Отчёт по лабораторной работе №7

Попов Олег Павлович

Содержание

List of Tables

List of Figures

1 Цель работы

Освоить на практике применение режима однократного гаммирования.

2 Выполнение лабораторной работы

Ниже представлен код консольного приложения для работы с шестнадцатеричными кодами символов.

main.py codecreate.py

```python
import codecreate
import numpy as np

mes = input("Введите сообщение: ")

code = np.array([])
code = codecreate.createCode(mes)
print("Original code:", code, "\n")

keyCode = codecreate.generateDecryptionCode(len(mes))
print("Key code:", keyCode, "\n")

xorCode = codecreate.xor(code, keyCode)
print("XOR code:", xorCode, "\n")
print('\n\t~~~~~~~~~~~~~~~~~~~~\n')

newCode = codecreate.xor(xorCode, keyCode)
print("Decrypted message:", codecreate.decryptCode(newCode))

```

Выше указан main файл программы, в котором находится исключительно реализация всех функций, написанных в файле createcode.

Порядок выполнения команд в файле main следующий: на вход программе дается ввод mes; все символы mes обрабатываются через функцию createCode(), которая создает массив шестнадца-

теричных кодов; генерируется рандомный ключ через функцию generateDecryptionCode(); сообщение кодируется через функцию однократного гаммирования xor(); все вышеперечисленное выводится на экран; и под конец сообщение xorCode дешифруется через функцию xor() [newCode] и выводится на экран.

main.py codecreate.py

```python
import numpy as np
import random


# нужен для создания кода из сообщения
def createCode(mes):
    code = np.array([])
    for i in range(len(mes)):
        code = np.append(code, hex(ord(mes[i])))
    return code


# нужен для создания рандомного ключа
def generateDecryptionCode(s):
    code = np.array([])
    for i in range(s):
        r = random.randint(0, 255)
        code = np.append(code, hex(r))
    return code


# функция "исключащее или"
def xor(code1, code2):
    code = np.array([])
    for i in range(code1.size):
        code = np.append(code, hex(int(code1[i], 16) ^ int(code2[i], 16)))
    return code


letters = "АаБбВвГгДдЕеЁёЖжЗзИиЙйКкЛлМмНнОоПпРрСсТтУуФфХхЦцЧчШшЩщЪъЫыЬьЭэЮюЯя"
let_code = createCode(letters)


# нужен для создания сообщения из кода
def decryptCode(code):
    mes = ""
    for i in range(code.size):
        if code[i] in let_code:
            found_index = np.where(let_code == code[i])
            mes = mes + letters[int(found_index[0])]
        else:
            mes = mes + bytes.fromhex(code[i][2:]).decode('ascii')
    return mes
```

Данный код представляет собой файл createcode. Здесь, как уже было сказано ранее, записаны все функции.

Функция createCode(): берет на вход сообщение типа string и для

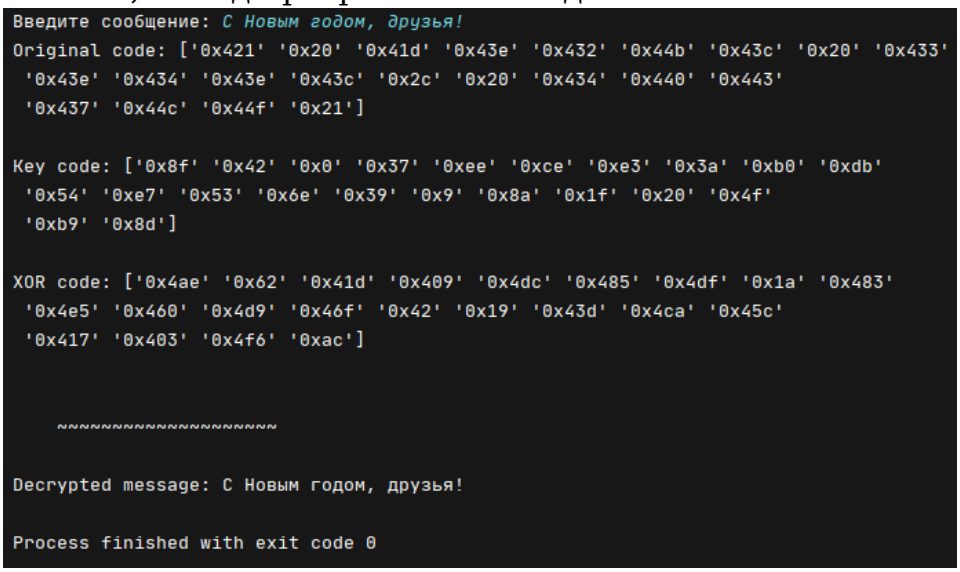
каждого символа сообщения генерирует его шестнадцатеричный код ascii.

Функция generateDecryptionCode(): создает рандомный ключ определенного размера s.

Функция xor(): реализует “исключающее или” для code1 и code2 и выводит зашифорованное сообщение в виде массива кодов.

Функция decryptCode(): нужна для расшифровки кодов code и вывода сообщения. Для реализации данной функции пришлось отдельно записывать символы кириллицы и создавать для них массив кодов, так как функция bytes.fromhex() не распознает трехзначные шестнадцатеричные коды (все коды кириллицы трехзначные). В итоге, функция проверяет, есть ли код в массиве кириллицы, и выводит символы.

В итоге, вывод программы выглядит так:



3 Выводы

В ходе данной работы я ознакомился с однократным гаммированием и реализовал его в приложении командной строки.