ФГБОУ ВО ‘‘Чувашский государственный университет им. И. Н. Ульянова’’.

Факультет: Информатики и вычислительной техники.

Кафедра: Вычислительной техники.

Средства защиты информации

Лабораторная работа №1

Выполнил: студент группы ИВТ-41-18

Сергеев Давид Евгеньевич

Проверил: доцент Ковалев Сергей Васильевич

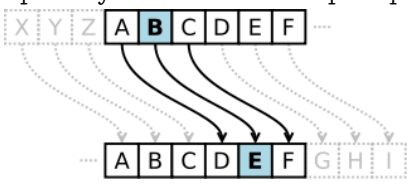
Чебоксары 2021 г.

Задание:

Разработать алгоритм и написать программу, обеспечивающую ввод произвольного открытого текста и выдачу шифрограммы, полученную изучаемым методом, а также дешифрацию - получение открытого текста из шифрограммы.

Решение:

В качестве примера в задаче был представлен один из видов шифра подстановки(шифр Цезаря), суть которого заключается в сдвиге внутри алфавита по некоторому правилу. В качестве примера, шифр Цезаря со сдвигом 3:



Изначальное состояние	A	B	C	D	E	F
Шифр Цезаря, сдвиг = 3	D	E	F	G	H	I

Тем не менее шифр Цезаря легко взламывается и не имеет почти никакого применения на практике.

Частью задания было условие, каждому m-символьному блоку открытого текста соответствует m-символьноый блок шифртекста. Это условие привело меня к иному более стойкому(абсолютно стойкому)алгоритму шифрования. Это шифр Вернама. Шифр является разновидностью одноразовых блокнотов.

> Идея шифроблокнотов:
>
> шифровальщик по дипломатической почте или при личной встрече снабжается блокнотом, каждая страница которого содержит ключи. Такой же блокнот есть и у принимающей стороны. Использованные страницы уничтожаются.

В нём используется булева функция - исключающее или(xor). Однако и этот шифр имеет свои недостатки:

- Машинной памяти используется вдвое больше.
- Ключ должен быть истинно случайным(полученным, например, на основе хаотически изменяющихся параметров протекающего физического процесса, а не алгоритма, разработанного человеком).
- Отсутствие подтверждения подлинности и целостности сообщения. Решается с помощью добавления хэша сообщения в его конец.
- Обеим сторонам нужно иметь при себе достаточное количество ключей, чтобы шифровать сообщения различной длины. На практике сложно предсказать конечную длину текста.

Предлагаю в качестве примера разобрать работу на 3ёх символьном сообщении. Для начала необходимо рассмотреть условия моего варианта.
V = {0,1,2,3,4} (алфавит)

m = 2 (длина блока шифрограммы)

|V| = 5, |V|^m = 25

Ввод пользователя: ‘‘123’’

По условию задачи необходимо разбить сообщение на блоки длиной m. Таким образом, сообщение будет выглядеть так: [‘12’, ‘3’]. Как мы можем заметить, количетво символов во втором элементе списка равно 1. В соответствии с заданием пробелы, а также недостаток символов в шифруемом блоке по договорённости сторон заменяются определённым символом. У меня этот символ ‘5’ при вводе данных и ‘_’ при выводе. Согласно данному правилу, обработанное сообщение принимает вид: [‘12’, ‘3_’]. Далее, я преобразую алфавит, данный мне в буквенный для обеспечения работы шифра Вернама, в соотвествии с правилом:

Прямой переход от числового представления к буквенному						
Исходный символ	0	1	2	3	4	5
Буквенное представление	a	b	c	d	e	f

Обратный переход от буквенного представления к числовому						
Исходный символ	a	b	c	d	e	f
Числовое представление	0	1	2	3	4	_

Также, для генерации шифра вернама необходим случайно сгенерированный ключ. Для примера, возьмём ключ: ‘mypj’.

Переход от числового представления к буквенному:

[‘12’, ‘3_’] -> [‘bc’, ‘df’].

Сообщение				
Буквенное представление	b	c	d	f
Код ascii	98	99	100	102
Битовое представление кода ascii	1100010	1100011	1100100	1100110

Ключ				
Буквенное представление	m	y	p	j
Код ascii	109	121	112	106
Битовое представление кода ascii	1101101	1111001	1110000	1101010

Алгоритм Вернама				
Ключ	m	y	p	j
Буква сообщения	b	c	d	f
Шифр Вернама	1100010 1101101	1100011 1111001	1100100 1110000	1100110 1101010
Результат работы	1101110	1100001	1110011	1101111
Шифротекст	n	a	s	o

Текст программы:

```
from random import choice
from string import ascii_lowercase

ALPHABET = ['0', '1', '2', '3', '4']
ALPHABET_STRAIGHT = {'0': 'a', '1': 'b', '2': 'c', '3': 'd', '4': 'e'}
ALPHABET_REVERSE = {'a': '0', 'b': '1', 'c': '2', 'd': '3', 'e': '4'}
CIPHER_MSG_BLOCK_SIZE = 2
CIPHER_KEY = ''


def vernam_cipher(message, key, encrypt):
    result = ""
    for i in range(len(message)):
        char = message[i]
        if encrypt:
            result += chr((ord(char) - 97 + ord(key[i]) - 97) % 26 + 97)
        else:
            result += chr((ord(char) - ord(key[i]) + 26) % 26 + 97)
    return result


def vernam_key_generator(msg_len: int):
    letters_choice = ascii_lowercase
    result_str = ''.join(choice(letters_choice) for _ in range(msg_len))
    return result_str


def cheker(msg: str):
    if not set(msg).issubset(set(ALPHABET)):
        raise ValueError('Symbol not in alphabet')


def number_to_new_alphabet(msg: str) -> str:
    result = ''
    for char in msg:
        result += ALPHABET_STRAIGHT[char]
    return result


def new_alphabet_to_number(msg: str) -> str:
    result = ''
    for char in msg:
        result += ALPHABET_REVERSE[char]
    return result
```

```
def input_data_handling():
    message = input("Ввведите сообщение: ")

    space_replacement = '5'
    global ALPHABET, ALPHABET_REVERSE, ALPHABET_STRAIGHT
    ALPHABET.append(space_replacement)  # space -> 5
    ALPHABET_STRAIGHT.update({'5': 'f'})  # 5 -> 'f'
    ALPHABET_REVERSE.update({'f': '_'})
    message.replace(' ', space_replacement)

    global CIPHER_KEY
    CIPHER_KEY = vernam_key_generator(len(message))
    if len(message) % CIPHER_MSG_BLOCK_SIZE != 0:
        CIPHER_KEY = vernam_key_generator(len(message) + CIPHER_MSG_BLOCK_SIZE -
len(message) % CIPHER_MSG_BLOCK_SIZE)

    msg_blocks = []
    word = ''
    msg_len = 0
    for idx in range(len(message)):
        if message[idx] != ' ':
            word += message[idx]
        else:
            word += space_replacement
        msg_len += 1

        if idx == len(message) - 1 and len(word) != CIPHER_MSG_BLOCK_SIZE or len(word) ==
CIPHER_MSG_BLOCK_SIZE:
            len_delta = CIPHER_MSG_BLOCK_SIZE - len(word)
            for x in range(len_delta):
                word += space_replacement
            cheker(word)
            msg_blocks.append(number_to_new_alphabet(word))
            word = ''
            msg_len = 0

    key_blocks = [CIPHER_KEY[idx: idx + CIPHER_MSG_BLOCK_SIZE]
                  for idx in range(0, len(CIPHER_KEY), CIPHER_MSG_BLOCK_SIZE)]

    return msg_blocks, key_blocks


def cipher(msg_blocks: list, key_blocks: list) -> str:
    msg_encrypted = []
    for idx in range(len(msg_blocks)):
        msg = msg_blocks[idx]
        key = key_blocks[idx]

        msg_processed = vernam_cipher(msg, key, True)
        msg_encrypted.append(msg_processed)

    encrypted = ''.join(str(msg) for msg in msg_encrypted)

    return encrypted
```

```python
def main():
    msg_blocks, key_blocks = input_data_handling()
    encrypted = cipher(msg_blocks, key_blocks)
    decrypted = vernam_cipher(encrypted, CIPHER_KEY, False)
    decrypted_processed = new_alphabet_to_number(decrypted)

    print(f'Ключ шифра Вернама: {CIPHER_KEY}')
    print(f'Зашифрованное сообщение: {encrypted}')
    print(f'Дешифрованное сообщение: {decrypted_processed}')
    print(f'Необработанное дешифрованное сообщение: {decrypted}')


if __name__ == '__main__':
    main()
```