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

Студент 49 группы

Лобода Д.А.

Преподаватель:

Крамаренко А.А.

Краснодар 2024

**Постановка задачи.**

Реализовать шифрование и расшифрование AES в режиме CFB, для

реализации AES можно пользоваться методами библиотек.

В режиме обратной связи по шифртексту (CFB, Cipher Feedback), блок данных шифруется и затем используется для шифрования следующего блока данных. Это обеспечивает конфиденциальность данных и предотвращает распространение ошибок. Рассмотрим более подробно, как работает шифрование и расшифрование AES в режиме CFB:

**Шифрование в режиме CFB:**

1. Инициализация: Начальное значение (IV) генерируется случайным образом и используется в качестве первого блока шифртекста.
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 CFB:**

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. **Одиночный блок обработки:**

CFB работает с одним блоком данных за раз, что делает его применимым к потоковым данным, таким как передача данных по сети.

Это позволяет шифровать и расшифровывать данные независимо от их размера, без необходимости дополнительной обработки.

2. **Отсутствие распространения ошибок:**

В отличие от режимов с обратной связью по выходу (OFB) и обратной связью по шифротексту (CBC), ошибка в одном блоке не распространяется на следующие блоки.Если данные повреждаются или теряются в передаче, это не повлияет на расшифровку остальных блоков.

3. **Самосинхронизация:**

В CFB используется предыдущий блок шифртекста для генерации следующего ключевого потока, что позволяет легко синхронизироваться между отправителем и получателем данных.Не требуется дополнительного обмена информацией о состоянии между отправителем и получателем.

4. **Гибкость в размере блока:**

Режим CFB позволяет выбирать размер блока, который может быть меньше, чем размер блока шифрования, обеспечивая гибкость в работе с данными различных размеров.

**Текст программы:**

**Файл Lab\_13.java:**

import org.bouncycastle.crypto.BufferedBlockCipher;  
import org.bouncycastle.crypto.CipherParameters;  
import org.bouncycastle.crypto.engines.AESEngine;  
import org.bouncycastle.crypto.modes.CFBBlockCipher;  
import org.bouncycastle.crypto.params.KeyParameter;  
import org.bouncycastle.crypto.params.ParametersWithIV;  
import org.bouncycastle.util.encoders.Hex;  
  
import java.nio.charset.StandardCharsets;  
import java.security.SecureRandom;  
  
public class Lab\_13 {  
  
 public static byte[] encrypt(byte[] key, byte[] iv, byte[] plaintext) throws Exception {  
 BufferedBlockCipher cipher = new BufferedBlockCipher(new CFBBlockCipher(new AESEngine(), 128));  
 CipherParameters params = new ParametersWithIV(new KeyParameter(key), iv);  
 cipher.init(true, params);  
 byte[] output = new byte[cipher.getOutputSize(plaintext.length)];  
 int bytesProcessed = cipher.processBytes(plaintext, 0, plaintext.length, output, 0);  
 cipher.doFinal(output, bytesProcessed);  
 return output;  
 }  
  
 public static byte[] decrypt(byte[] key, byte[] iv, byte[] ciphertext) throws Exception {  
 BufferedBlockCipher cipher = new BufferedBlockCipher(new CFBBlockCipher(new AESEngine(), 128));  
 CipherParameters params = new ParametersWithIV(new KeyParameter(key), iv);  
 cipher.init(false, params);  
 byte[] output = new byte[cipher.getOutputSize(ciphertext.length)];  
 int bytesProcessed = cipher.processBytes(ciphertext, 0, ciphertext.length, output, 0);  
 cipher.doFinal(output, bytesProcessed);  
 return output;  
 }  
  
 public static void main(String[] args) throws Exception {  
 // Генерация случайного ключа и IV  
 SecureRandom random = new SecureRandom();  
 byte[] key = new byte[16]; // 128-bit ключ  
 random.nextBytes(key);  
 byte[] iv = new byte[16]; // 128-bit IV  
 random.nextBytes(iv);  
  
 // Текст для шифрования  
 String plaintextString = "Верблюд";  
 byte[] plaintext = plaintextString.getBytes(StandardCharsets.*UTF\_8*);  
  
 // Шифрование  
 byte[] ciphertext = *encrypt*(key, iv, plaintext);  
 System.*out*.println("Зашифрованный текст: " + Hex.*toHexString*(ciphertext));  
  
 // Расшифрование  
 byte[] decryptedText = *decrypt*(key, iv, ciphertext);  
 System.*out*.println("Расшифрованный текст: " + new String(decryptedText, StandardCharsets.*UTF\_8*));  
 }  
}