**ОТЧЕТ К ЛАБОРАТОРНОЙ РАБОТЕ № 3**

**Лабораторная работа № 3**

**Криптоанализ аффинного шифра**

**Вариант №5**

Ф. И. О. студента: Гюнтер Тимофей Вячеславович

Группа: ФИТ-221

Проверил: Дата:

**Основные сведения**

Формула для зашифрования текста: Формула для расшифрования текста:

**Результаты**

ШИФР-ТЕКСТ (ШТ):

Результаты частотного анализа ШТ:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| буква | а | б | в | г | | д | е/ё | ж | з | и | й |
| частота | 0 | 0.021 | 0.014 | 0.034 | 0.021 | | 0 | 0.027 | 0.062 | 0.116 | 0 |
| буква | к | л | м | н | о | | п | р | с | т | у |
| частота | 0.021 | 0.082 | 0.014 | 0.041 | 0.014 | | 0 | 0.054 | 0.014 | 0.027 | 0 |
| буква | ф | х | ц | ч | ш | | щ | ъ | ы | ь | э |
| частота | 0.021 | 0.027 | 0.021 | 0 | 0.014 | | 0.11 | 0.021 | 0.082 | 0 | 0.068 |
| буква | ю | я |
| частота | 0.048 | 0.027 |

Наиболее часто встречающиеся символы:

и, щ

Система уравнения верного ключа:

Решения систем уравнений: a = 9, b = 27

ВЕРНЫЙ КЛЮЧ: (9, 27)

РАСШИФРОВАННЫЙ ТЕКСТ (ОТ):

тамкоролевичмимоходомпленяетгрозногоцарятамвоблакахпереднародомчерезлесачерезморяколдуннесетбогатырявтемницетамцаревнатужитабурыйволкейвернослужит

**Код программы**

def euclid\_extended(m, a):

    if a == 0: return m, 1, 0

    d, x, y = euclid\_extended(a, m%a)

    x, y = y, x - m//a\*y

    return d, x, y

def inverse\_element\_interface():

    user\_input = input("Введите элемент и модуль (через пробел), в кольце которого желаете найти обратный элемент: \n")

    a, m = map(int, user\_input.split())

    d, \_, inverse\_a = euclid\_extended(m, a%m)

    inverse\_a %= m

    if d != 1:

        print("Приносим свои извинения обратного элемента не существует...")

    print(f'Обратный элемент найден: {inverse\_a}\n')

# inverse\_element\_interface()

def comparison\_solver(a, b, m):

    gcd, \_, y = euclid\_extended(m, a )

    if gcd != 1 and b % gcd == 0:

        # print("Обратного элемента не существует!")

        # print(f"Решение имеет в точности {gcd} решений!")

        b //= gcd

        a //= gcd

        new\_m = m // gcd

        new\_gcd, \_, y = euclid\_extended(new\_m, a % new\_m)

        inverse\_a = y

        pair\_ans = []

        for k in range(gcd):

            # print(new\_betta \* inverse\_a, k)

            a = (b \* inverse\_a % new\_m) + new\_m\*k

            # b = (betta\_1 - alpha\_1\*a) % m

            pair\_ans.append(a)

        return pair\_ans

    if gcd != 1 and b % gcd != 0:

        # print("Обратного элемента не существует!")

        # print("Сравнение неразрешимо!")

        return None

    # print("Решение единственно!")

    inverse\_a = y

    a = b \* inverse\_a % m

    # b = (betta\_1 - alpha\_1\*a) % m

    return a

def comparison\_solver\_interface():

    user\_input = input("Введите a, b и m (через пробел): \n")

    a, b, m = map(int, user\_input.split())

    answer = comparison\_solver(a, b, m)

    if not answer:

        print("Приносим свои извинения решения не существует...")

    print(f'Решение найдено: {answer}\n')

def comparison\_system\_solver\_interface():

    user\_input = input("Введите a, b, с, d и m (через пробел): \n")

    a, b, c, d, m = map(int, user\_input.split())

    answer = comparison\_system\_solver(a, b, c, d, m)

    if not answer:

        print("Приносим свои извинения решения не существует...")

    print(f'Решение найдено: {answer}\n')

# print(comparison\_solver(9, 6, 12))

def comparison\_system\_solver(alpha, betta, alpha\_1, betta\_1, m=32):

    new\_alpha = (alpha\_1 - alpha) % m

    new\_betta = (betta\_1 - betta) % m

    gcd, \_, y = euclid\_extended(m, new\_alpha )

    if gcd != 1 and new\_betta % gcd == 0:

        # print("Обратного элемента не существует!")

        # print(f"Решение имеет в точности {gcd} решений!")

        new\_betta //= gcd

        new\_alpha //= gcd

        new\_m = m // gcd

        new\_gcd, \_, y = euclid\_extended(new\_m, new\_alpha % new\_m)

        inverse\_a = y

        pair\_ans = []

        for k in range(gcd):

            # print(new\_betta \* inverse\_a, k)

            a = (new\_betta \* inverse\_a % new\_m) + new\_m\*k

            b = (betta\_1 - alpha\_1\*a) % m

            pair\_ans.append((a,b))

        return pair\_ans

    if gcd != 1 and new\_betta % gcd != 0:

        # print("Обратного элемента не существует!")

        # print("Сравнение неразрешимо!")

        return None

    # print("Решение единственно!")

    inverse\_a = y

    a = new\_betta \* inverse\_a % m

    b = (betta\_1 - alpha\_1\*a) % m

    return a, b

print('------------------------------------------------------------')

from lab1 import Caesar, Encoding

# from menu import Menu

preprocessing = lambda x: Caesar.preprocessing(x)

def create\_frequent\_dict(message):

    preproc\_message = preprocessing(message)

    n = len(message)

    alphabet = set(preproc\_message)

    freq\_dict = {letter:0 for letter in alphabet}

    for letter in preproc\_message:

        freq\_dict[letter] += preproc\_message.count(letter)/n

    return freq\_dict

def decrypt(message):

    encoding = Encoding(message)

    preproc\_message = preprocessing(message)

    frequent\_dict = create\_frequent\_dict(message)

    m = 32

    # two assumptions: и (5), о (14)

    # let's say that again but in terms of comparisons system:

    # zero assumption

    def find\_two\_most\_frequent\_letters(frequent\_dict):

        copy\_dict = {key: value for key, value in frequent\_dict.items()}

        first\_max = max(copy\_dict, key=copy\_dict.get)

        copy\_dict.pop(first\_max)

        second\_max = max(copy\_dict, key=copy\_dict.get)

        first\_max\_code = encoding.letters\_to\_code(first\_max)[0]

        second\_max\_code = encoding.letters\_to\_code(second\_max)[0]

        return first\_max\_code, second\_max\_code

    first\_max\_code, second\_max\_code = find\_two\_most\_frequent\_letters(frequent\_dict)

    def alpha\_guesser(alpha, alpha\_1):

        betta, betta\_1 = first\_max\_code, second\_max\_code

        solution = comparison\_system\_solver(alpha, betta, alpha\_1, betta\_1)

        if solution:

            a, b = solution

        else:

            # print("Ищите другие буквы!")

            return 0

        d, \_, inverse\_a = euclid\_extended(32, a)

        assert d == 1

        coded\_message = encoding.letters\_to\_code(preproc\_message)

        decrypted\_message = [0 for \_ in preproc\_message]

        answer = [el for el in preproc\_message]

        for index in range(len(coded\_message)):

            decrypted\_message[index] = inverse\_a\*(coded\_message[index] - b ) % m

            answer[index] = encoding.code\_to\_letters(index, decrypted\_message[index])

        return ''.join(answer)

    from itertools import product

    frequency\_table = {

        'о': 0.090, 'е': 0.072, 'а': 0.062,

        'и': 0.062, 'т': 0.053, 'н': 0.053, 'с': 0.045,

        'р': 0.040, 'в': 0.038, 'л': 0.035, 'к': 0.028,

        'м': 0.026, 'д': 0.025, 'п': 0.023, 'у': 0.021,

        'я': 0.018, 'ы': 0.016, 'з': 0.016, 'ъ': 0.014,

        'б': 0.014, 'г': 0.013, 'ч': 0.012, 'й': 0.010,

        'ч': 0.009, 'ж': 0.007, 'ю': 0.006, 'ш': 0.006,

        'ц': 0.004, 'щ': 0.003, 'э': 0.003, 'ф': 0.002,

    }

    # alpha, alpha\_1 = encoding.letters\_to\_code('е')[0], encoding.letters\_to\_code("и")[0]

    # betta, betta\_1 = first\_max\_code, second\_max\_code

    # result = alpha\_guesser(alpha, alpha\_1)

    # print(result)

    def write\_to\_file(file, letter, letter\_1, result):

        file.write(f'key: ({letter}, {letter\_1}); text: {result}\n')

        file.write('----------------------------------------------------\n')

    c = 0

    answer = ''

    # !! ANSWER: е, о

    file = open("files/lab2\_decryption\_tries.txt", 'w', encoding='utf-8')

    for letter, letter\_1 in product(frequency\_table.keys(), repeat=2):

        # print(letter, letter\_1)

        c += 1

        alpha, alpha\_1 = encoding.letters\_to\_code(letter)[0], encoding.letters\_to\_code(letter\_1)[0]

        betta, betta\_1 = first\_max\_code, second\_max\_code

        result = alpha\_guesser(alpha, alpha\_1)

        if result != 0:

            print(f'{c}: {letter}, {letter\_1}, {result}')

            write\_to\_file(file, letter, letter\_1, result)

            answer = input("Вывод имеет смысл? (да/нет): \n")

            print('------------------------------------')

        if answer.lower() == "да": break

        if letter != letter\_1:

            alpha, alpha\_1 = encoding.letters\_to\_code(letter\_1)[0], encoding.letters\_to\_code(letter)[0]

            # betta, betta\_1 = first\_max\_code, second\_max\_code

            result = alpha\_guesser(alpha, alpha\_1)

            if result != 0:

                print(f'{c}: {letter\_1}, {letter}, {result}')

                write\_to\_file(file, letter\_1, letter, result)

                answer = input("Вывод имеет смысл? (да/нет): \n")

                print('------------------------------------')

            if c == 100 or answer.lower() == "да": break

    file.close()

def variant\_case\_decryption():

# message = input("Введите фразу: ")

    message = 'эызхщлщюингкзгзщшщящзвюиртиэцлщърщцщбылтэызнщдюыхышви\

лиярылщящзкилиъюифыкилиъзщлтхщюяжррифиэдщцыэолтнэизргбиэыз\

былинрыэжсгэыджломнщюхимнилрщфюжсгэ'

    decrypt(message)

def input\_message\_decryption():

    message = input("Введите сообщения для расшифрования: ")

    decrypt(message)

# Menu()