МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КУБГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**Отчет**

**по практическому заданию №13**

**по курсу**

**«КРИПТОГРАФИЧЕСКИЕ ПРОТОКОЛЫ»**

Работу выполнил

Студенты 46 группы

Нагалевский А.М.

Преподаватель:

Крамаренко А.А.

Краснодар 2024

**Постановка задачи.**

Реализовать шифрование и расшифрование AES в режиме OFB, для реализации AES можно пользоваться методами библиотек.

Режим обратной связи по выходу (OFB, Output Feedback) — это один из режимов работы блочных шифров, включая AES (Advanced Encryption Standard). Этот режим обеспечивает конфиденциальность данных, преобразуя блоки данных с использованием ключа шифрования и предыдущего выходного блока шифртекста.

Рассмотрим подробнее, как работает шифрование и расшифрование AES в режиме OFB:

**Шифрование в режиме OFB:**

1. **Инициализация**: Начинается с инициализации вектора начальной обратной связи (IV), который должен быть случайным и известным отправителю и получателю.
2. **Генерация ключевого потока**: IV передается в AES вместе с ключом шифрования для шифрования. Результат шифрования (блок шифртекста) становится первым блоком ключевого потока. Затем этот блок шифртекста используется для шифрования следующего блока (но не самого себя), что создает дополнительный блок ключевого потока.
3. **Шифрование**: Полученный ключевой поток складывается с открытым текстом для получения шифртекста.
4. **Обновление IV**: После каждого блока шифрования IV обновляется на основе предыдущего блока шифртекста.
5. **Повторение шагов 2-4**: Эти шаги повторяются для каждого блока открытого текста до завершения шифрования всего сообщения.

**Расшифрование в режиме OFB:**

1. **Инициализация**: Аналогично шифрованию, начинается с инициализации IV.
2. **Генерация ключевого потока:** IV передается в AES вместе с ключом шифрования для шифрования. Результат шифрования (блок шифртекста) становится первым блоком ключевого потока.
3. **Шифрование**: Полученный ключевой поток складывается с шифртекстом для получения открытого текста.
4. **Обновление IV**: После каждого блока шифрования IV обновляется на основе предыдущего блока шифртекста.
5. **Повторение шагов 2-4**: Эти шаги повторяются для каждого блока шифртекста до завершения расшифрования всего сообщения.

**Преимущества и особенности:**

1. **Самосинхронизирующийся режим**: Ошибка в передаче не влияет на последующие блоки, поскольку ключевой поток не зависит от результатов предыдущих шифрований.
2. **Отсутствие распространения ошибок**: Изменения в открытом тексте не распространяются на последующие блоки шифртекста.
3. **Необратимость**: OFB не обеспечивает аутентификацию, и поэтому нельзя проверить подлинность данных, используя только этот режим. Для этого требуется дополнительная проверка целостности данных, например, HMAC.

Режим OFB хорошо подходит для передачи данных в потоках, где ошибки и потери пакетов могут возникать без влияния на остальные данные.

**Текст программы:**

**Файл task1.py:**

from cryptography.hazmat.backends import default\_backend

from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes

from cryptography.hazmat.primitives import padding

from os import urandom

def encrypt\_aes\_ofb(key, iv, plaintext):

    backend = default\_backend()

    cipher = Cipher(algorithms.AES(key), modes.OFB(iv), backend=backend)

    encryptor = cipher.encryptor()

    padder = padding.PKCS7(128).padder()

    padded\_plaintext = padder.update(plaintext) + padder.finalize()

    ciphertext = encryptor.update(padded\_plaintext) + encryptor.finalize()

    return ciphertext

def decrypt\_aes\_ofb(key, iv, ciphertext):

    backend = default\_backend()

    cipher = Cipher(algorithms.AES(key), modes.OFB(iv), backend=backend)

    decryptor = cipher.decryptor()

    decrypted\_padded\_plaintext = decryptor.update(ciphertext) + decryptor.finalize()

    unpadder = padding.PKCS7(128).unpadder()

    plaintext = unpadder.update(decrypted\_padded\_plaintext) + unpadder.finalize()

    return plaintext

# Пример использования

# Генерация ключа и вектора инициализации

key = urandom(32)  # 256-bit ключ

iv = urandom(16)   # 128-bit вектор инициализации

# Текст, который нужно зашифровать

plaintext = b"Hello, world!"

# Шифрование

ciphertext = encrypt\_aes\_ofb(key, iv, plaintext)

print("Зашифрованный текст:", ciphertext)

# Расшифрование

decrypted\_plaintext = decrypt\_aes\_ofb(key, iv, ciphertext)

print("Расшифрованный текст:", decrypted\_plaintext.decode())