МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ

ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Чувашский государственный университет имени И.Н. Ульянова»

Факультет информатики и вычислительной техники

Кафедра вычислительной техники

Лабораторная работа №5

По дисциплине “Защита информации” на тему:

«Работа алгоритма DES в режиме CBC»

Выполнил: ст. гр. ИВТ-11-21

Еремеев С.А.

Проверил: канд. техн. наук. Ковалев С.В.

Чебоксары 2024

**Теоретическая часть**

# Цель работы

Целью работы является знакомство с особенностями работы алгоритма DES в режиме CBC.

# Задание на работу

В данной лабораторной работе необходимо программно реализовать алгоритмы шифрования и дешифрования DES в режиме CBC. 64-битовый начальный вектор C*0* следует сгенерировать, используя датчик ПСЧ, разработанный в лабораторной работе №3.

# Основные сведения

Примем, что открытый текст разбит на *n* 64-битовых блоков M*1*..M*n*. Через C*0* обозначим некоторый начальный 64-битовый вектор, загружаемый на начальных шагах шифрования и дешифрования в соответствующие буферы.

Далее для любого *i* = 0..*n* логика шифрования определяется соотношением:

C*i+1* = DES(K, M*i+1*  C*i*), а логика дешифрования - соотношением: M*i+1*  C*i* = DES-1 (K, C*i+1*), или, что то же самое: M*i+1* = C*i*  DES-1 (K, C*i+1*).

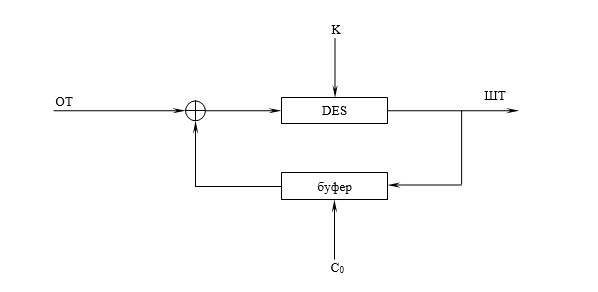


Рисунок 1 – Блок-схема алгоритма шифрования DES в режиме CBC

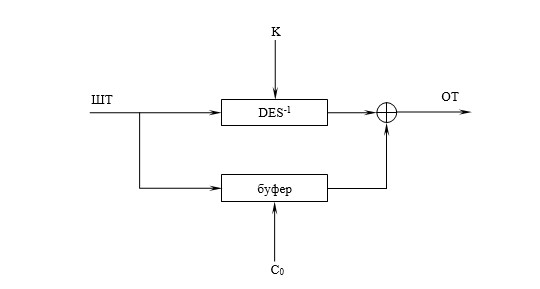


Рисунок 2 – Блок-схема алгоритма дешифрования DES в режиме CBC

Как следует из приведенных соотношений, каждый очередной блок криптограммы является функцией предшествующего. Поэтому искажение одного бита в блоке шифртекста искажает два блока, полученных в результате дешифрования. Однако поскольку искаженный блок криптограммы в процессе дешифрования гаммируется со следующим блоком открытого текста, число искажений в следующем дешифруемом блоке равно числу искажений в шифртексте.

**Практическая часть**

# Текст программы

using System;

using System.Security.Cryptography;

using System.Text;

class Program

{

static void Main()

{

Console.Write("Введите строку открытого текста: ");

string plaintext = Console.ReadLine() ?? string.Empty;

byte[] iv = GenerateIV();

Console.WriteLine("Сгенерированный IV (в двоичном виде): " + ToBinaryString(iv));

byte[] key = GenerateKey();

Console.WriteLine("Сгенерированный ключ (в двоичном виде): " + ToBinaryString(key));

byte[] encrypted = EncryptDES(plaintext, key, iv);

string encryptedText = Convert.ToBase64String(encrypted);

Console.WriteLine("Шифртекст (в base64): " + encryptedText);

string decrypted = DecryptDES(Convert.FromBase64String(encryptedText), key, iv);

Console.WriteLine("Дешифрованный текст: " + decrypted);

}

static byte[] GenerateIV()

{

int A = 25, C = 37, T0 = 7, B = 256;

byte[] iv = new byte[8];

int gamma = T0;

for (int i = 0; i < iv.Length; i++)

{

gamma = (A \* gamma + C) % B;

iv[i] = (byte)gamma;

}

return iv;

}

static byte[] GenerateKey()

{

byte[] key = new byte[8];

RandomNumberGenerator.Fill(key);

return key;

}

static byte[] EncryptDES(string plaintext, byte[] key, byte[] iv)

{

using (var des = DES.Create())

{

des.Key = key;

des.IV = iv;

des.Mode = CipherMode.CBC;

des.Padding = PaddingMode.PKCS7;

byte[] plaintextBytes = Encoding.UTF8.GetBytes(plaintext);

using (var encryptor = des.CreateEncryptor())

{

return encryptor.TransformFinalBlock(plaintextBytes, 0, plaintextBytes.Length);

}

}

}

static string DecryptDES(byte[] ciphertext, byte[] key, byte[] iv)

{

using (var des = DES.Create())

{

des.Key = key;

des.IV = iv;

des.Mode = CipherMode.CBC;

des.Padding = PaddingMode.PKCS7;

using (var decryptor = des.CreateDecryptor())

{

byte[] decryptedBytes = decryptor.TransformFinalBlock(ciphertext, 0, ciphertext.Length);

return Encoding.UTF8.GetString(decryptedBytes);

}

}

}

static string ToBinaryString(byte[] byteArray)

{

StringBuilder binaryString = new StringBuilder(byteArray.Length \* 8);

foreach (byte b in byteArray)

{

binaryString.Append(Convert.ToString(b, 2).PadLeft(8, '0') + " ");

}

return binaryString.ToString().Trim();

}

}

Функция формирования 𝑪𝟎:

static byte[] GenerateIV()

{

int A = 25, C = 37, T0 = 7, B = 256;

byte[] iv = new byte[8];

int gamma = T0;

for (int i = 0; i < iv.Length; i++)

{

gamma = (A \* gamma + C) % B;

iv[i] = (byte)gamma;

}

return iv;

}

Добавление в функцию шифрования:

static byte[] EncryptDES(string plaintext, byte[] key, byte[] iv)

{

using (var des = DES.Create())

{

des.Key = key;

des.IV = iv;

des.Mode = CipherMode.CBC;

des.Padding = PaddingMode.PKCS7;

byte[] plaintextBytes = Encoding.UTF8.GetBytes(plaintext);

using (var encryptor = des.CreateEncryptor())

{

return encryptor.TransformFinalBlock(plaintextBytes, 0, plaintextBytes.Length);

}

}

}

Добавление в функцию дешифрования:

static string DecryptDES(byte[] ciphertext, byte[] key, byte[] iv)

{

using (var des = DES.Create())

{

des.Key = key;

des.IV = iv;

des.Mode = CipherMode.CBC;

des.Padding = PaddingMode.PKCS7;

using (var decryptor = des.CreateDecryptor())

{

byte[] decryptedBytes = decryptor.TransformFinalBlock(ciphertext, 0, ciphertext.Length);

return Encoding.UTF8.GetString(decryptedBytes);

}

}

}

# Пример открытого текста и соответствующей ему шифрограммы

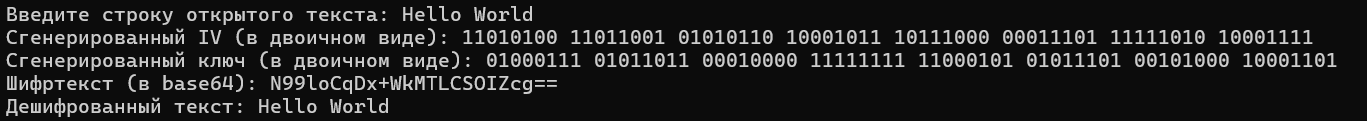


Рисунок 3 – Пример 1

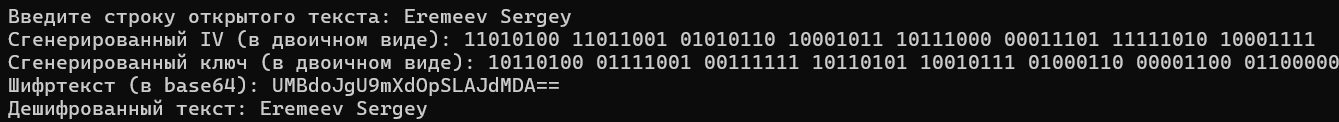


Рисунок 4 – Пример 2

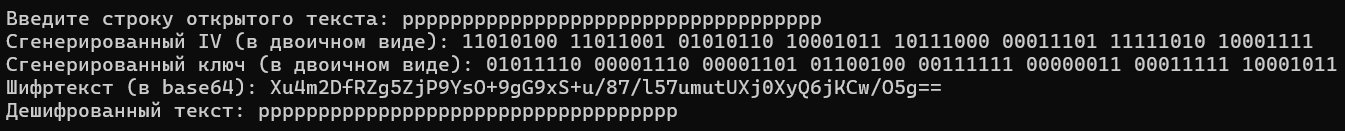


Рисунок 5 – Пример 3

# Вывод

В ходе данной лабораторной работы был рассмотрен и реализован на языке высокого программирования C# классический криптографический алгоритм DES в режиме CBC.