Oтчёта по лабораторной работе №7

Элементы криптографии. Однократное гаммирование

Акондзо Жордани Лади Гаэл

Содержание

# Цель работы

* Цель данной лабораторной работы — изучение метода однократного гаммирования для шифрования и дешифрования данных.

# Теоретическое введение

* Однократное гаммирование (шифр Вернама) — это метод симметричного шифрования, при котором каждое сообщение шифруется с помощью ключа, длина которого совпадает с длиной сообщения. Шифрование и дешифрование происходит с использованием операции побитового исключающего ИЛИ (XOR) между символами ключа и открытого текста. Преимущество однократного гаммирования заключается в его абсолютной криптостойкости, если ключ используется только один раз и является абсолютно случайным.

# Выполнение лабораторной работы

## Первый вариант программы: фиксированное сообщение для шифрования

* Первый вариант программы реализует шифрование фиксированного сообщения «С Новым Годом, друзья!», ключ для которого генерируется случайно. Этот пример наглядно демонстрирует работу операции XOR для шифрования и последующего дешифрования. (рис. 1)

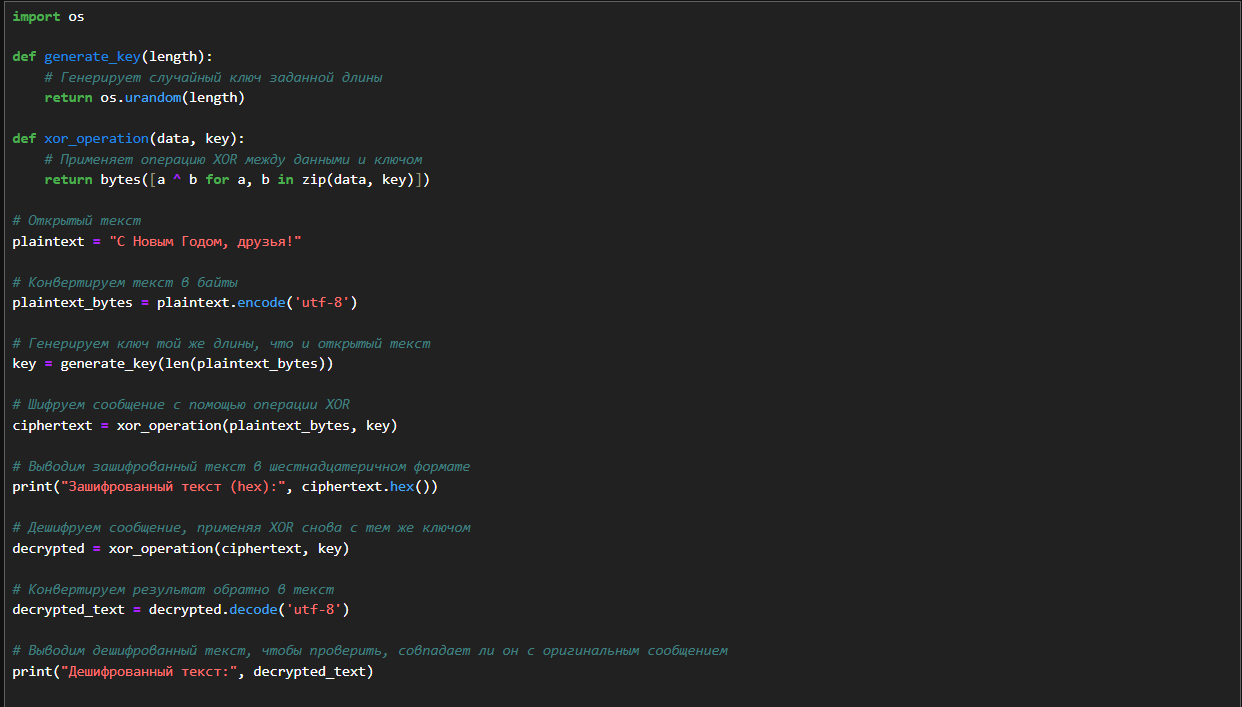


Рис. 1: фиксированное сообщение для шифрования

## КОД1:

import os  
  
def generate\_key(length):  
 # Генерирует случайный ключ заданной длины  
 return os.urandom(length)  
  
def xor\_operation(data, key):  
 # Применяет операцию XOR между данными и ключом  
 return bytes([a ^ b for a, b in zip(data, key)])  
  
# Открытый текст  
plaintext = "С Новым Годом, друзья!"  
  
# Конвертируем текст в байты  
plaintext\_bytes = plaintext.encode('utf-8')  
  
# Генерируем ключ той же длины, что и открытый текст  
key = generate\_key(len(plaintext\_bytes))  
  
# Шифруем сообщение с помощью операции XOR  
ciphertext = xor\_operation(plaintext\_bytes, key)  
  
# Выводим зашифрованный текст в шестнадцатеричном формате  
print("Зашифрованный текст (hex):", ciphertext.hex())  
  
# Дешифруем сообщение, применяя XOR снова с тем же ключом  
decrypted = xor\_operation(ciphertext, key)  
  
# Конвертируем результат обратно в текст  
decrypted\_text = decrypted.decode('utf-8')  
  
# Выводим дешифрованный текст, чтобы проверить, совпадает ли он с оригинальным сообщением  
print("Дешифрованный текст:", decrypted\_text)

## Описание работы программы 1:

1. Генерация ключа: Программа генерирует случайный ключ с помощью функции os.urandom(), который соответствует длине открытого текста.
2. Шифрование: С помощью операции XOR каждый байт открытого текста комбинируется с соответствующим байтом ключа.
3. Дешифрование: Повторное применение операции XOR с тем же ключом позволяет восстановить исходный текст.

## Результат\_1:

* Зашифрованный текст выводится в шестнадцатеричном формате.
* Дешифрованный текст полностью совпадает с исходным. (рис. 2)

Рис. 2: Результат

Рис. 2: Результат

## Второй вариант программы: ввод текста пользователем

* Во втором варианте программы пользователю предоставляется возможность ввести любое сообщение для шифрования. Программа генерирует ключ и выводит зашифрованный и дешифрованный текст. (рис. 3)

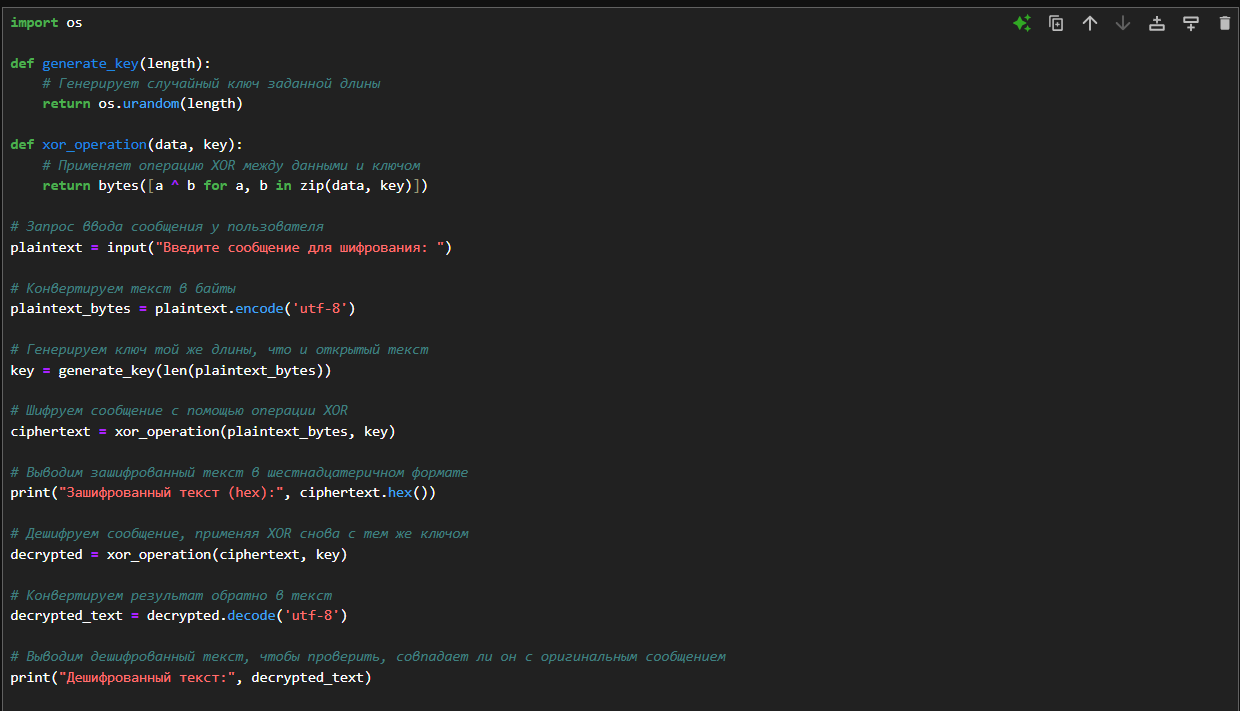


Рис. 3: ввод текста пользователем

## КОД2:

import os  
  
def generate\_key(length):  
 # Генерирует случайный ключ заданной длины  
 return os.urandom(length)  
  
def xor\_operation(data, key):  
 # Применяет операцию XOR между данными и ключом  
 return bytes([a ^ b for a, b in zip(data, key)])  
  
# Запрос ввода сообщения у пользователя  
plaintext = input("Введите сообщение для шифрования: ")  
  
# Конвертируем текст в байты  
plaintext\_bytes = plaintext.encode('utf-8')  
  
# Генерируем ключ той же длины, что и открытый текст  
key = generate\_key(len(plaintext\_bytes))  
  
# Шифруем сообщение с помощью операции XOR  
ciphertext = xor\_operation(plaintext\_bytes, key)  
  
# Выводим зашифрованный текст в шестнадцатеричном формате  
print("Зашифрованный текст (hex):", ciphertext.hex())  
  
# Дешифруем сообщение, применяя XOR снова с тем же ключом  
decrypted = xor\_operation(ciphertext, key)  
  
# Конвертируем результат обратно в текст  
decrypted\_text = decrypted.decode('utf-8')  
  
# Выводим дешифрованный текст, чтобы проверить, совпадает ли он с оригинальным сообщением  
print("Дешифрованный текст:", decrypted\_text)

## Описание работы программы 2:

1. Ввод сообщения: Пользователь вводит любое сообщение, которое будет зашифровано.
2. Генерация ключа: Ключ генерируется случайным образом и имеет ту же длину, что и введённое сообщение.
3. Шифрование и дешифрование: Программа выполняет шифрование и дешифрование с помощью операции XOR.

## Результат\_2:

* Программа позволяет шифровать и дешифровать любое сообщение, введённое пользователем.
* Зашифрованный текст выводится в виде шестнадцатеричных символов, что упрощает его анализ.
* Дешифрованный текст восстанавливается полностью и совпадает с введённым пользователем. (рис. 4) и (рис. 5)

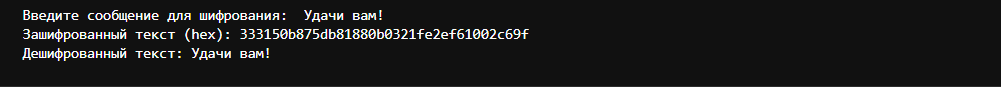


Рис. 4: Результат

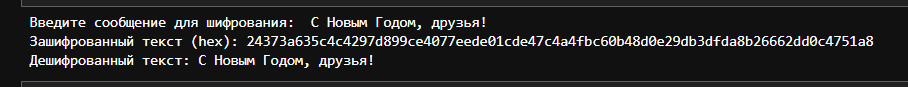


Рис. 5: Результат

# Выводы

В ходе выполнения лабораторной работы были созданы два варианта программы для однократного гаммирования. В первом случае программа демонстрировала шифрование фиксированного текста, а во втором случае пользователь мог вводить любое сообщение для шифрования. Оба варианта программы успешно продемонстрировали работу метода однократного гаммирования, а также его основное преимущество — невозможность восстановления исходного текста без знания ключа.