

Лабораторная работа №8

Элементы криптографии. Шифрование (кодирование) различных исходных текстов одним ключом

Щербак Маргарита Романовна

НПИбд-02-21

Студ. билет: 1032216537

2024

RUDN

Освоить на практике применение режима однократного гаммирования на примере кодирования различных исходных текстов одним ключом.

Гаммирование представляет собой наложение (снятие) на открытые (зашифрованные) данные последовательности элементов других данных, полученной с помощью некоторого криптографического алгоритма, для получения зашифрованных (открытых) данных. Иными словами, наложение гаммы — это сложение её элементов с элементами открытого (закрытого) текста по некоторому фиксированному модулю, значение которого представляет собой известную часть алгоритма шифрования.

Два текста кодируются одним ключом (однократное гаммирование). Требуется не зная ключа и не стремясь его определить, прочитать оба текста. Необходимо разработать приложение, позволяющее шифровать и дешифровать тексты P1 и P2 в режиме однократного гаммирования. Приложение должно определить вид шифротекстов C1 и C2 обоих текстов P1 и P2 при известном ключе ; необходимо определить и выразить аналитически способ, при котором злоумышленник может прочитать оба текста, не зная ключа и не стремясь его определить.

1. Генерация ключа: использовала функцию `generate_key`, которая создает случайный ключ заданной длины (максимальной длины между двумя текстами P1 и P2).
2. Шифрование текстов: функция `encrypt` применяет операцию XOR между каждым байтом открытого текста и соответствующим байтом ключа. Оба текста (P1 и P2) шифруются с использованием одного и того же ключа, и результаты сохраняются в переменных C1 и C2.
3. Дешифрование текстов: функция `decrypt` применяет ту же операцию XOR между шифротекстами и ключом, что позволяет восстановить оригинальные тексты P1 и P2.

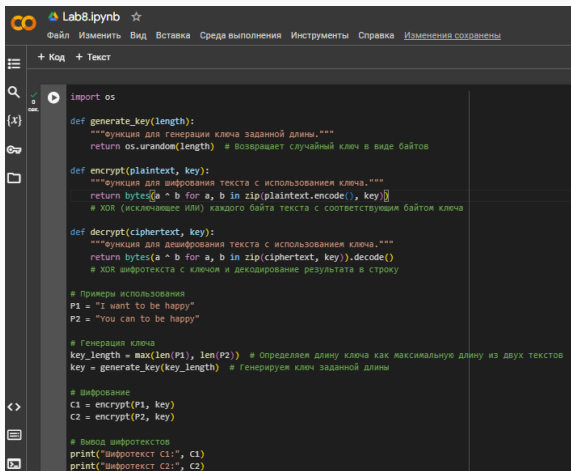
Чтобы злоумышленник мог прочитать оба текста, не зная ключа, можно использовать следующие методы:

- Анализ шифротекстов: Если злоумышленник имеет доступ к шифротекстам C1 и C2, он может заметить, что они были зашифрованы с помощью одного и того же ключа.
- Сравнительный анализ: При XOR-операции, если одно и то же значение (например, буква 'H') XOR'ится с одним и тем же байтом ключа, то это создает предсказуемый паттерн.
- Статистический анализ: Если злоумышленник знает, что тексты имеют характерный шаблон (например, фразы на английском языке), он может использовать статистические методы, чтобы проанализировать и, возможно, расшифровать тексты.

Итак, код эффективно реализует шифрование и дешифрование текстов в режиме однократного гаммирования. Однако он также иллюстрирует потенциальные уязвимости, которые могут быть использованы злоумышленниками для расшифровки текстов без знания ключа.

Выполнение лабораторной работы

Код целиком показан на рис.1:



```
import os

def generate_key(length):
    """Функция для генерации ключа заданной длины."""
    return os.urandom(length) # Возвращает случайный ключ в виде байтов

def encrypt(plaintext, key):
    """Функция для шифрования текста с использованием ключа."""
    return bytes(a ^ b for a, b in zip(plaintext.encode(), key))
    # XOR (исключающее ИЛИ) каждого байта текста с соответствующим байтом ключа

def decrypt(ciphertext, key):
    """Функция для дешифрования текста с использованием ключа."""
    return bytes(a ^ b for a, b in zip(ciphertext, key)).decode()
    # XOR шифротекста с ключом и декодирование результата в строку

# Примеры использования
P1 = "I want to be happy"
P2 = "You can to be happy"

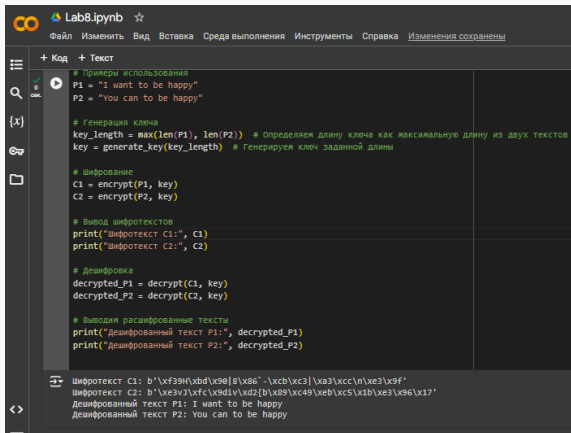
# Генерация ключа
key_length = max(len(P1), len(P2)) # Определяем длину ключа как максимальную длину из двух текстов
key = generate_key(key_length) # Генерируем ключ заданной длины

# Шифрование
C1 = encrypt(P1, key)
C2 = encrypt(P2, key)

# Вывод шифротекстов
print("Шифротекст C1:", C1)
print("Шифротекст C2:", C2)
```

Рис. 1: Код

Результат кода (рис.2):



```
Lab8.ipynb
Файл Изменить Вид Вставка Среда выполнения Инструменты Справка Изменения сохранены

+ Код + Текст

# Примеры использования
P1 = "I want to be happy"
P2 = "You can to be happy"

# Генерация ключа
key_length = max(len(P1), len(P2)) # Определяем длину ключа как максимальную длину из двух текстов
key = generate_key(key_length) # Генерируем ключ заданной длины

# Шифрование
C1 = encrypt(P1, key)
C2 = encrypt(P2, key)

# Вывод шифротекстов
print("Шифротекст C1:", C1)
print("Шифротекст C2:", C2)

# Дешифровка
decrypted_P1 = decrypt(C1, key)
decrypted_P2 = decrypt(C2, key)

# Выводим расшифрованные тексты
print("Дешифрованный текст P1:", decrypted_P1)
print("Дешифрованный текст P2:", decrypted_P2)

Шифротекст C1: b'\xf39N\xbd\x90|8\x86' - \xcb\xc3|\xa3\xcc\n\xe3\x9f'
Шифротекст C2: b'\xe3vJ\xfc\x9div\xd2{b\x89\xc49\xeb\xc5\x1b\xe3\x96\x17'
Дешифрованный текст P1: I want to be happy
Дешифрованный текст P2: You can to be happy
```

Рис. 2: Продолжение кода и результат

Таким образом, в ходе ЛР№8 я освоила на практике применение режима однократного гаммирования на примере кодирования различных исходных текстов одним ключом.

- Методические материалы курса.