Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

|  |
| --- |
| Институт космических и информационных технологий |
| *институт* |
| Кафедра Прикладной математики и компьютерной безопасности |
| *кафедра* |

**ЛАБОРАТОРНАЯ РАБОТА №16**

|  |
| --- |
| **Программная реализация шифра DES** |
| *тема* |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Руководитель | | |  |  |  | В.И.Вайнштейн |
|  | |  |  | *подпись, дата* |  | *инициалы, фамилия* |
| Студент | КИ15-01 №031508683 | |  |  |  | М.С.Димаксян |
|  | *номер группы, зачетной книжки* | |  | *подпись, дата* |  | *инициалы, фамилия* |

Красноярск 2019

**Задание:**

Необходимо реализовать шифрование на базе алгоритма DES (на любом языке программирования. В программе предусмотреть ввод сообщения с клавиатуры (буфера обмена) и загрузку из фала любого размера и формата (.doc, .docx, .rtf, .pdf, .rar, .zip и т.п.).

**Теория:**

Основные ***достоинства алгоритма DES***:

* используется только один ключ длиной 56 битов;
* зашифровав сообщение с помощью одного пакета, для расшифровки вы можете использовать любой другой;
* относительная простота алгоритма обеспечивает высокую скорость обработки информации;
* достаточно высокая стойкость алгоритма.

DES осуществляет шифрование 64-битовых блоков данных с помощью 56-битового ключа. Расшифрование в DES является операцией обратной шифрованию и выполняется путем повторения операций шифрования в обратной последовательности. Процесс шифрования заключается в начальной перестановке битов 64-битового блока, шестнадцати циклах шифрования и, наконец, обратной перестановки битов.

**Исходный код:**

**Глобальные переменные:**

// Для ключей

public byte[] Key = new byte[0]; //Исходный ключ

int[,] G = { { 57, 49, 41, 33, 25, 17, 9 },

{ 1, 58, 50, 42, 34, 26, 18 },

{ 10, 2, 59, 51, 43, 35, 27 },

{ 19, 11, 3, 60, 52, 44, 36 },

{ 63, 55, 47, 39, 31, 23, 15 },

{ 7, 62, 54, 46, 38, 30, 22 },

{ 14, 6, 61, 53, 45, 37, 29 },

{ 21, 13, 5, 28, 20, 12, 4 } }; // Для получения C0 и D0

int[] RotOnThisStep = { 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1 }; // Число левых циклических сдвигов

int[,] H = { { 14, 17, 11, 24, 1, 5 },

{ 3, 28, 15, 6, 21, 10 },

{ 23, 19, 12, 4, 26, 8 },

{ 16, 7, 27, 20, 13, 2 },

{ 41, 52, 31, 37, 47, 55 },

{ 30, 40, 51, 45, 33, 48 },

{ 44, 49, 39, 56, 34, 53 },

{ 46, 42, 50, 36, 29, 32 } }; //Перестановка со сжатием

// Для блоков

int[,] IP = { { 58, 50, 42, 34, 26, 18, 10, 2 },

{ 60, 52, 44, 36, 28, 20, 12, 4 },

{ 62, 54, 46, 38, 30, 22, 14, 6 },

{ 64, 56, 48, 40, 32, 24, 16, 8 },

{ 57, 49, 41, 33, 25, 17, 9, 1 },

{ 59, 51, 43, 35, 27, 19, 11, 3 },

{ 61, 53, 45, 37, 29, 21, 13, 5 },

{ 63, 55, 47, 39, 31, 23, 15, 7 } }; //Начальная перестановка

int[,] IP1 = { { 40, 8, 48, 16, 56, 24, 64, 32 },

{ 39, 7, 47, 15, 55, 23, 63, 31 },

{ 38, 6, 46, 14, 54, 22, 62, 30 },

{ 37, 5, 45, 13, 53, 21, 61, 29 },

{ 36, 4, 44, 12, 52, 20, 60, 28 },

{ 35, 3, 43, 11, 51, 19, 59, 27 },

{ 34, 2, 42, 10, 50, 18, 58, 26 },

{ 33, 1, 41, 9, 49, 17, 57, 25 } }; //Конечная перестановка

int[,] E = { { 32, 1, 2, 3, 4, 5 },

{ 4, 5, 6, 7, 8, 9 },

{ 8, 9, 10, 11, 12, 13 },

{ 12, 13, 14, 15, 16, 17 },

{ 16, 17, 18, 19, 20, 21 },

{ 20, 21, 22, 23, 24, 25 },

{ 24, 25, 26, 27, 28, 29},

{ 28, 29, 30, 31, 32, 1} }; //Расширение

int[,] S1 = { { 14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7 },

{ 0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8 },

{ 4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0 },

{ 15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13 } };//S-преобразование

int[,] S2 = { { 15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10 },

{ 3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5 },

{ 0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15 },

{ 13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9 } };

int[,] S3 = { { 10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8 },

{ 13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1 },

{ 13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7 },

{ 1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12 } };

int[,] S4 = { { 7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15 },

{ 13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9 },

{ 10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4 },

{ 3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14 } };

int[,] S5 = { { 2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9 },

{ 14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6 },

{ 4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14 },

{ 11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3 } };

int[,] S6 = { { 12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11 },

{ 10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8 },

{ 9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6 },

{ 4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13 } };

int[,] S7 = { { 4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1 },

{ 13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6 },

{ 1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2 },

{ 6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12 } };

int[,] S8 = { { 13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7 },

{ 1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2 },

{ 7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8 },

{ 2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11 } };

int[,] P = { { 16, 7, 20, 21 },

{ 29, 12, 28, 17 },

{ 1, 15, 23, 26 },

{ 5, 18, 31, 10 },

{ 2, 8, 24, 14 },

{ 32, 27, 3, 9 },

{ 19, 13, 30, 6 },

{ 22, 11, 4, 25} }; //Перестановка

bool keyIsGenerated = false;

**Загрузка ключа из файла:**

private void f16\_ButtonReadKeyword\_Click(object sender, EventArgs e)

{

byte[] tmp = new byte[8];

int byteNumber = 0;

int code;

OpenFileDialog Load = new OpenFileDialog();

if (Load.ShowDialog() == DialogResult.OK)

{

System.IO.BinaryReader reader = new System.IO.BinaryReader(System.IO.File.Open(Load.FileName, System.IO.FileMode.Open), Encoding.Default);

while ((code = reader.PeekChar()) > -1 && byteNumber < 8)

{

tmp[byteNumber] = reader.ReadByte();

byteNumber++;

}

reader.Close();

if (code == -1 && byteNumber == 8)

{

Key = tmp;

f16\_fieldKeyword.Text = "Source: " + Load.FileName;

}

else

{

if (code == -1) MessageBox.Show("Слишком короткий ключ в файле", "Ошибка");

if (byteNumber == 8) MessageBox.Show("Слишком длинный ключ в файле", "Ошибка");

f16\_fieldKeyword.Text = null;

Key = new byte[0];

}

}

}

**Проверка ключа на слабость:**

private void CheckWeaknessOfKey ()

{

byte[,] weak\_keys = { {0, 0, 0, 0, 0, 0, 0, 0}, //Абсолютно слабый

{1, 1, 1, 1, 1, 1, 1, 1 }, //Слабые

{254, 254, 254, 254, 254, 254, 254, 254 },

{31, 31, 31, 31, 14, 14, 14, 14 },

{224, 224, 224, 224, 241, 241, 241, 241 },

{1, 254, 1, 254, 1, 254, 1, 254 }, //Частично слабые

{254, 1, 254, 1, 254, 1, 254, 1 },

{1, 224, 1, 224, 1, 241, 1, 241 },

{224, 1, 224, 1, 241, 1, 241, 1 },

{1, 31, 1, 31, 1, 14, 1, 14 },

{31, 1, 31, 1, 14, 1, 14, 1 },

{31, 224, 31, 224, 31, 224, 31, 224 },

{224, 241, 224, 241, 224, 241, 224, 241 },

{31, 254, 31, 254, 14, 254, 14, 254 },

{254, 31, 254, 31, 254, 14, 254, 14 },

{224, 254, 224, 254, 241, 254, 241, 254 },

{254, 224, 254, 224, 254, 241, 254, 241 }};

int[] coincidences = new int [weak\_keys.GetLength(0)];

Array.Clear(coincidences, 0, coincidences.Length);

for (int i = 0; i < weak\_keys.GetLength(0); i++) //Проход по списку слабых кл

{

for (int j = 0; j < 8; j++) //Проход по байтам введенного ключа

{

if (Key[j] == weak\_keys[i, j]) coincidences[i]++;

}

}

for (int i = 0; i < coincidences.Length; i++) //Проход по списку совпадений

{

if (coincidences[i] == 8)

{

if (i == 0)

MessageBox.Show("Выбранный ключ является абсолютно слабым", "Предупреждение");

if (i >= 1 && i <=4)

MessageBox.Show("Выбранный ключ является слабым", "Предупреждение");

if (i >= 5)

MessageBox.Show("Выбранный ключ является частично-слабым", "Предупреждение");

}

}

}

**Генерация ключа:**

private void f16\_ButtonGenKeyword\_Click(object sender, EventArgs e)

{

Random rand = new Random(DateTimeOffset.Now.Millisecond);

Key = new byte[8];

rand.NextBytes(Key);

string KeyText = null;

for (int i = 0; i < 8; i++)

{

KeyText += (char)Key[i];

}

f16\_fieldKeyword.Text = KeyText;

keyIsGenerated = true;

progressBar.Value = 0;

f16\_labelPB.Text = "Готов к работе";

f16\_labelPB.Update();

}

**Генерация раундовых ключей на основе ключа:**

private void Generate\_Keys(out BitArray[] Keys)

{

BitArray Key\_Bits = new BitArray(Key);

//Получаем битовую последовательность основного ключа

BitArray New\_Key\_Bits = new BitArray(Key\_Bits.Count);

int index = 0;

for (int i = 0; i < 8; i++)

{

for (int j = 7; j >= 0; j--)

{

New\_Key\_Bits[index] = Key\_Bits[i \* 8 + j];

index++;

}

}

//Создаем из основной последовательности бит две последовательности C0 и D0 с ипользованием перестановки G

BitArray C0 = new BitArray(28);

index = 0;

for (int i = 0; i < G.GetLength(0) / 2; i++)

{

for (int j = 0; j < G.GetLength(1); j++)

{

C0[index] = New\_Key\_Bits[G[i, j] - 1];

index++;

}

}

BitArray D0 = new BitArray(28);

index = 0;

for (int i = G.GetLength(0) / 2; i < G.GetLength(0); i++)

{

for (int j = 0; j < G.GetLength(1); j++)

{

D0[index] = New\_Key\_Bits[G[i, j] - 1];

index++;

}

}

Keys = new BitArray[16];

//Создаем 16 подключей по 48 бит

for (int iteration = 0; iteration < 16; iteration++)

{

Keys[iteration] = new BitArray(48);

//Циклический сдвиг влево на 1 или 2 бита в зависимости от номера итерации

for (int cycle = 0; cycle < RotOnThisStep[iteration]; cycle++)

{

bool temp\_C = C0[0];

bool temp\_D = D0[0];

for (int i = 0; i < 27; i++)

{

C0[i] = C0[i + 1];

D0[i] = D0[i + 1];

}

C0[27] = temp\_C;

D0[27] = temp\_D;

}

//Склеиваем Ci и Di

BitArray Ki = new BitArray(56);

for (int i = 0; i < 28; i++)

{

Ki[i] = C0[i];

}

for (int i = 28; i < 56; i++)

{

Ki[i] = D0[i - 28];

}

//Перестановка со сжатием - из 56 бит в 48

index = 0;

for (int i = 0; i < H.GetLength(0); i++)

{

for (int j = 0; j < H.GetLength(1); j++)

{

Keys[iteration][index] = Ki[H[i, j] - 1];

index++;

}

}

}

}

**Нажатие на «зашифровать»:**

private void f16\_ButtonEncryptFile\_Click(object sender, EventArgs e)

{

f16\_labelPB.Text = "Проверка ключа...";

f16\_labelPB.Update();

if (Key.Length != 8)

{

MessageBox.Show("Необходимо ввести ключ длиной 8 байт", "Ошибка");

f16\_labelPB.Text = "Готов к работе";

f16\_labelPB.Update();

goto exit\_label;

}

CheckWeaknessOfKey();

progressBar.Value = 0;

LinkedList<byte> File = new LinkedList<byte>();

f16\_labelPB.Text = "Выбор файла...";

f16\_labelPB.Update();

OpenFileDialog Load = new OpenFileDialog();

if (Load.ShowDialog() == DialogResult.OK)

{

f16\_labelPB.Text = "Считываем из файла...";

f16\_labelPB.Update();

System.IO.BinaryReader reader = new System.IO.BinaryReader(System.IO.File.Open(Load.FileName, System.IO.FileMode.Open), Encoding.Default);

while (reader.PeekChar() > -1)

{

File.AddLast(reader.ReadByte());

}

reader.Close();

}

else

{

f16\_labelPB.Text = "Готов к работе";

f16\_labelPB.Update();

return;

}

File.AddLast(0x80);

while (File.Count % 8 != 0)

{

File.AddLast(0x00);

}

byte[] Original = File.ToArray();

File.Clear();

progressBar.Maximum = Original.Length / 8;

f16\_labelPB.Text = "Зашифровываем...";

f16\_labelPB.Update();

byte[] Ciphertext;

Cryption(true, ref Original, out Ciphertext);

SaveFileDialog Save = new SaveFileDialog();

if (Save.ShowDialog() == DialogResult.OK)

{

f16\_labelPB.Text = "Сохраняем результат в файл...";

f16\_labelPB.Update();

System.IO.BinaryWriter writer = new System.IO.BinaryWriter(System.IO.File.Open(Save.FileName, System.IO.FileMode.OpenOrCreate), Encoding.Default);

foreach (var b in Ciphertext)

{

writer.Write(b);

}

writer.Close();

}

else

{

f16\_labelPB.Text = "Готов к работе";

f16\_labelPB.Update();

return;

}

f16\_labelPB.Text = "Готово.";

f16\_labelPB.Update();

exit\_label:;

}

**Нажатие на «расшифровать»:**

private void f16\_ButtonDecryptFile\_Click(object sender, EventArgs e)

{

f16\_labelPB.Text = "Проверка ключа...";

f16\_labelPB.Update();

if (Key.Length != 8)

{

MessageBox.Show("Необходимо ввести ключ длиной 8 байт", "Ошибка");

f16\_labelPB.Text = "Готов к работе";

f16\_labelPB.Update();

goto exit\_label;

}

CheckWeaknessOfKey();

progressBar.Value = 0;

LinkedList<byte> File = new LinkedList<byte>();

f16\_labelPB.Text = "Выбор файла...";

f16\_labelPB.Update();

OpenFileDialog Load = new OpenFileDialog();

if (Load.ShowDialog() == DialogResult.OK)

{

f16\_labelPB.Text = "Считываем из файла...";

f16\_labelPB.Update();

System.IO.BinaryReader reader = new System.IO.BinaryReader(System.IO.File.Open(Load.FileName, System.IO.FileMode.Open), Encoding.Default);

while (reader.PeekChar() > -1)

{

File.AddLast(reader.ReadByte());

}

reader.Close();

}

else

{

f16\_labelPB.Text = "Готов к работе";

f16\_labelPB.Update();

return;

}

byte[] Ciphertext = File.ToArray();

File.Clear();

progressBar.Maximum = Ciphertext.Length / 8;

f16\_labelPB.Text = "Расшифровываем...";

f16\_labelPB.Update();

byte[] DecryptText;

Cryption(false, ref Ciphertext, out DecryptText);

int count = 0;

for (int i = DecryptText.Length - 1; i >= DecryptText.Length - 8; i--)

{

if (DecryptText[i] == 0x80)

{

count++;

break;

}

if (DecryptText[i] == 0x00)

{

count++;

continue;

}

}

SaveFileDialog Save = new SaveFileDialog();

if (Save.ShowDialog() == DialogResult.OK)

{

f16\_labelPB.Text = "Сохраняем результат в файл...";

f16\_labelPB.Update();

System.IO.BinaryWriter writer = new System.IO.BinaryWriter(System.IO.File.Open(Save.FileName, System.IO.FileMode.OpenOrCreate), Encoding.Default);

for (int i = 0; i < DecryptText.Length - count; i++)

{

writer.Write(DecryptText[i]);

}

writer.Close();

}

else

{

f16\_labelPB.Text = "Готов к работе";

f16\_labelPB.Update();

return;

}

f16\_labelPB.Text = "Готово.";

f16\_labelPB.Update();

exit\_label:;

}

**Шифрование:**

private void Cryption(bool isEncryption, ref byte[] Source, out byte[] Result)

{

Result = new byte[Source.Length];

BitArray[] Keys;

Generate\_Keys(out Keys); //Сгенерируем 16 ключей по 48 бит на основе введенного ключа

for (int block = 0; block < Source.Length / 8; block++) //Разбиение на блоки

{

byte[] Text\_Block = new byte[8];

for (int i = 0; i < 8; i++)

{

Text\_Block[i] = Source[block \* 8 + i];

}

BitArray Text\_Block\_Bits = new BitArray(Text\_Block);

BitArray New\_Text\_Block\_Bits = new BitArray(Text\_Block\_Bits.Count);

int index = 0;

for (int i = 0; i < 8; i++)

{

for (int j = 7; j >= 0; j--)

{

New\_Text\_Block\_Bits[index] = Text\_Block\_Bits[i \* 8 + j];

index++;

}

}

BitArray L0 = new BitArray(32);

index = 0;

for (int i = 0; i < IP.GetLength(0) / 2; i++)

{

for (int j = 0; j < IP.GetLength(1); j++)

{

L0[index] = New\_Text\_Block\_Bits[IP[i, j] - 1];

index++;

}

}

BitArray R0 = new BitArray(32);

index = 0;

for (int i = IP.GetLength(0) / 2; i < IP.GetLength(0); i++)

{

for (int j = 0; j < IP.GetLength(1); j++)

{

R0[index] = New\_Text\_Block\_Bits[IP[i, j] - 1];

index++;

}

}

if (isEncryption)

{

FeistelNetwork(ref L0, ref R0, ref Keys);

}

else

{

FeistelNetwork\_Reverse(ref L0, ref R0, ref Keys);

}

BitArray LR = new BitArray(64);

for (int i = 0; i < 32; i++)

{

LR[i] = L0[i];

}

for (int i = 32; i < 64; i++)

{

LR[i] = R0[i - 32];

}

BitArray ResultBlock\_Bits = new BitArray(64);

index = 0;

for (int i = 0; i < IP1.GetLength(0); i++)

{

for (int j = 0; j < IP1.GetLength(1); j++)

{

ResultBlock\_Bits[index] = LR[IP1[i, j] - 1];

index++;

}

}

byte[] ResultBlock\_Bytes = Convert\_Bits\_To\_Bytes(ResultBlock\_Bits);

for (int i = 0; i < 8; i++)

{

Result[block \* 8 + i] = ResultBlock\_Bytes[i];

}

progressBar.Value++;

if (((progressBar.Maximum > 100 && progressBar.Value % (progressBar.Maximum/100) == 0) ||

(progressBar.Value == progressBar.Maximum)) && isEncryption) //Составляет ровно n% или 100% и относится к зашифрованию

f16\_labelPB.Text = "Зашифровываем... Выполнено " + Convert.ToString(progressBar.Value) + " из " + Convert.ToString(progressBar.Maximum);

if (((progressBar.Maximum > 100 && progressBar.Value % (progressBar.Maximum / 100) == 0) ||

(progressBar.Value == progressBar.Maximum)) && !isEncryption)

f16\_labelPB.Text = "Расшифровываем... Выполнено " + Convert.ToString(progressBar.Value) + " из " + Convert.ToString(progressBar.Maximum);

f16\_labelPB.Update();

}

}

**Сеть Фейстеля:**

private void FeistelNetwork(ref BitArray L0, ref BitArray R0, ref BitArray[] Keys)

{

BitArray Li = new BitArray(32);

BitArray Ri = new BitArray(32);

for (int iteration = 0; iteration < 16; iteration++)

{

Li = R0;

Ri = L0.Xor(f(R0, Keys[iteration]));

L0 = Li;

R0 = Ri;

}

}

**Сеть Фейстеля (обратный алгоритм):**

private void FeistelNetwork\_Reverse(ref BitArray L16, ref BitArray R16, ref BitArray[] Keys)

{

BitArray Li = new BitArray(32);

BitArray Ri = new BitArray(32);

for (int iteration = 15; iteration >= 0; iteration--)

{

Ri = L16;

Li = R16.Xor(f(L16, Keys[iteration]));

R16 = Ri;

L16 = Li;

}

}

**Функция f:**

private BitArray f(BitArray R0, BitArray Ki)

{

BitArray Ri = new BitArray(48);

//Применяем расширитель E

int index = 0;

for (int i = 0; i < E.GetLength(0); i++)

{

for (int j = 0; j < E.GetLength(1); j++)

{

Ri[index] = R0[E[i, j] - 1];

index++;

}

}

//Применяем XOR к Ключу и расширенным данным

Ri = Ri.Xor(Ki);

BitArray Result = new BitArray(32);

index = 0;

//Разбиваем на восемь 6-битовых блоков B1-B8 и используем S-боксы

for (int cycle = 0; cycle < 8; cycle++)

{

string Bi = null;//6

string temp\_line = null;//2

string temp\_column = null;//4

for (int i = cycle \* 6; i < ((cycle \* 6) + 6); i++)

{

Bi += Convert.ToInt32(Ri[i]);

}

temp\_line += Bi[0];

temp\_line += Bi[5];

int line = Convert\_BitString\_To\_Bytes(temp\_line);

for (int i = 1; i < 5; i++)

{

temp\_column += Bi[i];

}

int column = Convert\_BitString\_To\_Bytes(temp\_column);

int[] Binary\_Result = null;

switch (cycle)

{

case 0:

Binary\_Result = Convert\_Int\_To\_Bits(S1[line, column]);

break;

case 1:

Binary\_Result = Convert\_Int\_To\_Bits(S2[line, column]);

break;

case 2:

Binary\_Result = Convert\_Int\_To\_Bits(S3[line, column]);

break;

case 3:

Binary\_Result = Convert\_Int\_To\_Bits(S4[line, column]);

break;

case 4:

Binary\_Result = Convert\_Int\_To\_Bits(S5[line, column]);

break;

case 5:

Binary\_Result = Convert\_Int\_To\_Bits(S6[line, column]);

break;

case 6:

Binary\_Result = Convert\_Int\_To\_Bits(S7[line, column]);

break;

case 7:

Binary\_Result = Convert\_Int\_To\_Bits(S8[line, column]);

break;

}

for (int i = 0; i < 4; i++)

{

if (Binary\_Result[i] == 1)

{

Result[index] = true;

}

index++;

}

}

//Окончательная перестановка P

BitArray New\_Result = new BitArray(32);

index = 0;

for (int i = 0; i < P.GetLength(0); i++)

{

for (int j = 0; j < P.GetLength(1); j++)

{

New\_Result[index] = Result[P[i, j] - 1];

index++;

}

}

return New\_Result;

}

**Вспомогательные функции:**

private static byte[] Convert\_Bits\_To\_Bytes(BitArray Bits)

{

byte[] Text\_Bytes = new byte[8];

for (int i = 0; i < 8; i++)

{

string temp = null;

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

{

temp += Convert.ToInt32(Bits[i \* 8 + j]);

}

Text\_Bytes[i] = Convert.ToByte(Convert\_BitString\_To\_Bytes(temp));

}

return Text\_Bytes;

}

private static int Convert\_BitString\_To\_Bytes(string bits)

{

int degree = bits.Length - 1;

int result = 0;

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

{

if (bits[i] == '1')

{

result += Convert.ToInt32(Math.Pow(2, degree));

}

degree--;

}

return result;

}

private static int[] Convert\_Int\_To\_Bits(int amount)

{

int[] binary = new int[4];

int index = 3;

while (amount > 0)

{

binary[index] = amount % 2;

amount /= 2;

index--;

}

return binary;

}

**Результат работы программы:**

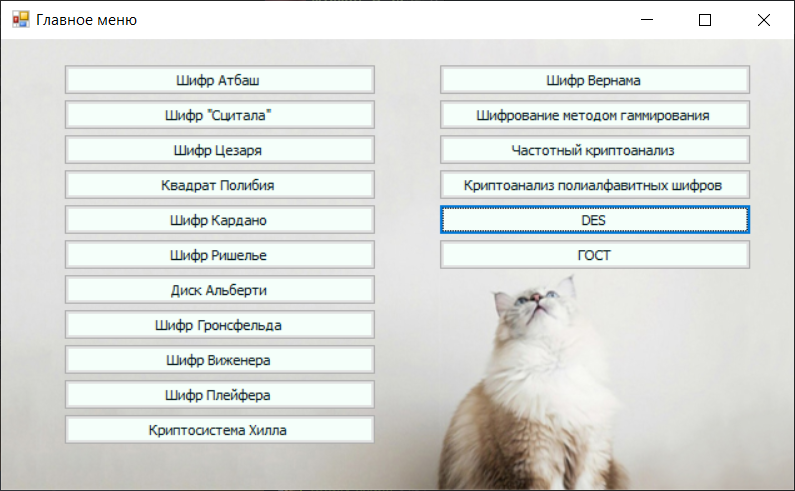


Рисунок 1. Главное меню программы.

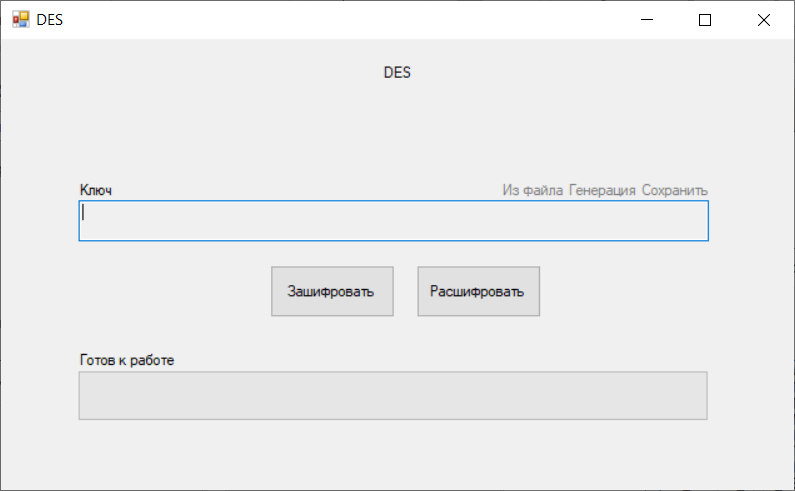


Рисунок 2. Окно шифрования DES.

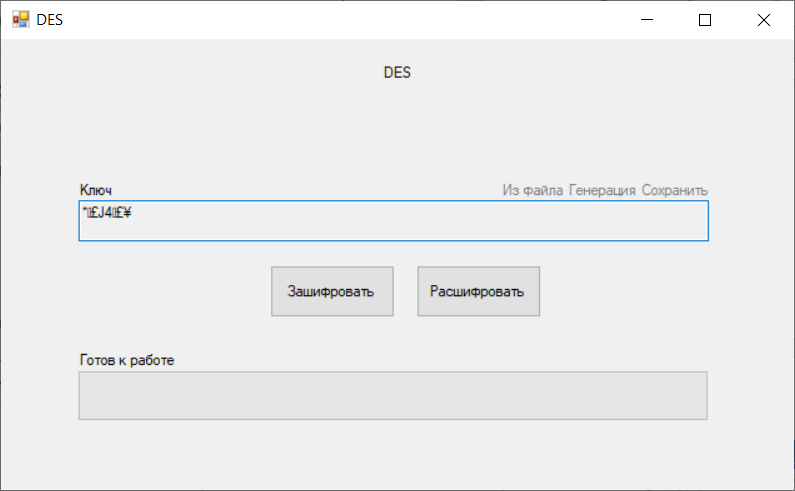


Рисунок 3. Генерация ключа.

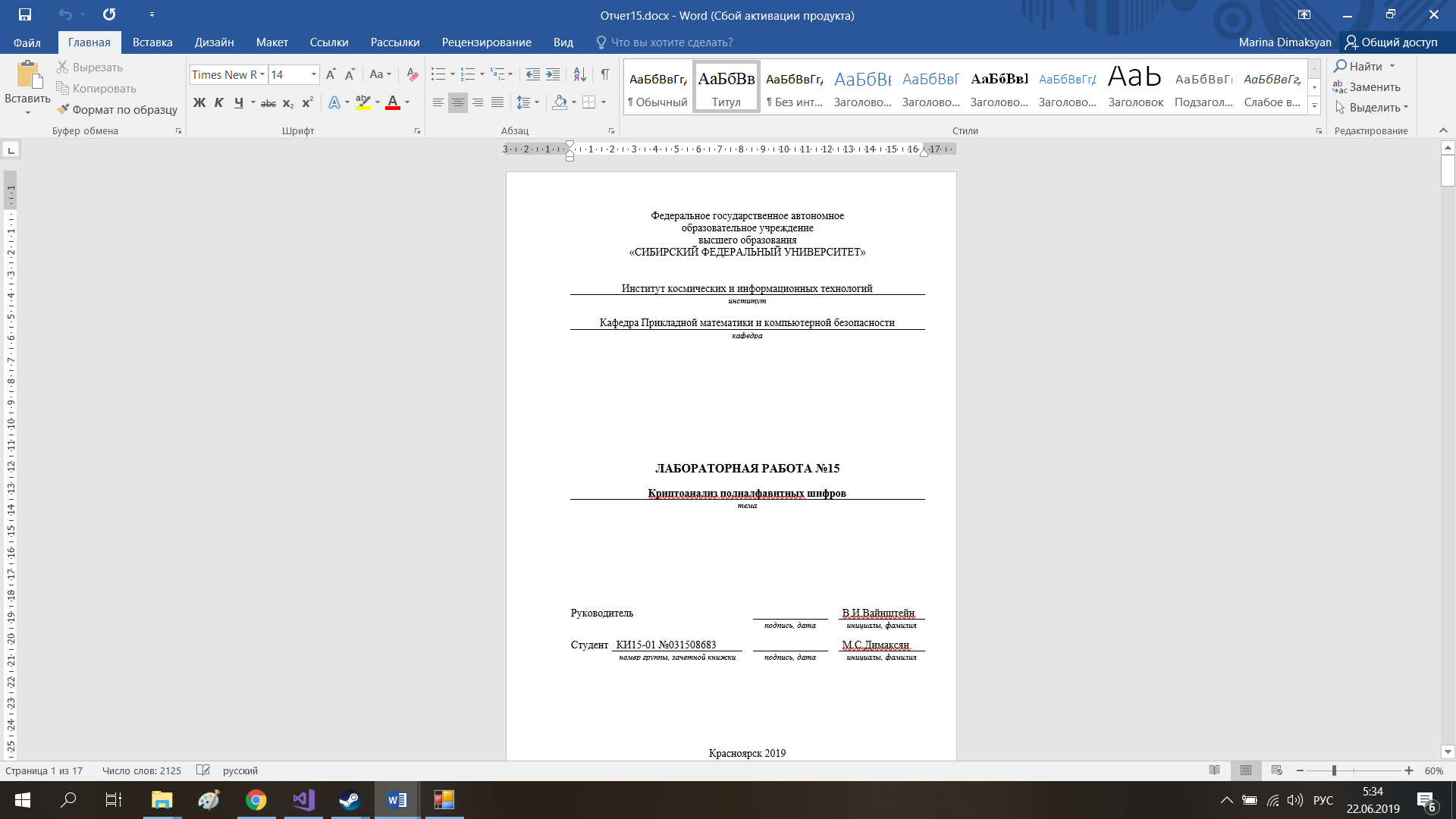


Рисунок 4. Содержимое исходного файла «Отчет15.docx».

Нажмем «Зашифровать», выберем для зашифрования данный файл.

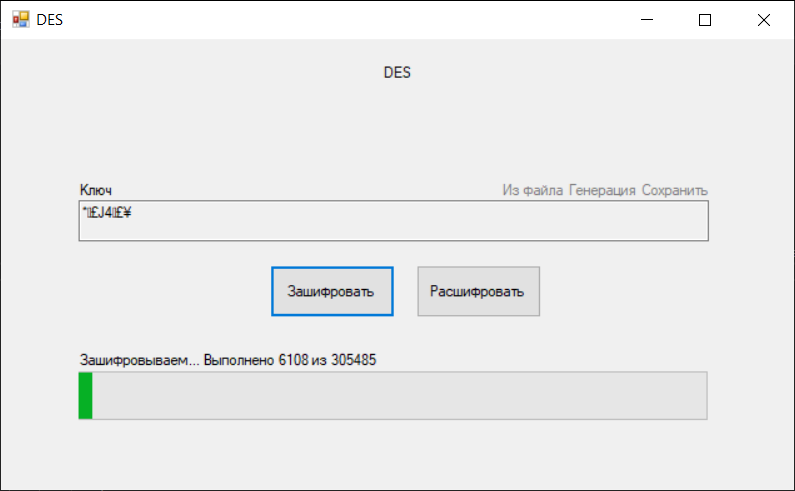
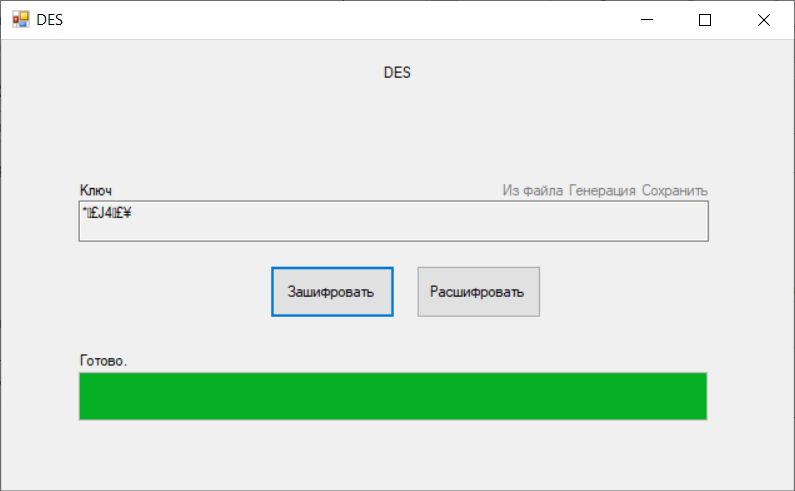
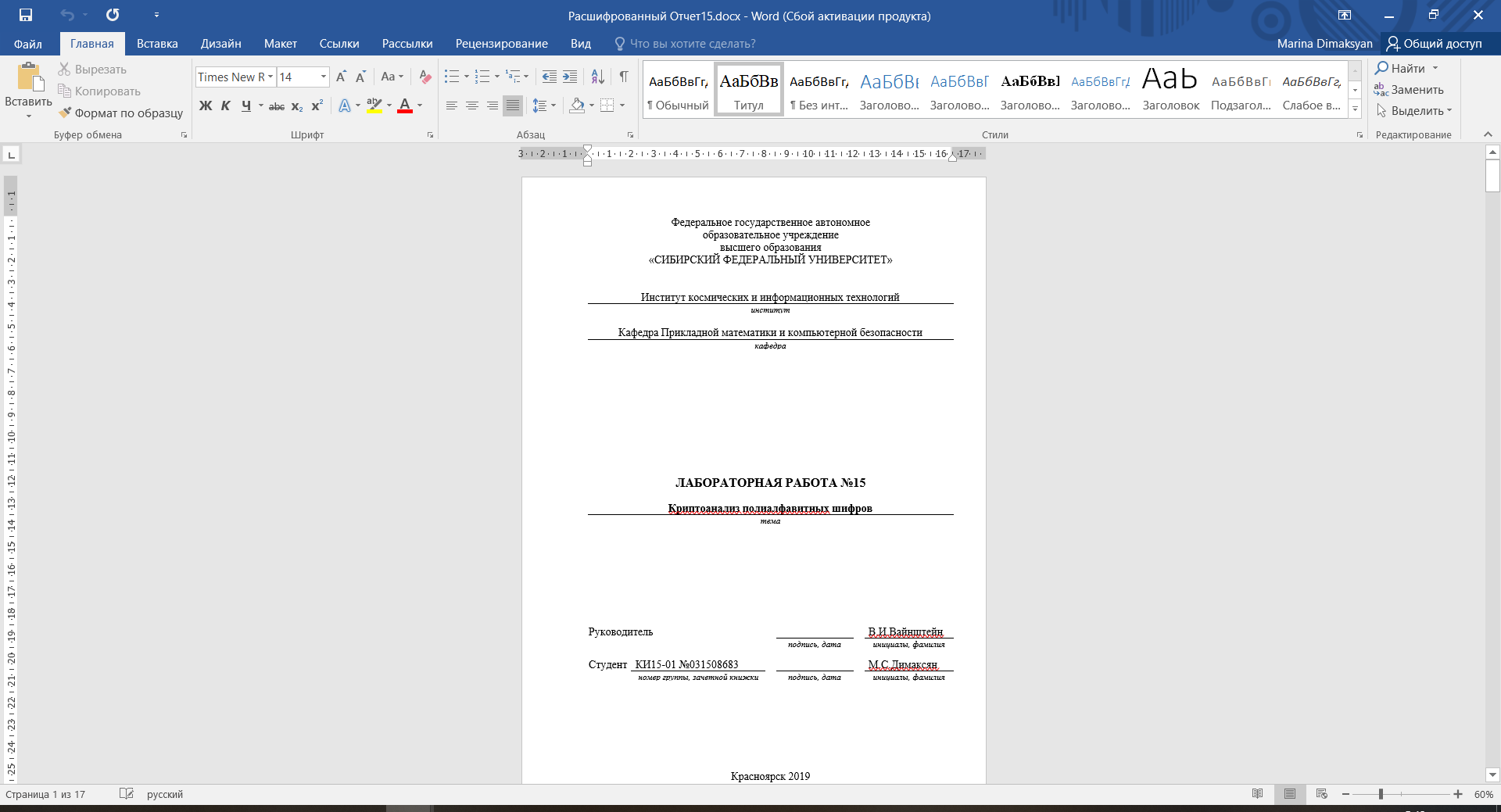


Рисунок 5. Процесс зашифрования.

Сохраним результат в файл «Зашифрованный Отчет15».

 Рисунок 6. Отображение результата на прогресс-баре.

Расшифруем файл «Зашифрованный Отчет15» и сохраним результат в файл «Расшифрованный Отчет15».

Рисунок 7. Содержимое файла «Расшифрованный Отчет15».

**Вывод:**

В ходе выполнения лабораторной работы я ознакомилась с алгоритмом шифрования DES, реализовала на практике программу по зашифрованию и расшифрованию данным шифром на языке C#. Данные навыки я могу применить при реализации других шифров и при дальнейшем использовании шифрования DES.