно должно быть взаимно простым с мощностью алфавита 4
Введите второе число второго ключа 2
кот
Сохранить содержимое вывода в отдельный файл? [у/н]: н

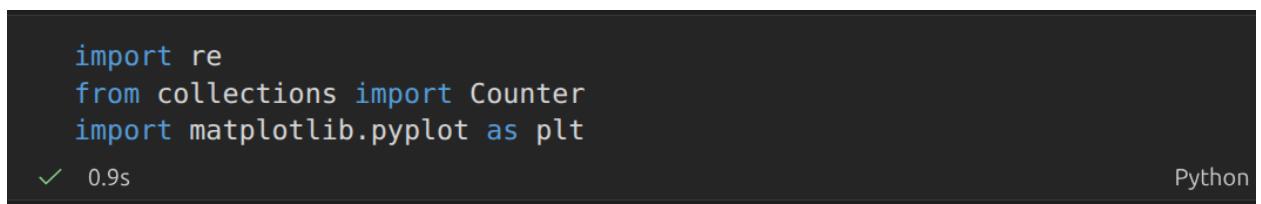
```

Рисунок 17 — Пример расшифрования аффинного рекуррентного шифра.

5 Примеры криptoанализа

Пример криptoанализа шифра простой замены. Из-за вычислительной сложности полного перебора, наиболее эффективным методом криptoанализа шифра простой замены является частотный анализ. Этот метод позволяет сопоставить наиболее часто встречающиеся символы в зашифрованном тексте с наиболее частыми буквами в языке, на котором написан открытый текст. Для этого нужно определить какие буквы в естественном языке встречаются часто, а какие нет. Можно использовать внешние источники информации, но можно и проанализировать большой объем текста на исходном языке. Для этого я взял первый том «Войны и мир» Л.Н. Толстого [1] и проанализировал частоту появления каждой буквы, французский текст переведен на русский.

Я использовал Jupyter Notebook для анализа текста и построения гистограммы частоты появления каждой буквы. Для подсчета букв в тексте я использовал регулярные выражения и класс Counter из модуля collections, который создает мультимножество, в котором одинаковые символы могут встречаться несколько раз. Для построения гистограммы использовал библиотеку Matplotlib (см. Рисунок 18).

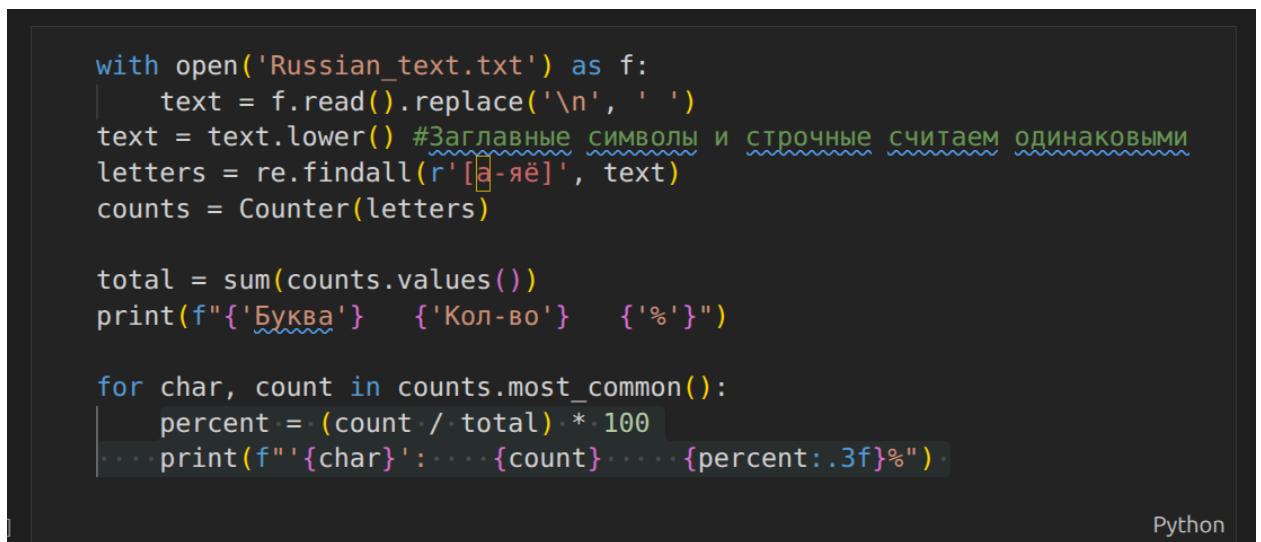


```
import re
from collections import Counter
import matplotlib.pyplot as plt
```

✓ 0.9s Python

Рисунок 18 — Импорт необходимых модулей

Далее был разработан алгоритм подсчета количества каждой буквы в тексте и их процентное соотношение относительно всех букв.



```
with open('Russian_text.txt') as f:
    text = f.read().replace('\n', ' ')
text = text.lower() #Заглавные символы и строчные считаем одинаковыми
letters = re.findall(r'[а-яё]', text)
counts = Counter(letters)

total = sum(counts.values())
print(f"{'Буква'} {'Кол-во'} {'%'}")

for char, count in counts.most_common():
    percent = (count / total) * 100
    print(f"{'{char}'} {'{count}'} {'{percent:.3f}%'})
```

Python

Рисунок 19 — Алгоритм подсчета количества каждой буквы в тексте

...	Буква	Кол-во	%
	'о':	170350	11.440%
	'е':	122691	8.240%
	'а':	122599	8.233%
	'и':	98730	6.630%
	'н':	97847	6.571%
	'т':	86561	5.813%
	'с':	78317	5.260%
	'л':	75534	5.073%
	'в':	67509	4.534%
	'р':	66235	4.448%
	'к':	51504	3.459%
	'д':	45244	3.038%
	'м':	44492	2.988%
	'у':	41408	2.781%
	'п':	38662	2.596%
	'я':	33406	2.243%
	'ъ':	29566	1.986%
	'г':	29329	1.970%
	'ы':	28763	1.932%
	'б':	26480	1.778%
	'з':	25312	1.700%
	'ч':	21693	1.457%
	'й':	17263	1.159%
	'ж':	15497	1.041%
	...		
	'щ':	4237	0.285%
	'ф':	2932	0.197%
	'ъ':	659	0.044%
	'ё':	22	0.001%

Рисунок 20 — Результат алгоритма подсчета количества каждой буквы в тексте

Подобный вывод достаточно неудобен для анализа, поэтому я построил гистограмму, которая наглядно показывает частоту появления каждой буквы (см. Рисунок 16).

```
sorted_items = counts.most_common()
labels = [item[0] for item in sorted_items]
values = [(item[1] / total) * 100 for item in sorted_items]

plt.figure(figsize=(15, 6))

plt.bar(labels, values, color='tan')

plt.title('Частотный анализ текста', fontsize=16)
plt.xlabel('Буква', fontsize=12)
plt.ylabel('Частота (%)', fontsize=12)

plt.show()
```

Рисунок 21 — Код для гистограммы

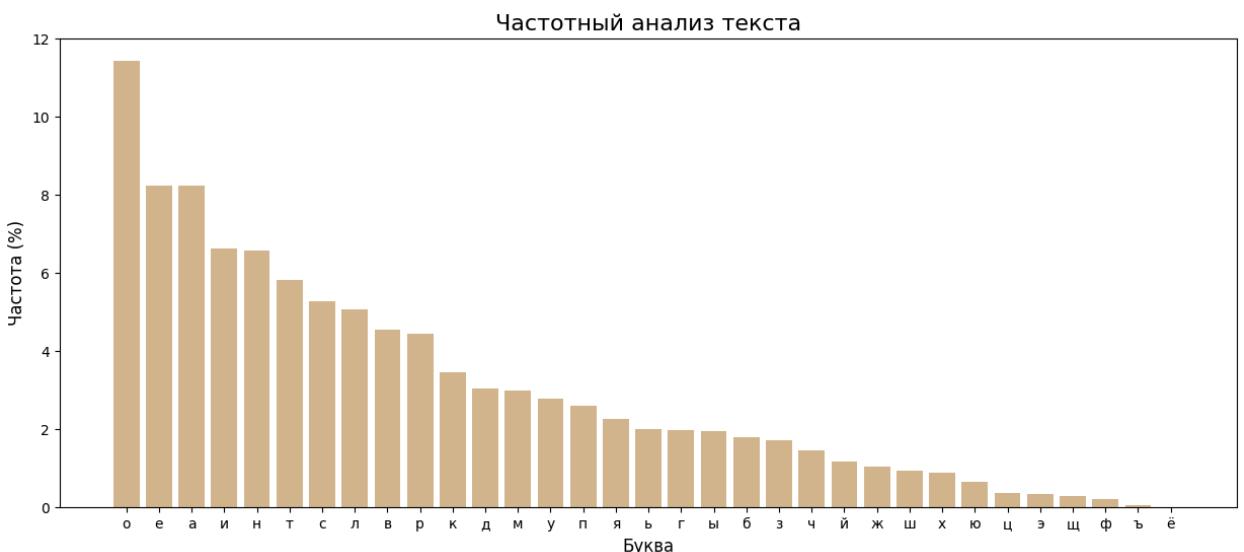


Рисунок 22 — Гистограмма частоты появления каждой буквы в тексте

На гистограмме (см. Рисунок 22) видно, что наиболее часто встречающиеся буквы в тексте — это «о», «е», «а», «и», «н», «т». Поэтому, если в зашифрованном тексте есть символ, который встречается чаще всего, то он вероятно соответствует букве «о». Аналогично, можно сопоставить другие часто встречающиеся символы с наиболее частыми буквами в языке.

Теперь необходимо провести частотный анализ шифротекста. Для этого я зашифровал текст ключом

(а б в г д е ё ж з и й к л м н о п р с т у ф х ц ч ш щ ъ ы ь э ю я)
 (м ф ы э ж щ щ ц л н б ю к ъ я р п а ъ с ч т х д г е в и о з ё у ў ї)

и написал программу для частотного анализа.

```
[27]  with open("cryptotext_simple.txt") as file:  
      text = file.read()  
      letters = re.findall("[а-яё]", text)  
  
      letters_count = Counter(letters)  
      total_count = sum(letters_count.values())  
      print(f"{'Буква'} {'Кол-во'} {'%'})  
      print(letters_count)  
      for char, l_count in letters_count.most_common():  
          percent = (l_count / total_count) * 100  
          print(f'{char}: {l_count} {percent:.3f}%')  
[27]  ✓  0.0s
```

Рисунок 23 — Подсчет количества каждой буквы в шифротексте

```
sorted_items_crypto = letters_count.most_common()  
labels = [item[0] for item in sorted_items_crypto]  
values = [(item[1] / total_count) * 100 for item in sorted_items_crypto]  
  
plt.figure(figsize=(15, 6))  
  
plt.bar(labels, values, color='tan')  
  
plt.title('Частотный анализ шифротекста', fontsize=16)  
plt.xlabel('Буква', fontsize=12)  
plt.ylabel('Частота (%)', fontsize=12)  
  
plt.show()  
[3]  ✓  0.0s
```

Рисунок 24 — Код для построения гистограммы

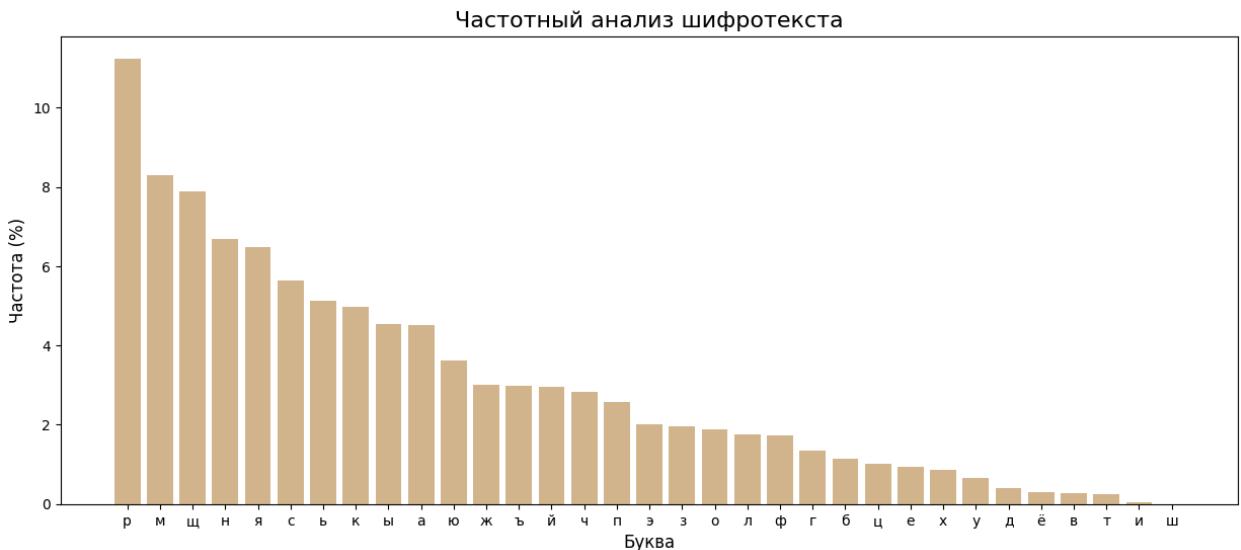


Рисунок 25 — Гистограмма частоты появления каждой буквы в шифротексте

Из этого графика видно, какие буквы чаще встречаются в шифротексте, а какие реже. Из этого можно сделать вывод, что частовстречаемые буквы в исходном тексте перейдут в частовстречаемые буквы в шифротексте. Так, буква «о» перешла в „р“, «ё» в „ш“ и тд. можно восстановить ключ - «мфыэжщцлнбюкъярпьсчтхдгевиозёй» или «щфыэжмшцлнбюкъярпьсчтхдгевиозёй». Установить взаимооднозначное соответствие не удалось из-за того, что в русском языке процент встречаемости букв «а» и «е» примерно одинаковый.



```
from simpleCiphers import SimpleCiphers
from simpleCiphers import save_in_file
alphabet = "а б в г д ё ж з и й к л м н ћ п ћ с т є ф ћ ц ч ћ Ѣ ћ ў ћ є ю ј".replace(" ", "")
cipher = SimpleCiphers(alphabet)
with open("cryptotext_simple.txt") as file:
    file = file.read()
save_in_file(cipher.simple_substitution_decrypt(file, "мфыэжшшилнбюкъярпьсчтхдгевиозёуй"))
save_in_file(cipher.simple_substitution_decrypt(file, "шфыэжмшшилнбюкъярпьсчтхдгевиозёуй"))
```

Рисунок 26 — Код для перебора кандидатов на ключ.

Криптоанализ аффинного шифра. Аффинный шифр остается моноалфавитным шифром подстановки, поэтому каждой букве открытого текста соответствует ровно одна буква шифротекста.

Чтобы провести частотный анализ, необходимо зашифровать текст с помощью аффинного шифра и построить гистограмму.

```

from simpleCiphers import SimpleCiphers
from simpleCiphers import save_in_file
alphabet = "а·б·в·г·д·е·ё·ж·з·и·й·к·л·м·н·о·п·р·с·т·у·ф·х·ц·ч·ш·щ·"
cipher = SimpleCiphers(alphabet)
with open("big_shablon.txt") as file:
    file = file.read()
save_in_file(cipher.affine_cipher_encrypt(file, 4, 7))

```

[31] ✓ 7.4s

Python

```

with open("affine_test.txt") as file:
    text = file.read()
letters = re.findall("[а-яё]", text)

letters_count = Counter(letters)
total_count = sum(letters_count.values())
print(f"{'Буква'} {'Кол-во'} {'%'}")
for char, l_count in letters_count.most_common():
    percent = (l_count / total_count) * 100
    print(f' {char}: {l_count} {percent:.3f}%')

```

[32] ✓ 0.2s

Python

Рисунок 27 — Код для подсчета количества каждой буквы в шифротексте, зашифрованном аффинным шифром

```

sorted_items_crypto = letters_count.most_common()
labels = [item[0] for item in sorted_items_crypto]
values = [(item[1] / total_count) * 100 for item in sorted_items_c

plt.figure(figsize=(15, 6))

plt.bar(labels, values, color='tan')

plt.title('Частотный анализ шифротекста', fontsize=16)
plt.xlabel('Буква', fontsize=12)
plt.ylabel('Частота (%)', fontsize=12)

plt.show()

```

✓ 0.0s

Python

Рисунок 28 — Код для построения гистограммы

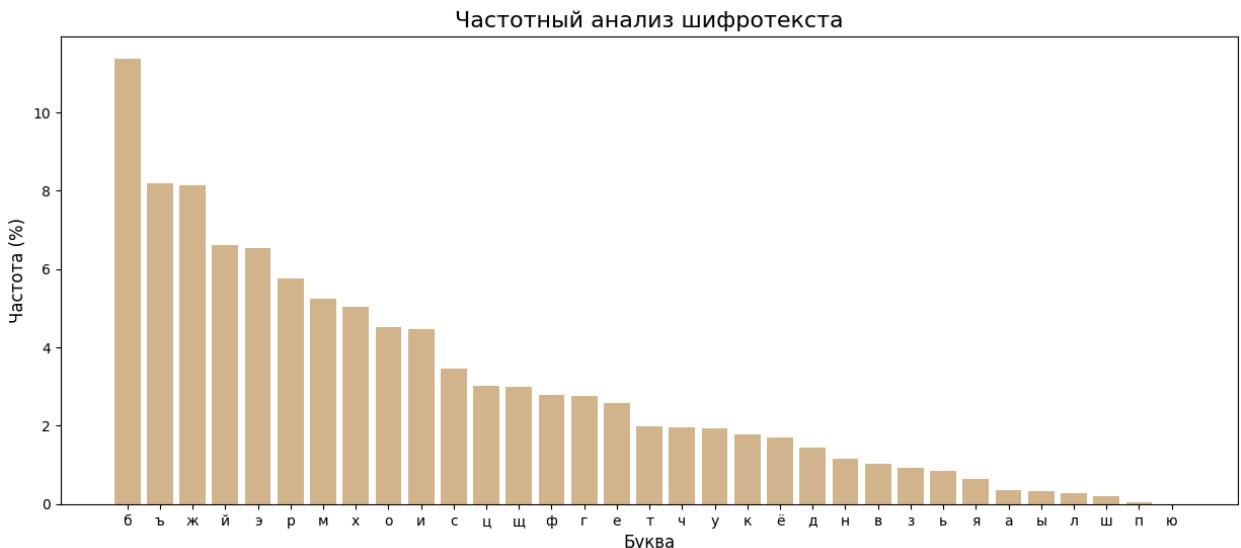


Рисунок 29 — Гистограмма частоты появления каждой буквы в шифротексте, зашифрованном аффинным шифром

Поскольку шифр моноалфавитный, очевидно, что «о» перешла в «б», а «ё» перешла в «ю». Этих знаний достаточно, чтобы вычислить исходный ключ. Решим систему:

$$\begin{cases} 15a + b \equiv 1 \pmod{33} \\ 6a + b \equiv 31 \pmod{33} \end{cases}$$

$$(15a + b) - (6a + b) \equiv 1 - 31 \pmod{33} \quad (10)$$

$$9a \equiv -30 \pmod{33} \Rightarrow 9a \equiv 3 \pmod{33}$$

$$a = 4$$

$$\equiv 31 \pmod{33} \Rightarrow 24 + b \equiv 31 \pmod{33} \Rightarrow b = 7$$

Нам удалось подобрать ключ, теперь достаточно воспользоваться обычной формулой расшифрования (6)

Рисунок 30 — Код для расшифрования текста с использованием найденного ключа

Криптоанализ аффинного рекуррентного шифра. Аффинный рекуррентный шифр является полиалфавитным шифром, что делает его более устойчивым к частотному анализу. Для того, чтобы в этом убедиться, я зашифровал текст и построил гистограмму (см. Рисунок 33).

```
with open("affine_recur_test.txt") as file:  
    text = file.read()  
letters = re.findall("[а-яё]", text)  
  
letters_count = Counter(letters)  
total_count = sum(letters_count.values())  
print(f'{Буква} {Кол-во} {{"%"} }')  
print(letters_count)  
for char, l_count in letters_count.most_common():  
    percent = (l_count / total_count) * 100  
    print(f'{char}: {l_count} {percent:.3f}%')
```

Рисунок 31 — Подсчет количества символов в шифротексте

```
sorted_items_crypto = letters_count.most_common()
labels = [item[0] for item in sorted_items_crypto]
values = [(item[1] / total_count) * 100 for item in sorted_items_crypto]

plt.figure(figsize=(15, 6))

plt.bar(labels, values, color='tan')

plt.title('Частотный анализ шифротекста', fontsize=16)
plt.xlabel('Буква', fontsize=12)
plt.ylabel('Частота (%)', fontsize=12)

plt.show()
```

Рисунок 32 — Код для гистограммы

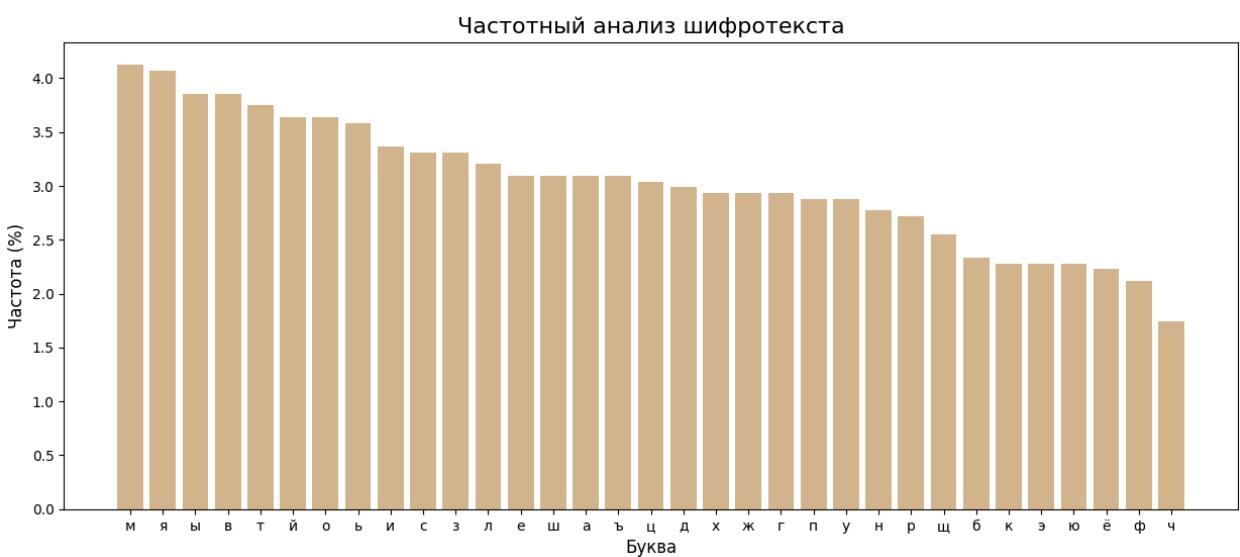
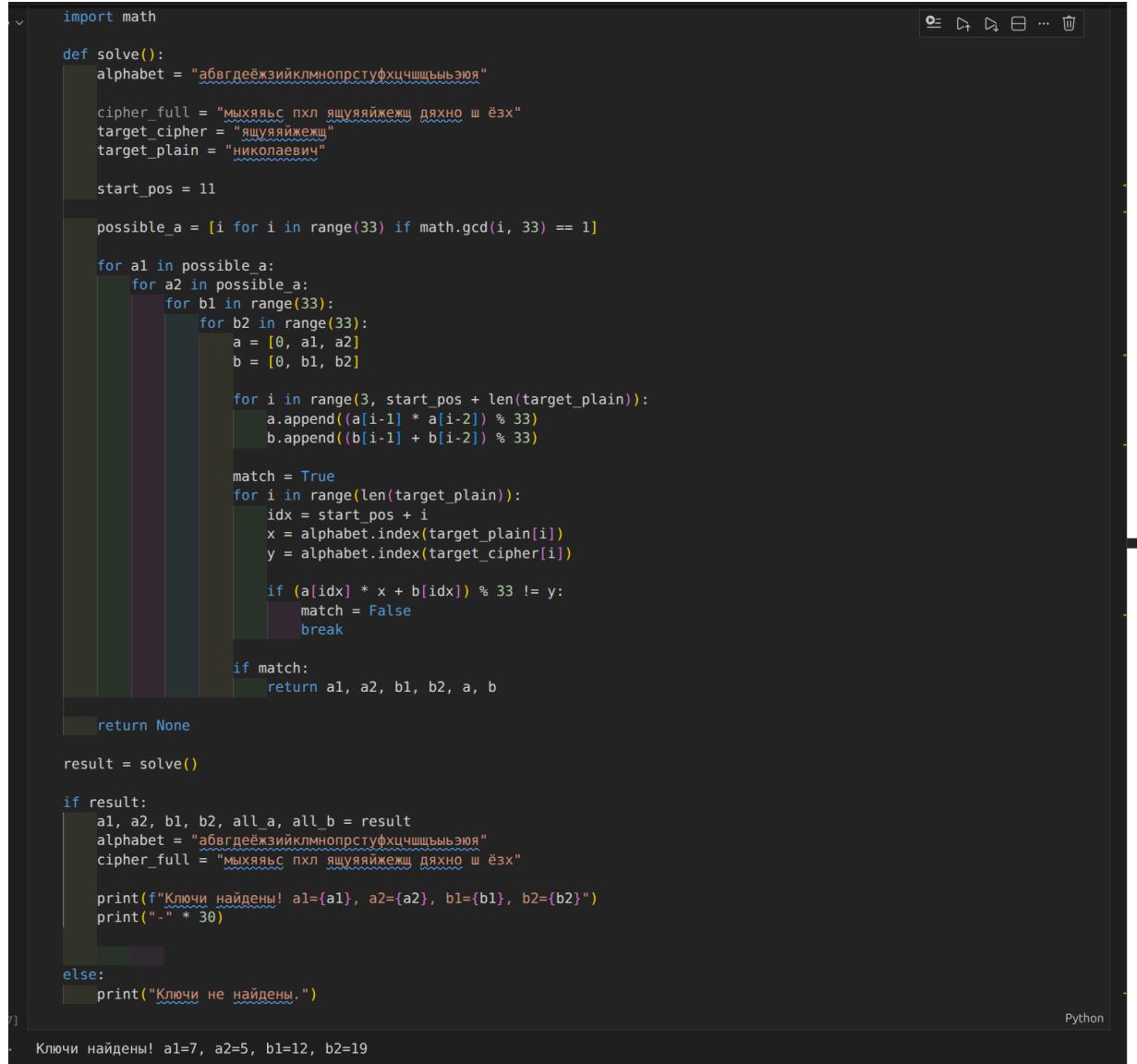


Рисунок 33 — Гистограмма встречаемости букв в шифротексте аффинного рекуррентного шифра

Можно увидеть, что символы встречаются примерно одинаковое количество раз. Это не позволяет провести простой частотный анализ, как в обычном аффинном шифре. Однако, рекуррентный шифр слаб к криптоанализу, основанном на открытом тексте. Допустим, нам

известно, что третье слово шифротекста «ящуяяйжежщ» переводится в открытый текст как «николаевич» и первая буква этого слова стоит на 11 месте. В таком случае можно сделать полный перебор всех возможных ключей и проверить, какой из них будет переводить «ящуяяйжежщ» в «николаевич». При этом, перебор будет достаточно быстрым, так как знакомое нам слово находится почти в начале текста.



```

import math

def solve():
    alphabet = "абвгдеёжзийклмнопрстуфхцчишъэюя"

    cipher_full = "мыхяяьс пхл ящуяяйжежщ дяхно ш ёзх"
    target_cipher = "ящуяяйжежщ"
    target_plain = "николаевич"

    start_pos = 11

    possible_a = [i for i in range(33) if math.gcd(i, 33) == 1]

    for a1 in possible_a:
        for a2 in possible_a:
            for b1 in range(33):
                for b2 in range(33):
                    a = [0, a1, a2]
                    b = [0, b1, b2]

                    for i in range(3, start_pos + len(target_plain)):
                        a.append((a[i-1] * a[i-2]) % 33)
                        b.append((b[i-1] + b[i-2]) % 33)

                    match = True
                    for i in range(len(target_plain)):
                        idx = start_pos + i
                        x = alphabet.index(target_plain[i])
                        y = alphabet.index(target_cipher[i])

                        if (a[idx] * x + b[idx]) % 33 != y:
                            match = False
                            break

                    if match:
                        return a1, a2, b1, b2, a, b

    return None

result = solve()

if result:
    a1, a2, b1, b2, all_a, all_b = result
    alphabet = "абвгдеёжзийклмнопрстуфхцчишъэюя"
    cipher_full = "мыхяяьс пхл ящуяяйжежщ дяхно ш ёзх"

    print(f"Ключи найдены! a1={a1}, a2={a2}, b1={b1}, b2={b2}")
    print("-" * 30)

else:
    print("Ключи не найдены.")

```

Рисунок 34 — Код для перебора ключей на основе открытого текста

Далее можно использовать формулу (6) для расшифрования всего текста.

```

from simpleCiphers import SimpleCiphers
alphabet = "абвгдеёжийклмнопрстуфхүүшшыъэюя"
with open('affine_recur_test.txt') as file:
    cipher_text = file.read()
key_1a = 7
key_1b = 12
key_2a = 5
key_2b = 19
cipher = SimpleCiphers(alphabet)
decrypted_text = cipher.affine_recurrent_cipher_decrypt(
    cipher_text,
    key_1a,
    key_1b,
    key_2a,
    key_2b
)

print("Расшифрованный текст:")
print(decrypted_text)

```

[66] Python

... Расшифрованный текст:
толстой лев николаевич война и мир. первый вариант романа .классика регистрация найти рейтинги обсуждения новинки обзоры помо

Рисунок 35 — Код для расшифрования текста с помощью найденного ключа

Если же открытого текста нет, то необходимо дешифровать первые 20-30 символов шифротекста полным перебором, а затем воспользоваться криptoанализом на основе открытого текста.

```

import math
from simpleCiphers import SimpleCiphers
from simpleCiphers import save_in_file
alphabet = "абвгдеёжийклмнопрстуфхүүшшыъэюя"
cipher = SimpleCiphers(alphabet)
with open('affine_recur_test.txt') as file:
    cipher_text = file.read()
cipher_text = cipher_text[:15]
m = 33
possible_a = [i for i in range(33) if math.gcd(i, 33) == 1]
all =""
for a1 in possible_a:
    for a2 in possible_a:
        for b1 in range(33):
            for b2 in range(33):
                decrypted_text = cipher.affine_recurrent_cipher_decrypt(
                    cipher_text,
                    a1,
                    b1,
                    a2,
                    b2
                )
                all += decrypted_text + "\n"
print(a1)
save_in_file(all)

```

Python

Рисунок 36 — Код для перебора ключей без открытого текста

Полный перебор 15 символов занял у меня примерно 3 минуты, а перебор на основе открытого текста был почти мгновенным.

6 Выводы о проделанной работе

Краткие выводы о проделанной работе: достоинства и недостатки исследуемых шифров, ограничения выбранных методов криптоанализа, наиболее эффективные сценарии криптоанализа.

Достоинства шифра простой замены. Он довольно прост в ручной реализации и невозможен для перебора даже на самых современных компьютерах, так как количество ключей равно $m!$, что для русского алфавита составляет $33!$. *Недостатки шифра простой замены.* Он является моноалфавитным шифром, что делает его уязвимым к частотному анализу. Частотный анализ позволяет сопоставить большую часть символов зашифрованного текста с символами открытого текста, что позволяет восстановить ключ и расшифровать текст, однако если в естественном языке буквы встречаются с примерно одинаковой частотой, то появится больше различных ключей, однако, перебрать их будет несложно.

Достоинства аффинного шифра. Он также прост в ручной реализации. *Недостатки аффинного шифра.* Он является самым уязвимым к криптоанализу из всех трех рассмотренных шифров, так как имеет малое количество различных ключей, что делает его простым для перебора, а так же он является моноалфавитным шифром, что делает его уязвимым к частотному анализу. Причем, подобрать ключ можно взаимно однозначно сопоставив всего две буквы, поэтому ограничений у метода криптоанализа нет.

Достоинства аффинного рекуррентного шифра. Среди рассмотренных шифров он является самым устойчивым к криптоанализу, так как является полиалфавитным шифром, он способен зашифровать одинаковые символы разными буквами и наоборот. *Недостатки аффинного рекуррентного шифра.* Он является самым сложным для ручной реализации, а так же он уязвим к криптоанализу на основе открытого текста, который позволяет подобрать глобальные ключи почти мгновенно. Ограничением данного метода криптоанализа является отсутствие открытого текста, в таком случае необходимо провести полный перебор первых 10-30 символов, который займет от 3-10 минут (в зависимости от длины шифротекста).

7 Список используемые источников

1. Атака на основе открытых текстов [Электронный ресурс] : Материал из Википедии — свободной энциклопедии : Версия 147353010, сохранённая в 20:44 UTC 4 августа 2025 / Авторы Википедии // Википедия, свободная энциклопедия. — Электрон. дан. — Сан-Франциско: Фонд Викимедиа, 2025. — Режим доступа: <https://ru.wikipedia.org/?curid=2565041&oldid=147353010>