Презентация по лабораторной работе №7

Информационная безопасность

Акондзо Жордани Лади Гаэл.

17 Октября 2024

Российский университет дружбы народов, Москва, Россия

Информация

Докладчик

- Акондзо Жордани Лади Гаэл.
- студент 4-го курса группы НКНбд-01-21
- 1032215649
- Российский университет дружбы народов
- GitHub

Вводная часть

Актуальность

- Обеспечение безопасности
- Предотвращение пересечений между пользовательскими аккаунтами
- Совместный доступ к файлам

Цель работы

Цель работы

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

Теоретическое введение

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.

Материалы и методы

- Веб-сервис `GitHub` для работы с репозиториями
- Программа для виртуализации ОС `VirtualBox`
- Процессор `pandoc` для входного формата Markdown
- Результирующие форматы
  - `pdf`
  - `docx`
- Автоматизация процесса создания: `Makefile`

Выполнение лабораторной работы

Первый вариант программы: фиксированное сообщение для шифрования

```python
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)
```

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
```

Описание работы программы:

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

Второй вариант программы: ввод текста пользователем

```python
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)
```

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
```

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:

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

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

Выводы

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.