Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы защиты информации

ОТЧЁТ

по лабораторной работе

на тему

Симметричная криптография. Стандарт шифрования ГОСТ 28147-89

Выполнил

Студент гр. 053502

Юрьев В. А.

Проверил

Ассистент кафедры информатики

Лещенко Е. А.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_Toc147330926)

[2 Теоретические сведения 3](#_Toc147330927)

[3 Блок-схема алгоритма 4](#_Toc147330928)

[4 Демонстрация работы 5](#_Toc147330929)

[Вывод 7](#_Toc147330930)

[Приложение А 8](#_Toc147330931)

1. **ПОСТАНОВКА ЗАДАЧИ**

Реализовать программные средства шифрования и дешифрования текстовых файлов при помощи стандарта шифрования ГОСТ 28147-89 в режиме выработки имитовставки.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

ГОСТ 28147-89 «Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования» — государственный стандарт СССР (а позже межгосударственный стандарт СНГ), описывающий алгоритм симметричного блочного шифрования и режимы его работы.

ГОСТ 28147-89 представляет собой симметричный 64-битовый блочный алгоритм с 256-битовым ключом. Этот алгоритм криптографического преобразования данных предназначен для аппаратной и программной реализации, удовлетворяет криптографическим требованиям и не накладывает ограничений на степень секретности защищаемой информации.

Данные, подлежащие зашифровке, разбивают на 64-разрядные блоки.

Эти блоки разбиваются на два субблока N1 и N2 по 32 бит.

Шифруемый блок данных разбивается на две части, которые затем обрабатываются как отдельные 32-битовые целые числа без знака.

Сначала правая половина блока и подключ раунда складываются по модулю 2 в степени 32.

Затем производится поблочная подстановка.

32-битовое значение, полученное на предыдущем шаге (обозначим его S), интерпретируется как массив из восьми 4-битовых блоков кода: S=(S0,S1,S2,S3,S4,S5,S6,S7).

Далее значение каждого из восьми блоков заменяется на новое, которое выбирается по таблице замен (S-box). В каждой строке таблицы замен записаны числа от 0 до 15 в произвольном порядке без повторений. Значения элементов таблицы замен взяты от 0 до 15, так как в четырех битах, которые подвергаются подстановке, может быть записано целое число без знака в диапазоне от 0 до 15.

Значение блока S1 (четыре младших бита 32-разрядного числа S) заменится на число, стоящее на позиции, номер которой равен значению заменяемого блока.

После выполнения подстановки все 4-битовые блоки снова объединяются в единое 32-битное слово, которое затем циклически сдвигается на 11 битов влево.

Наконец, с помощью побитовой операции "сумма по модулю 2" результат объединяется с левой половиной, вследствие чего получается новая правая половина Ri.

Новая левая часть Li берется равной младшей части преобразуемого блока: Li= Ri-1.

Полученное значение преобразуемого блока рассматривается как результат выполнения одного раунда алгоритма шифрования.

**Процедуры шифрования и расшифрования**

ГОСТ 28147-89 является блочным шифром, поэтому преобразование данных осуществляется блоками в так называемых базовых циклах.

Базовые циклы заключаются в многократном выполнении для блока данных основного раунда, рассмотренного нами ранее, с использованием разных элементов ключа и отличаются друг от друга порядком использования ключевых элементов.

В каждом раунде используется один из восьми возможных 32-разрядных подключей.

**Режимы работы**

Режим простой замены: все блоки шифруются независимо друг от друга с разными подключами в разных раундах. Для одинаковых блоков сообщения М блоки шифртекста будут одинаковыми.

Режим выработки имитовставки. Данный режим используется для проверки того, что в шифротекст случайно или преднамеренно не были внесены искажения.

Имитовставка вырабатывается для M ≥ 2 блоков открытого текста по 64 бит. Алгоритм следующий:

Блок открытых данных записывается в регистры N1 и N2, после чего подвергается преобразованию, соответствующему первым 16 циклам шифрования в режиме простой замены.

К полученному результату побитно по модулю 2 прибавляется следующий блок открытых данных. Последний блок при необходимости дополняется нулями. Сумма также шифруется в соответствии с пунктом 1.

После добавления и шифрования последнего блока из результата выбирается имитовставка длиной L бит: с бита номер 32 − L до 32 (отсчёт начинается с 1). Стандарт рекомендует выбирать L исходя из того, что вероятность навязывания ложных данных равна 2−L. Имитовставка передается по каналу связи после зашифрованных блоков.

Для проверки принимающая сторона проводит аналогичную описанной процедуру. В случае несовпадения результата с переданной имитовставкой все соответствующие M блоков считаются ложными.

1. **БЛОК-СХЕМА АЛГОРИТМА**

Общая схема алгоритма приведена на рисунке 3.1.

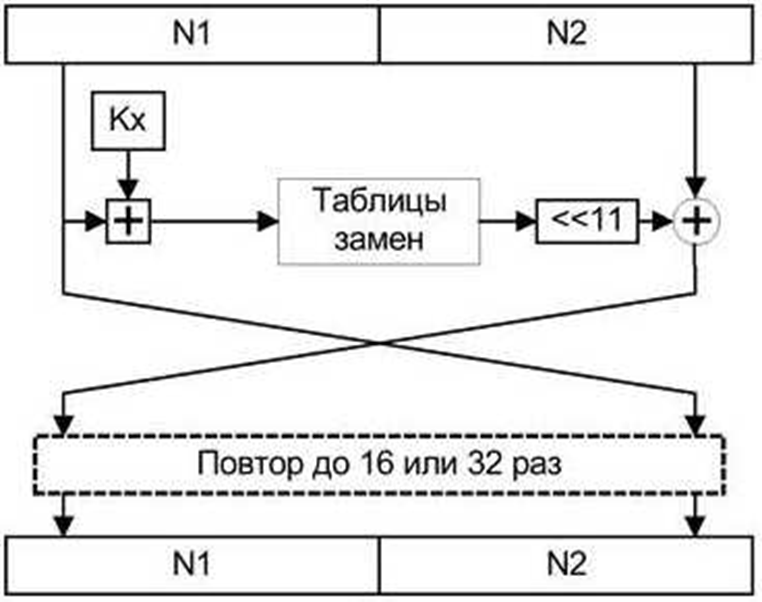


Рисунок 3.1 – Общая схема алгоритма

Схема алгоритма в режиме выработки имитовставки представлена на рисунке 3.2.

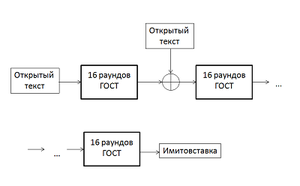


Рисунок 3.2 – Схема алгоритма в режиме выработки имитовставки

1. **ДЕМОНСТРАЦИЯ РАБОТЫ**

Пример работы программы показан на рисунке 4.1.

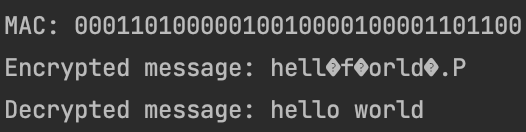


Рисунок 4.1 – Пример работы программы

Исходный текст считывается с файла input.txt, после шифрования, полученные данные записываются в файл encrypted\_message.txt. Полученная имитовставка, текущий ключ и таблица замен также сохраняются в файлы.

При расшифровке, все записанные ранее данные считываются из файлов и используются для расшифровки.

**ВЫВОД**

В ходе выполнения данной лабораторной работы был изучен алгоритм шифрования ГОСТ 28147-89. Было разработано программное средство, которое шифрует и расшифровывает текстовые файлы алгоритмом ГОСТ 28147-89 в режиме выработки имитовставки.

# **ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Код программы**

using System.Text;

namespace lab1;

public class Cryptographer

{

private List<int[]> Key { get; set; }

private List<int[]> SubstitutionTable { get; set; }

private Random Random { get; set; }

public Cryptographer()

{

Random = new Random();

Key = GenerateKey();

SubstitutionTable = GenerateSubstitutionTable();

}

private List<int[]> GenerateKey()

{

var key = Enumerable.Range(0, 8)

.Select(\_ => Enumerable.Range(0, 32)

.Select(\_ => Random.Next(0, 2))

.ToArray())

.ToList();

return key;

}

private List<int[]> GenerateSubstitutionTable()

{

var table = Enumerable.Range(0, 8)

.Select(\_ => Enumerable.Range(0, 16)

.Select(\_ => Random.Next(0, 16))

.ToArray())

.ToList();

return table;

}

private static List<int[]> SplitMessageIntoBlocks(IReadOnlyList<byte> messageBytes)

{

var blocks = new List<int[]>();

for (var i = 0; i < messageBytes.Count; i += 8)

{

var block = new byte[8];

for (var j = 0; j < 8; j++)

{

if (i + j < messageBytes.Count)

{

block[j] = messageBytes[i + j];

}

else

{

block[j] = 0;

}

}

blocks.Add(BytesToBits(block));

}

return blocks;

}

private int[] SimpleReplacementMode(int[] block, int roundCount)

{

var n1 = block[..32];

var n2 = block[32..];

for (var round = 0; round < roundCount; round++)

{

var sBits = BitXor(n1, Key[round < 24 ? round % 8 : 7 - round % 8]);

var subsequences = new List<int[]>();

for (var i = 0; i < sBits.Length; i += 4)

{

subsequences.Add(sBits[i..(i + 4)]);

}

var s = new List<int>();

for (var i = 0; i < subsequences.Count; i++)

{

var newSI = SubstitutionTable[i][BitsToInt(subsequences[i])];

s.AddRange(ByteToBits(BitConverter.GetBytes(newSI)));

}

var shiftedS = ShiftBitsLeft(s, 11);

var result = BitXor(shiftedS, n2);

block = n1.Concat(result).ToArray();

}

return block;

}

private int[] GetMAC(List<int[]> blocks)

{

var s = new int[64];

for (var i = 0; i < blocks.Count; i++)

{

s = SimpleReplacementMode(BitXor(s, blocks[i]), 16);

blocks[i] = SimpleReplacementMode(blocks[i], 32);

}

return s;

}

public void Encrypt()

{

var messageBytes = File.ReadAllBytes("./input.txt");

var blocks = SplitMessageIntoBlocks(messageBytes);

var mac = GetMAC(blocks)[..32];

Console.WriteLine($"MAC: {string.Join("", mac)}");

File.WriteAllText("./MAC.txt", string.Join("", mac));

Console.WriteLine(

$"Encrypted message: {Encoding.UTF8.GetString(BitsToBytes(blocks.SelectMany(b => b).ToArray()))}");

File.WriteAllText("./encrypted\_message.txt",

string.Join("", string.Join("\n", blocks.Select(block => string.Join("", block.Select(b => b))))));

File.WriteAllText("./key.txt", string.Join("\n", Key.Select(k => string.Join(" ", k))));

File.WriteAllText("./substitution\_table.txt", string.Join("\n", SubstitutionTable.Select(k => string.Join(" ", k))));

}

public void Decrypt()

{

Key = File.ReadAllLines("./key.txt")

.Select(line => line.Split(' ').Select(int.Parse).ToArray())

.ToList();

SubstitutionTable = File.ReadAllLines("./substitution\_table.txt")

.Select(line => line.Split(' ').Select(int.Parse).ToArray())

.ToList();

var messageBits = File.ReadAllText("./encrypted\_message.txt");

var blocks = messageBits

.Split('\n')

.Select(line => line

.Select(c => int.Parse(c.ToString())

).ToArray()

).ToList();

for (var i = 0; i < blocks.Count; i++)

{

blocks[i] = SimpleReplacementMode(blocks[i], 32);

}

var message = Encoding.UTF8.GetString(BitsToBytes(blocks.SelectMany(b => b).ToArray()));

var mac = GetMAC(blocks)[..32];

var initialMac = File.ReadAllText("./MAC.txt");

if (string.Join("", mac) != initialMac)

{

throw new Exception("MAC does not match");

}

Console.WriteLine($"Decrypted message: {message}");

}

private static byte[] BitsToBytes(IReadOnlyList<int> bits)

{

var numBytes = (bits.Count + 7) / 8;

var bytes = new byte[numBytes];

for (var i = 0; i < bits.Count; i++)

{

var byteIndex = i / 8;

var bitIndex = i % 8;

bytes[byteIndex] |= (byte)(bits[i] << (7 - bitIndex));

}

return bytes;

}

private static int[] BytesToBits(IEnumerable<byte> bytes)

{

return bytes.SelectMany(b => Enumerable.Range(0, 8).Select(i => (b >> (7 - i)) & 1)).ToArray();

}

private static int[] ByteToBits(IEnumerable<byte> bytes)

{

return bytes.Take(4).SelectMany(b => Enumerable.Range(0, 8).Select(i => (b >> i) & 1)).Reverse()

.ToArray()[^4..];

}

private static int[] BitXor(IReadOnlyList<int> array1, IReadOnlyList<int> array2)

{

if (array1.Count != array2.Count)

{

throw new ArgumentException("Arrays should have same length.");

}

var result = new int[array1.Count];

for (var i = 0; i < array1.Count; i++)

{

result[i] = array1[i] ^ array2[i];

}

return result;

}

private static int BitsToInt(IEnumerable<int> bits)

{

return Convert.ToInt32(string.Join("", bits), 2);

}

private static int[] ShiftBitsLeft(IReadOnlyList<int> bits, int shiftAmount)

{

var result = new int[bits.Count];

for (var i = shiftAmount; i < bits.Count; i++)

{

result[i - shiftAmount] = bits[i];

}

return result;

}

}