Отчёт по лабораторной работе №7

Дисциплина: Информационная безопасность

Евдокимов Иван Андреевич

Содержание

# Техническое оснащение:

* Персональный компьютер с операционной системой Windows 10;
* OBS Studio, использующийся для записи скринкаста лабораторной работы;
* Приложение Visual Studio Code для редактирования файлов формата *md*, а также для конвертации файлов отчётов и презентаций;

# Цель работы:

Освоить на практике применение режима однократного гаммирования.

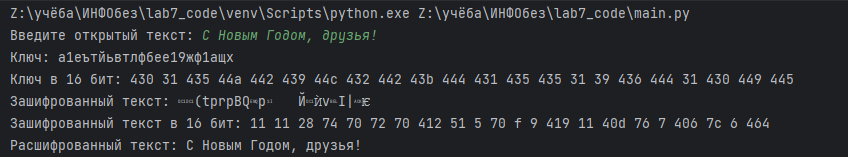
# Постановка задачи

Нужно подобрать ключ, чтобы получить сообщение «С Новым Годом, друзья!». Требуется разработать приложение, позволяющее шифровать и дешифровать данные в режиме однократного гаммирования. Приложение должно:

# Код программы

# Импортируем модули для работы со строками и для генерации случайных чисел  
import random  
import string  
  
  
# Создаем класс для кодирования и декодирования текста  
class TextEncoding:  
  
 @staticmethod  
 # Метод для определения алфавита, который следует использовать для генерации ключа  
 def determine\_alphabet(text):  
 # Если первый символ текста в английском алфавите (в нижнем регистре),  
 # то возвращаем английский алфавит и цифры  
 if text[0] in string.ascii\_lowercase:  
 return string.ascii\_lowercase + string.digits  
 else:  
 # В противном случае возвращаем русский алфавит и цифры  
 return "абвгдеёжзийклмнопрстуфхцчшщъыьэюя" + string.digits  
  
 @staticmethod  
 # Метод для генерации ключа  
 # Ключ состоит из случайных символов алфавита (определенного в методе determine\_alphabet)  
 def generate\_key(size, alphabet):  
 return "".join(random.choice(alphabet) for \_ in range(size))  
  
 @staticmethod  
 # Метод для преобразования строки в шестнадцатеричный формат  
 # Каждый символ кодируется в шестнадцатеричную систему и объединяется в строку с помощью пробела  
 def to\_hex(coding):  
 return " ".join(hex(ord(character))[2:] for character in coding)  
  
 @staticmethod  
 # Метод для кодирования строки  
 # Происходит применение операции XOR между кодами символов текста и ключа  
 def encode\_string(text, key):  
 return "".join(chr(ord(char) ^ ord(key\_char)) for char, key\_char in zip(text, key))  
  
 @staticmethod  
 # Метод для поиска возможных ключей  
 # Принимает на вход строку текста и фрагмент этого текста  
 # Создает список возможных ключей, которые могут декодировать зашифрованный текст обратно в известный фрагмент  
 def find\_possible\_keys(text, fragment):  
 key\_length = len(fragment)  
 possible\_keys = []  
  
 # Проходим по всему тексту с шагом, равным длине фрагмента  
 for index in range(len(text) - key\_length + 1):  
 # Получаем возможный ключ путем применения операции XOR между очередной частью текста и фрагментом  
 key = [chr(ord(char) ^ ord(key\_char)) for char, key\_char in zip(text[index:index + key\_length], fragment)]  
 # Предполагаемый расшифрованный текст получаем путем кодирования зашифрованного текста с использованием полученного ключа  
 presumed\_plaintext = TextEncoding.encode\_string(text, key)  
  
 # Если известный фрагмент присутствует в предполагаемом расшифрованном тексте, добавляем ключ в список возможных  
 if fragment in presumed\_plaintext:  
 possible\_keys.append(''.join(key))  
  
 return possible\_keys  
  
# Получаем от пользователя открытый текст  
plaintext = input("Введите открытый текст: ")  
# Определяем алфавит для генерации ключа  
alphabet = TextEncoding.determine\_alphabet(plaintext)  
# Генерируем ключ  
key = TextEncoding.generate\_key(len(plaintext), alphabet)  
  
# Выводим сгенерированный ключ и его шестнадцатеричное представление  
print(f"Ключ: {key}", f"Ключ в 16 бит: {TextEncoding.to\_hex(key)}", sep='\n')  
  
# Кодируем открытый текст с помощью сгенерированного ключа  
ciphertext = TextEncoding.encode\_string(plaintext, key)  
# Выводим зашифрованный текст и его шестнадцатеричное представление  
print(f"Зашифрованный текст: {ciphertext}", f"Зашифрованный текст в 16 бит: {TextEncoding.to\_hex(ciphertext)}",  
 sep='\n')  
  
# Декодируем зашифрованный текст с помощью сгенерированного ключа  
decrypted\_text = TextEncoding.encode\_string(ciphertext, key)  
# Выводим расшифрованный текст  
print("Расшифрованный текст:", decrypted\_text)  
  
# Получаем от пользователя известный фрагмент открытого текста  
known\_fragment = input("Введите фрагмент открытого текста: ")  
# Ищем возможные ключи для шифротекста  
possible\_keys = TextEncoding.find\_possible\_keys(ciphertext, known\_fragment)  
# Выводим найденные ключи  
print("Возможные ключи для шифротекста:", possible\_keys)

вывод запуска программы 1 (шифровка и дишифровка текста).



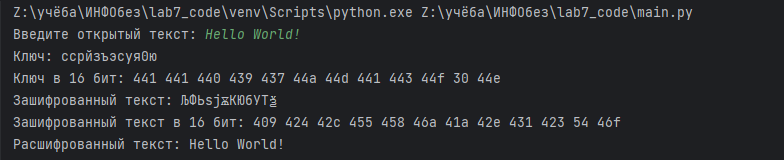
шифровка и дишифровка текста

вывод запуска программы 2 (шифровка фрагмента текста).

шифровка фрагмента текста

шифровка фрагмента текста

вывод запуска программы 3 на английском (шифровка и дишифровка текста на английском).



шифровка и дишифровка текста на английском

вывод запуска программы 4 на английском (шифровка фрагмента текста на английском).

шифровка фрагмента текста на английском

шифровка фрагмента текста на английском

на видео к выполнения работы будут представленны более удачные варианты запуска программы

**Выводы:**

Мною были освоино на практике применение режима однократного гаммирования.

# Список литературы

1. [Официальный сайт VirtualBox](https://www.virtualbox.org/)
2. [Материал для выполнения лабораторной](https://esystem.rudn.ru/pluginfile.php/2090284/mod_resource/content/2/007-lab_crypto-gamma.pdf)
3. [Официальный сайт CentOS](https://www.centos.org/)