

Отчёта по лабораторной работе №7

Элементы криптографии. Однократное гаммирование

Акондзо Жордани Лади Гаэл

Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Выполнение лабораторной работы	7
3.1	Первый вариант программы: фиксированное сообщение для шиф-	
	рования	7
3.2	КОД1:	7
3.3	Описание работы программы 1:	9
3.4	Результат_1:	9
3.5	Второй вариант программы: ввод текста пользователем	9
3.6	КОД2:	10
3.7	Описание работы программы 2:	11
3.8	Результат_2:	12
4	Выводы	13

Список иллюстраций

3.1	фиксированное сообщение для шифрования	7
3.2	Результат	9
3.3	ввод текста пользователем	10
3.4	Результат	12
3.5	Результат	12

List of Tables

1 Цель работы

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

2 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.

3 Выполнение лабораторной работы

3.1 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. 3.1)

```
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)
```

Рис. 3.1: фиксированное сообщение для шифрования

3.2 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')

```



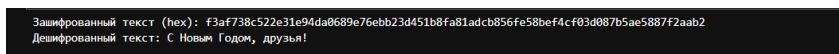
```
# Выводим дешифрованный текст, чтобы проверить, совпадает ли он с оригинальным со
print("Дешифрованный текст:", decrypted_text)
```

3.3 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.

3.4 Результат_1:

- Зашифрованный текст выводится в шестнадцатеричном формате.
- Дешифрованный текст полностью совпадает с исходным. (рис. 3.2)



```
Зашифрованный текст (hex): f3af738c522e31e94da0689e76ebb23d451b8fa81adcb856fe58bef4cf03d087b5ae5887f2aab2
Дешифрованный текст: С Новым Годом, друзья!
```

Рис. 3.2: Результат

3.5 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. 3.3)

```

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 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)

```

Рис. 3.3: ввод текста пользователем

3.6 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')

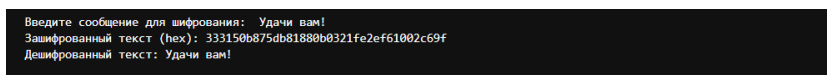
# Выводим дешифрованный текст, чтобы проверить, совпадает ли он с оригинальным со
print("Дешифрованный текст:", decrypted_text)
```

3.7 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.

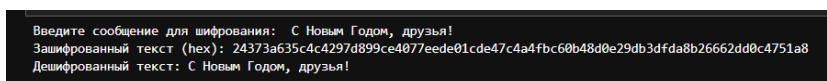
3.8 Результат_2:

- 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. 3.4) и (рис. 3.5)



```
Введите сообщение для шифрования: Удачи вам!  
Зашифрованный текст (hex): 333150b875db81880b0321fe2ef61002c69f  
Дешифрованный текст: Удачи вам!
```

Рис. 3.4: Результат



```
Введите сообщение для шифрования: С Новым Годом, друзья!  
Зашифрованный текст (hex): 24373a635c4c4297d899ce4077eede01cde47c4a4fbc60b48d0e29db3dfda8b26662dd0c4751a8  
Дешифрованный текст: С Новым Годом, друзья!
```

Рис. 3.5: Результат

4 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.