МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра информационных систем и технологий

Специальность Информационные системы и технологии

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2 НА ТЕМУ:**

**Исследование криптографических шифров на основе подстановки (замены) символов**

Выполнила студентка 3 курса 1 группы

Пригодич Вера Валерьевна

Минск 2023

**Цель:** изучение и приобретение практических навыков разработки и использования приложений для реализации подстановочных шифров.

**Теоретические сведения**

Криптоанализ – это раздел криптологии, занимающийся методами взлома шифров или методами организации криптографических атак на шифры.

Сущность подстановочного шифрования состоит в том, что исходный текст (из множества *М*) и зашифрованный текст (из множества *С*) основаны на использовании одного и того же или разных алфавитов, а тайной или ключевой информацией является алгоритм подстановки.

Существуют следующие типы подстановочных шифров:

* моноалфавитные (шифры однозначной замены или простые подстановочные);
* полиграммные;
* омофонические (однозвучные шифры или шифры многозначной замены);
* полиалфавитные.

В моноалфавитных шифрах операция замены производится раздельно над каждым одиночным символом сообщения *Мi*. Примерами являются шифр Цезаря, шифр Цезаря с ключевым словом (лозунгом), Атбаш.

К полиграммным шифрам относят шифры Порты, Плейфера и Хилла. В таких шифрах одна подстановка соответствует сразу нескольким символам исходного текста.

Омофонические шифры (омофоническая замена), или однозвучные шифры подстановки, создавались с целью увеличить сложность частотного анализа шифртекстов путем маскировки реальных частот появления символов текста с помощью омофонии. К ним относится книжный шифр.

Полиалфавитные (или многоалфавитные) шифры состоят из нескольких шифров однозначной замены. Выбор варианта алфавита для зашифрования одного символа зависит от особенностей метода шифрования. Примерами являются диск Альберти, таблица Трисемуса, шифр Виженера.

**Задание 1:** Разработать авторское приложение в соответствии с целью лабораторной работы. Приложение должно реализовывать следующие операции:

* выполнять зашифрование/расшифрование текстовых документов (объемом не менее 5 тысяч знаков), созданных на основе алфавита языка в соответствии с нижеследующей таблицей вариантов задания; при этом следует использовать шифры подстановки из третьего столбца данной таблицы;

|  |  |  |
| --- | --- | --- |
| Вариант | Алфавит | Шифр |
| 8 | Английский | 1. Шифр Цезаря с ключевым словом, ключевое слово – собственная фамилия, *а* = 24  2. Таблица Трисемуса, ключевое слово – собственное имя |

Для **системы шифрования Цезаря с ключевым словом (лозунгом)** формируется алфавит подстановки по следующему принципу: ключевое слово записывается под буквами алфавита, начиная с буквы, индекс которой совпадает с выбранным числом *a*. исходный алфавит и алфавит подстановки с ключом 24 выглядят следующим образом:

А B C D Е F G Н I J К L М N O Р Q R S T U V W Х Y Z

E F I J К L М N Q S T U V W Х Y Z P R G O D I C H A B

Повторяющаяся буква I была удалена. Шифрования проводится путем замены буквы сообщения из исходного алфавита на букву, соответствующую ее индексы в новом алфавите.

Например, если сообщение *Mi*= «*VERA*», *Ci* = «*DLPE*».

Расшифрование сообщения проводится аналогичным образом. Для выполнения задания был использован код на языке *python*. Опишем функции для работы с шифром Цезаря caesar\_cipher.py:

alphabet = list('abcdefghijklmnopqrstuvwxyz')

def remove(keyword):

    keyword = list(keyword)

    for character in keyword:

        # удаляем повторяющиеся символы в ключевом слове

        if keyword.count(character) > 1:

           keyword.remove(character)

        # удаляем символы ключевого слова из алфавита

        if character in alphabet:

            alphabet.remove(character)

    return keyword, alphabet

def insert(keyword, key):

    keyword\_alphabet = remove(keyword)

    for index, character in enumerate(keyword\_alphabet[1]):

        keyword\_alphabet[0].insert((key + index)%26, character)

    return keyword\_alphabet[0]

def caesar\_encrypt(keyword: str, key: int, text: str):

    res = ''

    alphabet\_list = list('abcdefghijklmnopqrstuvwxyz')

    keyword = remove(keyword.lower())[0]

    print('\nНормализованное ключевое слово: ',  ''.join(keyword))

    new\_alphabet = insert(keyword, key)

    print('\nИсходный алфавит:', alphabet\_list)

    print('\nАлфавит с ключем:', new\_alphabet)

    for character in text:

            if character in alphabet\_list:

                res += new\_alphabet[alphabet\_list.index(character)]

            else:

                res += character

    return res

def caesar\_decrypt(keyword: str, key: int, text: str):

    res = ''

    alphabet\_list = list('abcdefghijklmnopqrstuvwxyz')

    new\_alphabet = insert(keyword, key)

    for character in text:

            if character in new\_alphabet:

                res += alphabet\_list[new\_alphabet.index(character)]

            else:

                res += character

    return res

Листинг 1.1 – caesar\_cipher.py

Шифрование с использованием **таблиц Трисемуса** проводится следующим образом: формируется таблица определенного размера, в которую вписывается по строкам ключевое слово, причем повторяющиеся буквы отбрасываются. Затем эта таблица дополняется не вошедшими в нее буквами алфавита по порядку. Было использовано имя в качестве ключа, таблица была дополнена знаками «?;,.» и сформирована таблица 5×6.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| v | e | r | a | b |
| c | d | f | g | h |
| i | j | k | l | m |
| n | o | p | q | S |
| t | u | w | x | y |
| z | ? | ; | , | . |

При шифровании находят в этой таблице букву открытого текста и записывают в шифротекст букву, расположенную ниже нее в том же столбце. Если буква текста оказывает в нижней строке таблицы, тогда для шифротекста берут самую верхнюю букву из того же столбца.

Например, если сообщение *Mi*= «*hello*», *Ci* = «*mdqqu*».

Расшифрование проводится аналогичным образом: находят в этой таблице букву шифротекста и записывают в расшифрованный текст букву, расположенную выше нее в том же столбце. Если буква текста оказывает в верхней строке таблицы, тогда для шифротекста берут самую нижнюю букву из того же столбца.

Функции для работы с таблицей Трисемуса описаны в файле trithemius\_cipher.py:

import numpy as np

alphabet = list('abcdefghijklmnopqrstuvwxyz')

def remove(keyword : str):

    keyword = list(keyword)

    for character in keyword:

        # удаляем повторяющиеся символы в ключевом слове

        if keyword.count(character) > 1:

           keyword.remove(character)

        # удаляем символы ключевого слова из алфавита

        if character in alphabet:

            alphabet.remove(character)

    return keyword, alphabet

def insert(keyword: str):

    keyword\_alphabet = remove(keyword)

    for character in keyword\_alphabet[1]:

        keyword\_alphabet[0].append(character)

    return keyword\_alphabet[0]

def form\_table(keyword : str):

    table = insert(keyword)

    additional\_chars = ['?', ';', ',', '.']

    table = table + additional\_chars

    table = np.reshape(table, (6,5))

    return table

def trithemius\_encrypt(keyword : str, text : str):

    res = ''

    new\_alphabet = form\_table(keyword)

    for row in new\_alphabet:

        print(row)

    for character in text:

        if character in new\_alphabet:

            i, j = np.where(new\_alphabet == character)

            if i == 5:

                encoded\_character = ''.join(str(x) for x in new\_alphabet[0, j])

                res += encoded\_character

            else:

                encoded\_character = ''.join(str(x) for x in new\_alphabet[i+1, j])

                res += encoded\_character

        else:

            res += ' '

    return res

def trithemius\_decrypt(keyword: str, text: str):

    res = ''

    new\_alphabet = form\_table(keyword)

    for character in text:

        if character in new\_alphabet:

            i, j = np.where(new\_alphabet == character)

            if i == 0:

                encoded\_character = ''.join(str(x) for x in new\_alphabet[5, j])

                res += encoded\_character

            else:

                encoded\_character = ''.join(str(x) for x in new\_alphabet[i-1, j])

                res += encoded\_character

        else:

            res += ' '

    return res

Листинг 1.2 – trithemius\_cipher.py

* сформировать гистограммы частот появления символов для исходного и зашифрованного сообщений;
* оценить время выполнения операций зашифрования/расшифрования;

Применим разработанные функции в файле lab2.py. Также пропишем в конце код для формирования гистограмм частот символов, используем функции для отсчета времени выполнения операций. В качестве источника используется файл text.txt, содержащие текст на английском языке.

from caesar\_cipher import \*

from trithemius\_cipher import \*

import matplotlib.pyplot as plt

from datetime import datetime

a = 24

caesar\_keyword = 'prigodich'

trithemius\_keyword = 'vera'

with open('text.txt',  encoding='utf8') as file:

    text = file.read().lower()

print('Шифр Цезаря')

start\_time = datetime.now()

caesar\_encrypted = caesar\_encrypt(caesar\_keyword, a, text)

encrypt\_time = datetime.now() - start\_time

print('\n\nЗашированное сообщение:\n\n')

print(caesar\_encrypted)

print('\n\nРасшированное сообщение:\n\n')

start\_time = datetime.now()

print(caesar\_decrypt(caesar\_keyword, a, caesar\_encrypted))

decrypt\_time = datetime.now() - start\_time

print('\nВремя зашифрования:', encrypt\_time)

print('Время расшифрования:', decrypt\_time)

print('\n\n\nШифр Трисемуса')

start\_time = datetime.now()

trithemius\_encrypted = trithemius\_encrypt(trithemius\_keyword, text)

encrypt\_time = datetime.now() - start\_time

print('\n\nЗашированное сообщение:\n\n')

print(trithemius\_encrypted)

print('\n\nРасшированное сообщение:\n\n')

start\_time = datetime.now()

print(trithemius\_decrypt(trithemius\_keyword, trithemius\_encrypted))

decrypt\_time = datetime.now() - start\_time

print('\nВремя зашифрования:', encrypt\_time)

print('Время расшифрования:', decrypt\_time)

# словарь {символ:количество потворений этого символа} отсортированный по ключу

def get\_letters\_amount(seq):

    letters\_dictionary = {}

    for i in seq:

        if i.isalpha():

            if i not in letters\_dictionary:

                letters\_dictionary[i] = 0

            letters\_dictionary[i] += 1

    return dict(sorted(letters\_dictionary.items()))

text\_probs = get\_letters\_amount(text)

encryptes\_probs\_caesar = get\_letters\_amount(caesar\_encrypted)

encryptes\_probs\_trithemius = get\_letters\_amount(trithemius\_encrypted)

#строим гистограммы по словарям

fig, a = plt.subplots(2,2, figsize=(12, 10))

a[0][0].set\_title('Исходное соо')

a[0][0].bar(list(text\_probs.keys()), text\_probs.values(), color='g')

a[0][1].set\_title('Шифр Цезаря')

a[0][1].bar(list(encryptes\_probs\_caesar.keys()), encryptes\_probs\_caesar.values(), color='b')

a[1][0].set\_title('Шифр Трисемуса')

a[1][0].bar(list(encryptes\_probs\_trithemius.keys()), encryptes\_probs\_trithemius.values(), color='r')

a[1][1].set\_title('Трисемуса и Цезаря')

a[1][1].bar(list(encryptes\_probs\_caesar.keys()), encryptes\_probs\_caesar.values(), color='r')

a[1][1].bar(list(encryptes\_probs\_trithemius.keys()), encryptes\_probs\_trithemius.values(), color='b')

plt.show()

Листинг 1.3 – lab2.py

Результат работы итогового кода:

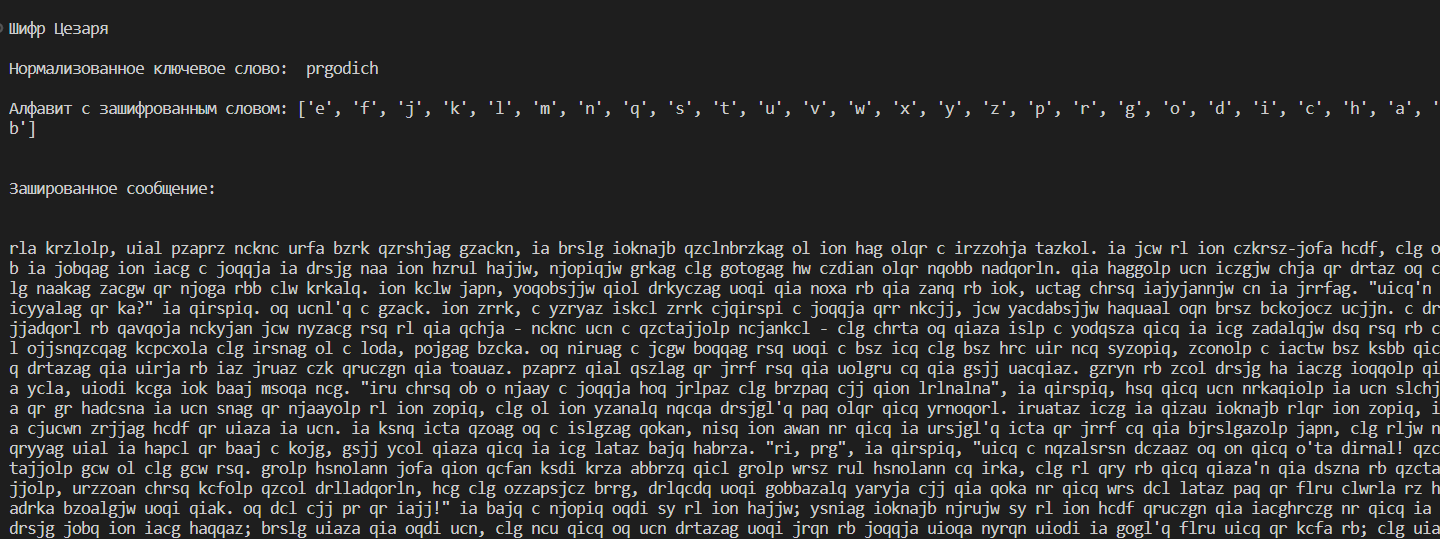


Рисунок 1.1 ­­­­– Результат работы программы. Часть 1

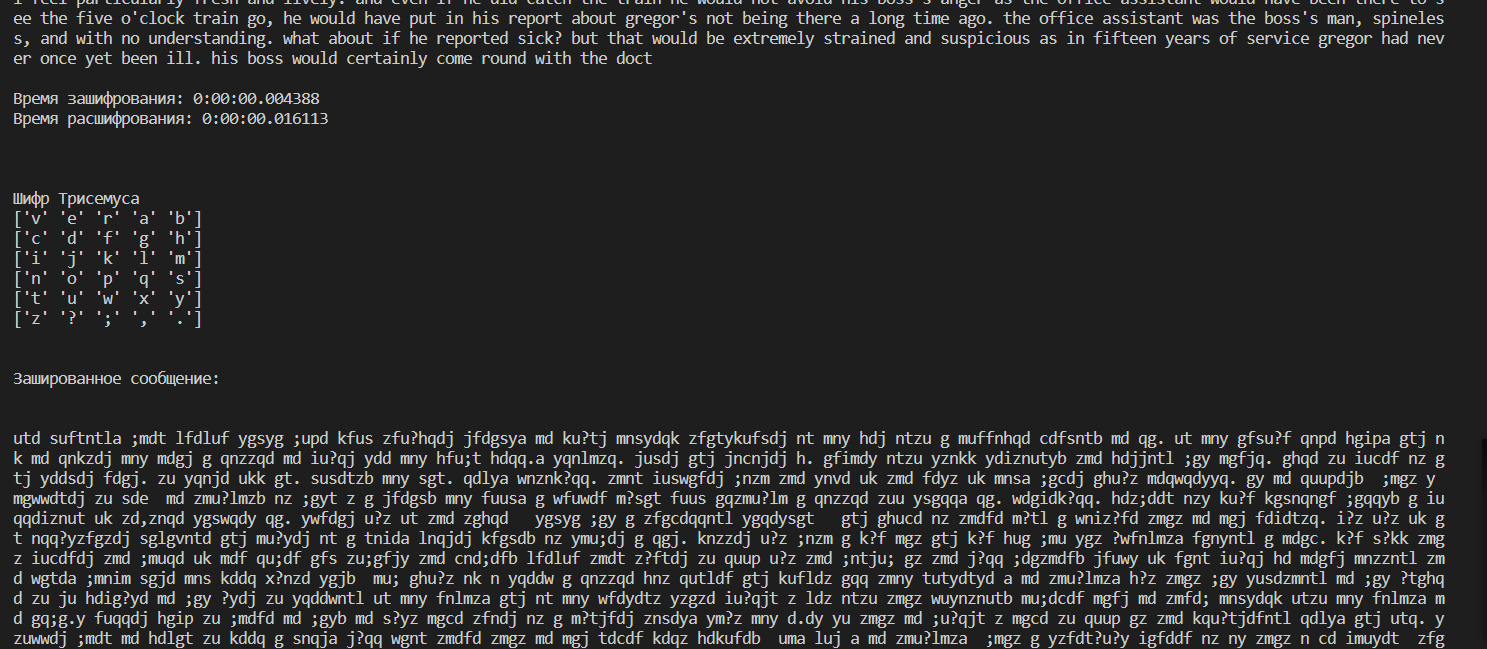


Рисунок 1.2 ­­­­– Результат работы программы. Часть 2

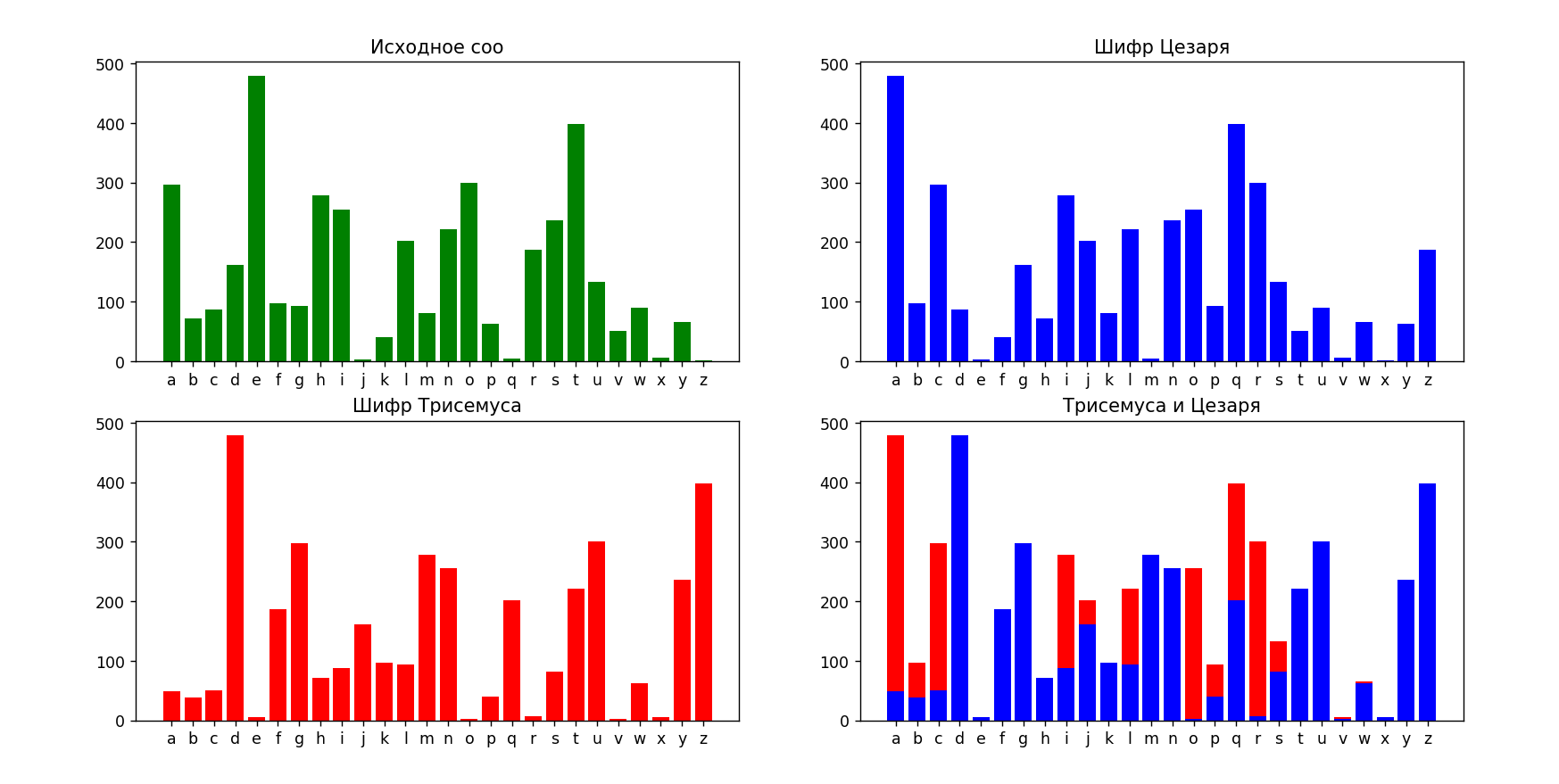


Рисунок 1.3 – Гистограмма частот символов в исходном тексте и в зашифрованном

Время шифрования/расшифрования для шифра Цезаря:

Время шифрования: 0:00:00.005000

Время расшифрования: 0:00:00.008999

Время шифрования/расшифрования для таблиц Трисемуса:

Время шифрования: 0:00:00.077104

Время расшифрования: 0:00:00.108659

Анализируя гистограммы частот можно заметить, что если в открытом сообщении часто встречается какая-либо буква, то в шифрованном сообщении также часто будет встречаться соответствующий ей символ. Так, можно предположить, что буква *e* переходит в букву *d* при использовании таблицы Трисемуса. Посмотрев на сформированную таблицу, можно сказать, что данный вывод является справедливым.

Заметно, что разработанный алгоритм шифра Цезаря работает в разы быстрее алгоритма таблиц Трисемуса. Время декодирования для обоих алгоритмов оказалось больше времени кодирования.

**Контрольные вопросы:**

1. **В чем заключается основная идея криптографических преобразований на основе шифров замены?**

Сущность преобразований на основе шифров замены состоит в том, что исходный текст (из множества М) и зашифрованный текст (из множества С) основаны на использовании одного и того же или разных алфавитов, а тайной или ключевой информацией является алгоритм подстановки.

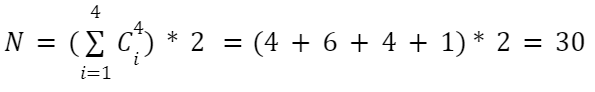
1. **Привести классификационные признаки и дать сравнительную характеристику разновидностям подстановочных шифров**

Подстановочные шифры делятся на моноалфавитные, полиграммные, омофонические и полиалфавитные. В моноалфавитных символы сообщения заменяются на другие символы того же алфавита. Для полиграммных шифров характерна замена группы символов и на символ того же или другого алфавита. В омофонических шифрах производят замену сходных по звучанию частей сообщения. В полиалфавитных для шифрования используется 2 и более алфавита.

Наиболее сложными для реализации, но также наиболее криптостойкими, являются омофонические шифры, так как сложно создать алгоритм, определяющий созвучие слов или групп символов. Наименее криптостойкими являются моноалфавитные шифры, так как легко проанализировать частоту встречаемости символов, за счёт чего сообщение и можно расшифровать.

1. **Сколько разновидностей шифров, подобных шифру Цезаря, можно составить для алфавитов русского и белорусского языков?**

Существует 4 разновидности шифра Цезаря, но так как мы можем комбинировать несколько шифров, необходимо найти сумму сочетаний, умноженную на 2, так как 2 алфавита:



1. **Найти ключ шифра, с помощью которого получен шифротекст: «byajhvfwbjyyfzgjcktljdfntkmyjcnm»**

Предполагая, что сообщение зашифровано шифром Цезаря, используем следующий код на Python, чтобы найти ключ брутфорсом:

message = 'byajhvfwbjyyfzgjcktljdfntkmyjcnm'

LETTERS = 'abcdefghijklmnopqrstuvwxyz'

for key in range(len(LETTERS)):

    translated = ''

    for symbol in message:

        if symbol in LETTERS:

            num = LETTERS.find(symbol)

            num = num - key

        if num < 0:

            num = num + len(LETTERS)

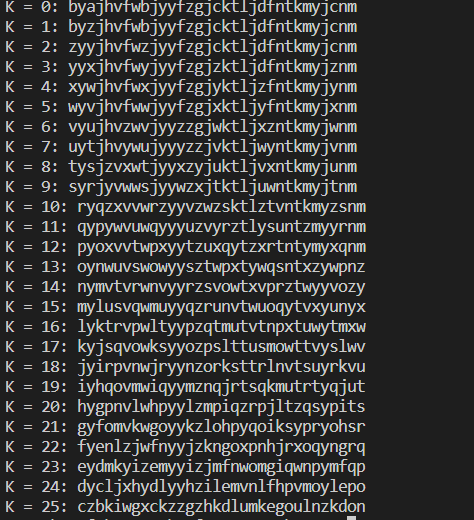
            translated = translated + LETTERS[num]

        else:

            translated = translated + symbol

    print('K = %s: %s' % (key, translated))

Получим следующий результат:



1. **Расшифровать (с демонстрацией каждого шага алгоритма) текст Сi = «qrscqcocqclc», зашифрованный аффинным шифром Цезаря при N = 26, а = 3, b = 5**

Расшифрование основано на использовании соотношения:

*x* ≡ *a*–1 (*y* + *N* – *b*) *mod N*

Найдем число, обратное 3 по модулю 26*:*

26 = 3\*8 + 2

3 = 2\*1 + 1

1 = 3 – 2\*1 = 3 – 26 + 3\*8 = 3\*9 – 26

*a*-1 = 9

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Mi* | *q* | *r* | *s* | *c* | *q* | *c* | *o* | *c* | *q* | *c* | *l* | *c* |
| *y* | 16 | 17 | 18 | 2 | 16 | 2 | 14 | 2 | 16 | 2 | 11 | 2 |
| *x =* 9(*y +* 26 – 5) | 333 | 342 | 351 | 207 | 333 | 207 | 315 | 207 | 333 | 207 | 288 | 207 |
| *x =* 9(*y +* 26 – 5) *(mod 26)* | 21 | 4 | 13 | 25 | 21 | 25 | 3 | 25 | 21 | 25 | 2 | 25 |
| *Ci* | *v* | *e* | *n* | *z* | *v* | *z* | *d* | *z* | *v* | *z* | *c* | *z* |

1. **Зашифровать и расшифровать свою фамилию (на основе кириллицы), используя аффинный шифр Цезаря**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Mi* | п | р | и | г | о | д | и | ч |
| *x* | 16 | 17 | 9 | 3 | 15 | 4 | 9 | 24 |
| *y =* 5*x*  + 3 | 83 | 88 | 48 | 18 | 78 | 23 | 48 | 123 |
| *y =* (5*x +*3) *(mod 33)* | 17 | 22 | 15 | 18 | 12 | 23 | 15 | 24 |
| *Ci* | р | х | о | с | л | ц | о | ч |

*x* ≡ *a*–1 (*y* + *N* – *b*) *mod N*

33 = 5\*6 + 3

5 = 3\*1 + 2

3 = 2\*1 + 1

1 = 3 – 2\*1 = 3 – 5 + 3\*1 = 3\*2 – 5 = 33\*2 – 5\*12 – 5 = 33\*2 – 5\*13

33 – 13 = 20

*a*-1 = 20

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Mi* | р | х | о | с | л | ц | о | ч |
| *y* | 17 | 22 | 15 | 18 | 12 | 23 | 15 | 24 |
| *x =* 20(*y +* 33 – 3) | 940 | 1040 | 900 | 960 | 840 | 1060 | 900 | 1080 |
| *x =* 20(*y +* 33 – 3) *(mod 33)* | 16 | 17 | 9 | 3 | 15 | 4 | 9 | 24 |
| *Ci* | п | р | и | г | о | д | и | ч |

1. **Можно ли использовать в качестве ключевого в шифре Виженера слово, равное по длине открытому тексту?**

Да

1. **По какому признаку можно определить, что текст зашифрован шифром Плейфера?**

Шифр предусматривает шифрование пар символов (биграмм) вместо одиночных символов, следовательно, в зашифрованном сообщении будут в среднем чаще встречаться двойные повторения символов.

1. **Имеются ли предпочтения в выборе размеров таблицы Трисемуса для виртуального алфавита мощностью 40: 4×10; 10×4; 5×8; 8×5; 2×20; 20×2?**

Предпочтений нет, но стоит учитывать, что при выборе разных размерностей таблиц получатся разные шифры.

1. **Охарактеризовать основные виды атак на шифры**

Атака с известным шифротекстом – известен алгоритм, имеет образец шифра, неизвестен ключ.

Атака с выбором шифротекста – имеется большое количество шифротекста, из которого можно выбрать только нужную часть.

Адаптивная атака с выбором шифротекста - имеется возможность выбирать новые шифрограммы для расшифрования с учетом того, что известна некоторая информация из предыдущих сообщений.

Атака с известным открытым шифротекстом - то же, что и предыдущая, но для некоторых шифрограмм имеются соответствующие им открытые тексты.

Атака с выбором открытого текста – имеется несколько открытых текстов и соответствующих им шифротекстов.

Адаптивная атака с выбором открытого текста – имеется возможность выбирать новые шифрограммы и соответствующий им открытый текст, при этом известна некоторая информация из предыдущих сообщений.

Атака на основе связанных ключей – неизвестны ключи, но известно соотношение или связь между ними.

Атака с выбором ключа – часть ключа задаётся при взломе.

1. **Сравнить криптостойкость шифра Цезаря и шифра Виженера**

Шифр Виженера является более криптостойким, чем шифрм Цезаря: в качестве ключа в нем используется слово и его сложно взломать вручную с помощью одного только частотного анализа или перебора. Каждая буква ключа генерирует число, и в результате мы получаем несколько ключей для сдвига букв.

1. **Охарактеризовать основные методы взлома подстановочных шифров**

Брутфорс – прямой перебор всех возможных вариантов.

Частотный анализ – установка однозначного соответствия между символами шифра и алфавита на основе схожего числа вероятности встречи символа в тексте.

Анализ встречаемости триграмм – лучше всего подходит для зашифрованного английского текста. Так как триграммы THE, AND, ING и другие встречаются в английском языке достаточно часто, необходимо установить соответствие между буквами триграмм и буквами шифра (как правило на основе частотного анализа). Соответствие между оставшимися символами устанавливается на основе уже установленных.

Поиск восхождением к вершине – берётся часть шифротекста (называемая ключом), и оставшаяся часть расшифровывается с его помощью. Затем вычисляется коэффициент вероятности принадлежности расшифрованного текста к естественному языку. В ключе производятся какие-либо изменения (например, перестановка первых двух букв местами) и снова расшифровывается текст и вычисляется коэффициент. Так повторяется, пока коэффициент не перестанет изменяться.

**Вывод:** в ходе лабораторной работы были приобретены практические навыки разработки и использования приложений для реализации подстановочных шифров Цезаря и таблицы Трисемуса с ключевым словом, были построены гистрограммы частоты встречаемость символов и измерено время шифрования/дешифрования. Временные затраты незначительны.