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 №11**

**по курсу**

**«КРИПТОГРАФИЧЕСКИЕ ПРОТОКОЛЫ»**

Работу выполнил

Студент 49 группы

Лобода Д.А.

Преподаватель:

Крамаренко А.А.

Краснодар 2024

**Постановка задачи.**

Реализовать программный продукт построения Хэш-функции от

введенного сообщения на блочных шифрах для произвольного блочного

шифра согласно схемы Миагуччи-Пренеля. Для построения Хэш алгоритма запрещено пользоваться готовыми библиотеками языков. Для применения

блочного шифра можно пользоваться готовыми библиотеками.

XOR шифрование (Exclusive OR) - это простой криптографический алгоритм, который работает путем применения операции XOR к каждому биту входных данных с соответствующим битом ключа. Рассмотрим основные характеристики и особенности алгоритма:

Рассмотрим основные характеристики и особенности алгоритма:

1. **Длина зашифрованного сообщения**: Для XOR шифрования длина зашифрованного сообщения равна длине исходного сообщения.
2. **Размер блока**: XOR шифрование не работает с блоками как другие алгоритмы, такие как SHA-1. Он оперирует на уровне отдельных битов.
3. **Простота алгоритма**: XOR шифрование прост в реализации и требует только ключа для шифрования и дешифрования.
4. **Одноразовый ключ (One-Time Pad)**: Если ключ используется только один раз и генерируется случайным образом и сохраняется в секрете, XOR шифрование обеспечивает абсолютную криптографическую стойкость.
5. **Уязвимость при повторном использовании ключа**: Если ключ используется несколько раз, XOR шифрование становится уязвимым для криптоанализа.
6. **Применение в различных областях:** XOR шифрование используется в различных областях, таких как защита данных, обеспечение конфиденциальности, а также в криптографических атаках и алгоритмах.
7. **Примеры использования**: XOR шифрование может быть использовано для шифрования текстовых сообщений, файлов, а также для создания хеш-функций и контрольных сумм.

XOR шифрование может быть полезным инструментом в определенных сценариях, но обычно его применяют с осторожностью из-за его уязвимостей.

Логика алгоритма XOR включает в себя несколько шагов:

1. **Принятие входного сообщения:** В алгоритме XOR-шифрования начальным этапом является принятие входного сообщения произвольной длины, которое требуется зашифровать.
2. **Разбиение сообщения на блоки:** Входное сообщение разбивается на блоки фиксированной длины (например, 8 байт). Если длина входного сообщения не кратна длине блока, последний блок может быть дополнен для соответствия этой длине.
3. **XOR-шифрование блоков:** Каждый байт блока XOR-шифруется с соответствующим байтом ключа. Если длина ключа меньше длины блока, ключ циклически повторяется. Например, если у нас есть блок данных [A1, A2, A3, A4] и ключ [K1, K2, K3], то зашифрованный блок будет [A1^K1, A2^K2, A3^K3, A4^K1].
4. **Объединение зашифрованных блоков**: Шифрованные блоки данных объединяются в итоговое зашифрованное сообщение. Это зашифрованное сообщение может быть отправлено или сохранено в зависимости от целей использования алгоритма.
5. **Дешифрование**: Если требуется восстановление исходного сообщения, применяется обратный процесс. Зашифрованные блоки декодируются путем повторного применения XOR-шифрования с использованием того же ключа. Полученные блоки объединяются для получения исходного сообщения.

**Текст программы:**

**Файл Lab\_11.java:**

import java.util.Scanner;  
import java.nio.charset.StandardCharsets;  
  
public class Lab\_11 {  
 public static class XORCipher {  
 private final byte[] key;  
  
 public XORCipher(byte[] key) {  
 this.key = key;  
 }  
  
 public byte[] encrypt(byte[] input) {  
 byte[] output = new byte[input.length];  
 for (int i = 0; i < input.length; i++) {  
 output[i] = (byte) (input[i] ^ key[i % key.length]);  
 }  
 return output;  
 }  
 }  
  
 private final XORCipher xorCipher;  
  
 public Lab\_11(byte[] key) {  
 this.xorCipher = new XORCipher(key);  
 }  
  
 public byte[] hash(byte[] message) {  
 byte[] hash = new byte[16]; // Фиксированный размер хэша 128 бит (16 байт)  
 byte[] block = new byte[16]; // Фиксированный размер блока 128 бит (16 байт)  
  
 // Инициализация начального хэша  
 for (int i = 0; i < hash.length; i++) {  
 hash[i] = (byte) i; // Просто для примера, можно выбрать другое начальное значение  
 }  
  
 // Добавление сообщения  
 int numBlocks = (int) Math.*ceil*((double) message.length / block.length);  
 for (int i = 0; i < numBlocks; i++) {  
 int blockLength = Math.*min*(block.length, message.length - i \* block.length);  
 System.*arraycopy*(message, i \* block.length, block, 0, blockLength);  
  
 // Применение блочного шифра (XOR в данном случае)  
 hash = xorCipher.encrypt(hash);  
 for (int j = 0; j < block.length; j++) {  
 hash[j % hash.length] ^= block[j % block.length];  
 }  
 }  
  
 return hash;  
 }  
  
 public static void main(String[] args) {  
 try {  
 Scanner scanner = new Scanner(System.*in*);  
 System.*out*.print("Введите сообщение: ");  
 String inputMessage = scanner.nextLine();  
 byte[] message = inputMessage.getBytes(StandardCharsets.*UTF\_8*);  
  
 System.*out*.print("Введите ключ: ");  
 String inputKey = scanner.nextLine();  
 byte[] key = inputKey.getBytes(StandardCharsets.*UTF\_8*);  
  
 Lab\_11 hashFunction = new Lab\_11(key);  
 byte[] hashedMessage = hashFunction.hash(message);  
  
 System.*out*.println("Hashed message: " + *bytesToHex*(hashedMessage));  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 private static String bytesToHex(byte[] bytes) {  
 StringBuilder result = new StringBuilder();  
 for (byte b : bytes) {  
 result.append(String.*format*("%02x", b));  
 }  
 return result.toString();  
 }  
}