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 №17**

|  |
| --- |
| **Программная реализация шифра ГОСТ** |
| *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 ГОСТ28147-89 (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:**

Для зашифровывания в этом режиме 64-битный блок открытого текста сначала разбивается на две половины (младшие биты — A, старшие биты — B). На i-ом цикле используется подключ Ki:



Для генерации подключей исходный 256-битный ключ разбивается на восемь 32-битных блоков: K1…K8.

Ключи K9…K24 являются циклическим повторением ключей K1…K8 (нумеруются от младших битов к старшим). Ключи K25…K32 являются ключами K8…K1.

После выполнения всех 32 раундов алгоритма, блоки A33 и B33 склеиваются (обратите внимание, что старшим блоком становится A33, а младшим — B33) — это есть результат работы алгоритма.

**Функция {\displaystyle f(A\_{i},K\_{i})}** вычисляется следующим образом:

Ai и Ki складываются по модулю 232.

Результат разбивается на восемь 4-битовых подпоследовательностей, каждая из которых поступает на вход своего *узла таблицы замен* (в порядке возрастания старшинства битов), называемого ниже *S-блоком*. Общее количество S-блоков стандарта — восемь, то есть столько же, сколько и подпоследовательностей. Каждый *S-блок* представляет собой перестановку чисел от 0 до 15 (конкретный вид S-блоков в стандарте не определен). Первая 4-битная подпоследовательность попадает на вход первого S-блока, вторая — на вход второго и т. д.

Если узел *S-блока* выглядит так:

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

и на входе S-блока 0, то на выходе будет 1, если 4, то на выходе будет 5, если на входе 12, то на выходе 6 и т. д.

Выходы всех восьми S-блоков объединяются в 32-битное слово, затем всё слово циклически сдвигается влево (к старшим разрядам) на 11 битов.

**Исходный код:**

**Глобальные переменные:**

byte[][] Key = new byte[8][];

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

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

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

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

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

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

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

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

**Загрузка ключа из файла:**

private void f17\_ButtonReadKeyword\_Click(object sender, EventArgs e)

{

Key = new byte[8][];

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

{

Key[i] = new byte[4];

}

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);

int i = 0, j = 0;

while ((code = reader.PeekChar()) > -1 && (byteNumber = i\*4+j) < 32)

{

Key[i][j] = reader.ReadByte();

j++;

if (j == 4) { j = 0; i++; }

}

reader.Close();

if (code == -1 && byteNumber == 31)

{

f17\_fieldKeyword.Text = "Source: " + Load.FileName;

}

else

{

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

if (byteNumber == 31) MessageBox.Show("Слишком длинный ключ в файле", "Ошибка");

f17\_fieldKeyword.Text = null;

Key = new byte[0][];

}

}

}

**Генерация ключа:**

private void f17\_ButtonGenKeyword\_Click(object sender, EventArgs e)

{

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

Key = new byte[8][];

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

{

Key[i] = new byte[4];

rand.NextBytes(Key[i]);

}

string KeyText = null;

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

{

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

{

KeyText += (char)Key[i][j];

}

}

f17\_fieldKeyword.Text = KeyText;

}

**Зашифрование:**

private void f17\_ButtonEncryptFile\_Click(object sender, EventArgs e)

{

progressBar.Value = 0;

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

f17\_labelPB.Text = "Выбор файла...";

f17\_labelPB.Update();

OpenFileDialog Load = new OpenFileDialog();

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

{

f17\_labelPB.Text = "Считываем из файла...";

f17\_labelPB.Update();

System.IO.FileInfo Inform = new System.IO.FileInfo(Load.FileName);

progressBar.Maximum = Convert.ToInt32(Inform.Length / 8) + 1;

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

f17\_labelPB.Text = "Зашифровываем...";

f17\_labelPB.Update();

byte[] SourceL = new byte[4];

byte[] SourceR = new byte[4];

byte[] Result = new byte[8];

for (long block = 0; block < (Inform.Length / 8); block++) //Блоки по 64 бит

{

SourceL = reader.ReadBytes(4); //4\*8=32

SourceR = reader.ReadBytes(4);

Result = FeistelNetwork(SourceL, SourceR);

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

{

ResultFile.AddLast(Result[i]);

}

progressBar.Value++;

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

(progressBar.Value == progressBar.Maximum))) //Составляет ровно n% или 100%

f17\_labelPB.Text = "Зашифровываем... Выполнено " + Convert.ToString(progressBar.Value) + " из " + Convert.ToString(progressBar.Maximum);

f17\_labelPB.Update();

}

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

while (reader.PeekChar() > -1)

{

Tail.AddLast(reader.ReadByte());

}

Tail.AddLast(0x80);

while (Tail.Count != 8)

{

Tail.AddLast(0x00);

}

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

{

SourceL[i] = Tail.ElementAt(i);

}

for (int i = 4; i < 8; i++)

{

SourceR[i - 4] = Tail.ElementAt(i);

}

Tail.Clear();

Result = FeistelNetwork(SourceL, SourceR);

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

{

ResultFile.AddLast(Result[i]);

}

progressBar.Value++;

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

(progressBar.Value == progressBar.Maximum))) //Составляет ровно n% или 100%

f17\_labelPB.Text = "Зашифровываем... Выполнено " + Convert.ToString(progressBar.Value) + " из " + Convert.ToString(progressBar.Maximum);

f17\_labelPB.Update();

reader.Close();

}

else

{

f17\_labelPB.Text = "Готов к работе";

f17\_labelPB.Update();

return;

}

SaveFileDialog Save = new SaveFileDialog();

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

{

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

f17\_labelPB.Text = "Сохраняем результат в файл...";

f17\_labelPB.Update();

foreach (var Byte in ResultFile)

{

writer.Write(Byte);

}

writer.Close();

}

else

{

f17\_labelPB.Text = "Готов к работе";

f17\_labelPB.Update();

return;

}

f17\_labelPB.Text = "Готово.";

f17\_labelPB.Update();

}

**Расшифрование:**

private void f17\_ButtonDecryptFile\_Click(object sender, EventArgs e)

{

progressBar.Value = 0;

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

f17\_labelPB.Text = "Выбор файла...";

f17\_labelPB.Update();

OpenFileDialog Load = new OpenFileDialog();

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

{

f17\_labelPB.Text = "Считываем из файла...";

f17\_labelPB.Update();

System.IO.FileInfo Inform = new System.IO.FileInfo(Load.FileName);

progressBar.Maximum = Convert.ToInt32(Inform.Length / 8);

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

f17\_labelPB.Text = "Расшифровываем...";

f17\_labelPB.Update();

byte[] SourceL = new byte[4];

byte[] SourceR = new byte[4];

byte[] Result = new byte[8];

for (long block = 0; block < (Inform.Length / 8); block++) //Блоки по 64 бита

{

SourceL = reader.ReadBytes(4); //4\*8=32

SourceR = reader.ReadBytes(4);

Result = FeistelNetwork\_Reverse(SourceL, SourceR);

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

{

ResultFile.AddLast(Result[i]);

}

progressBar.Value++;

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

(progressBar.Value == progressBar.Maximum))) //Составляет ровно n% или 100%

f17\_labelPB.Text = "Расшифровываем... Выполнено " + Convert.ToString(progressBar.Value) + " из " + Convert.ToString(progressBar.Maximum);

f17\_labelPB.Update();

}

reader.Close();

}

else

{

f17\_labelPB.Text = "Готов к работе";

f17\_labelPB.Update();

return;

}

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

{

if (ResultFile.Last() == 0x80)

{

ResultFile.RemoveLast();

break;

}

if (ResultFile.Last() == 0x00)

{

ResultFile.RemoveLast();

continue;

}

}

SaveFileDialog Save = new SaveFileDialog();

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

{

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

f17\_labelPB.Text = "Сохраняем результат в файл...";

f17\_labelPB.Update();

foreach (var Byte in ResultFile)

{

writer.Write(Byte);

}

writer.Close();

}

else

{

f17\_labelPB.Text = "Готов к работе";

f17\_labelPB.Update();

return;

}

f17\_labelPB.Text = "Готово.";

f17\_labelPB.Update();

}

**Сеть Фейстеля:**

private byte[] FeistelNetwork(byte[] L0, byte[] R0)

{

byte[] Li = new byte[4];

byte[] Ri = new byte[4];

byte[] tmp;

for (int iteration = 0; iteration < 24; iteration++)

{

tmp = f(R0, (iteration % 8));

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

{

Ri[i] = Convert.ToByte(L0[i] ^ tmp[i]);

}//Ri = L0.Xor(f(ref R0, (iteration % 8)));

R0.CopyTo(Li, 0);

Li.CopyTo(L0, 0);

Ri.CopyTo(R0, 0);

}

for (int iteration = 24; iteration < 31; iteration++)

{

tmp = f(R0, (31 - iteration));

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

{

Ri[i] = Convert.ToByte(L0[i] ^ tmp[i]);

}//Ri = L0.Xor(f(ref R0, (31 - iteration)));

R0.CopyTo(Li, 0);

Li.CopyTo(L0, 0);

Ri.CopyTo(R0, 0);

}

R0.CopyTo(Ri, 0);

tmp = f(R0, 0);

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

{

Li[i] = Convert.ToByte(L0[i] ^ tmp[i]);

}//Li = L0.Xor(f(ref R0, 0));

byte[] Result = new byte[8];

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

{

Result[i] = Li[i];

Result[i + 4] = Ri[i];

}

return Result;

}

**Сеть Фейстеля (обратный алгоритм):**

private byte[] FeistelNetwork\_Reverse(byte[] L32, byte[] R32)

{

byte[] Li = new byte[4];

byte[] Ri = new byte[4];

byte[] tmp;

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

{

tmp = f(R32, iteration);

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

{

Ri[i] = Convert.ToByte(L32[i] ^ tmp[i]);

}//Ri = L32.Xor(f(ref R32, iteration));

R32.CopyTo(Li, 0);

Li.CopyTo(L32, 0);

Ri.CopyTo(R32, 0);

}

for (int iteration = 8; iteration < 31; iteration++)

{

tmp = f(R32, ((31 - iteration) % 8));

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

{

Ri[i] = Convert.ToByte(L32[i] ^ tmp[i]);

}//Ri = L32.Xor(f(ref R32, ((31 - iteration) % 8)));

R32.CopyTo(Li, 0);

Li.CopyTo(L32, 0);

Ri.CopyTo(R32, 0);

}

R32.CopyTo(Ri, 0);

tmp = f(R32, 0);

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

{

Li[i] = Convert.ToByte(L32[i] ^ tmp[i]);

}//Li = L32.Xor(f(ref R32, 0));

byte[] Result = new byte[8];

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

{

Result[i] = Li[i];

Result[i + 4] = Ri[i];

}

return Result;

}

**Функция f:**

private byte[] f(byte[] R0, int number)

{

//Складываем по модулю 2^32 блок и ключ

uint A = BitConverter.ToUInt32(R0.Reverse().ToArray(), 0);

A += BitConverter.ToUInt32(Key[number].Reverse().ToArray(), 0);

R0 = BitConverter.GetBytes(A);

R0 = R0.Reverse().ToArray();

BitArray Block\_Bits = new BitArray(R0);

BitArray NewBlock\_Bits = new BitArray(32);

int index = 0;

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

{

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

{

NewBlock\_Bits[index] = Block\_Bits[i \* 8 + j];

index++;

}

}

Block\_Bits.SetAll(false);

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

{

string temp = null;

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

{

temp += Convert.ToInt32(NewBlock\_Bits[cycle \* 4 + i]);

}

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

byte[] Binary\_S\_Box = Convert\_Int\_To\_Bits(S\_Box[cycle, column]);

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

{

if (Binary\_S\_Box[i] == 1)

{

Block\_Bits[cycle \* 4 + i] = true;

}

}

}

for (int cycle = 0; cycle < 11; cycle++)

{

bool temp = Block\_Bits[0];

for (int i = 0; i < 31; i++)

{

Block\_Bits[i] = Block\_Bits[i + 1];

}

Block\_Bits[31] = temp;

}

byte[] Out = Convert\_BitArray\_To\_Bytes(Block\_Bits);

return Out;

}

**Вспомогательные функции:**

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 byte[] Convert\_Int\_To\_Bits(int A)

{

byte[] binary = new byte[4];

int index = 3;

while (A > 0)

{

binary[index] = Convert.ToByte(A % 2);

A /= 2;

index--;

}

return binary;

}

private static byte[] Convert\_BitArray\_To\_Bytes(BitArray bits)

{

byte[] Result = new byte[4];

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

{

int degree = 7;

int sum = 0;

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

{

if (bits[i \* 4 + j] == true)

{

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

}

degree--;

}

Result[i] = Convert.ToByte(sum);

}

return Result;

}

**Результат работы программы:**

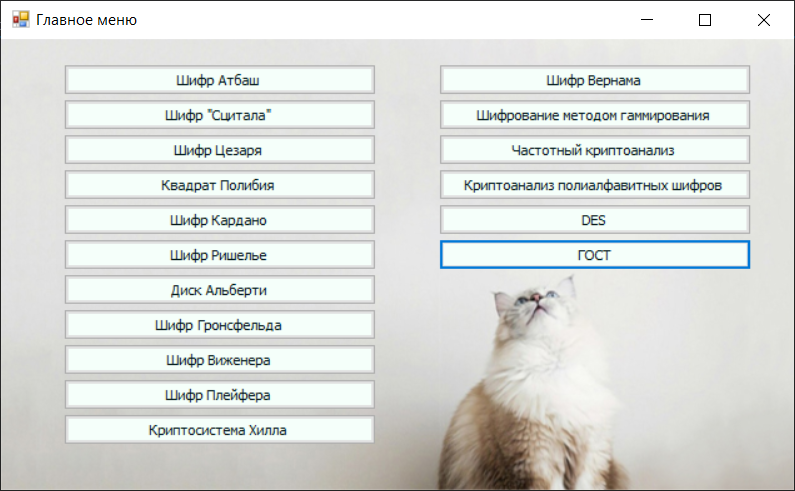


Рисунок 1. Главное меню программы.

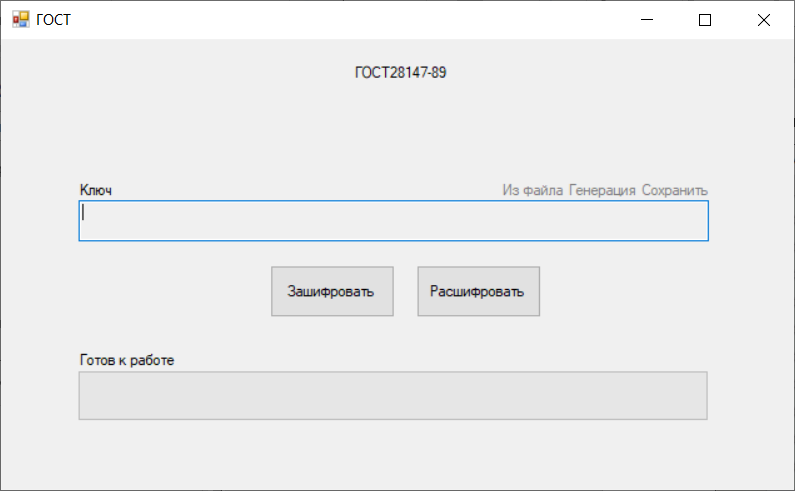


Рисунок 2. Окно шифрования DES.

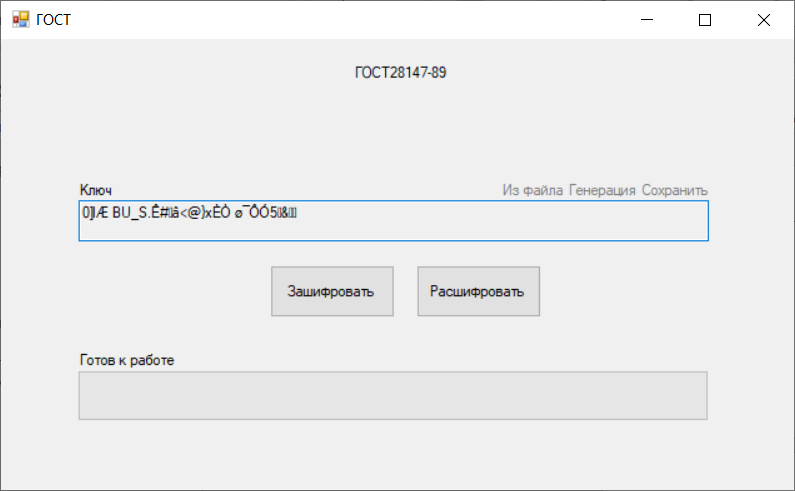


Рисунок 3. Генерация ключа.

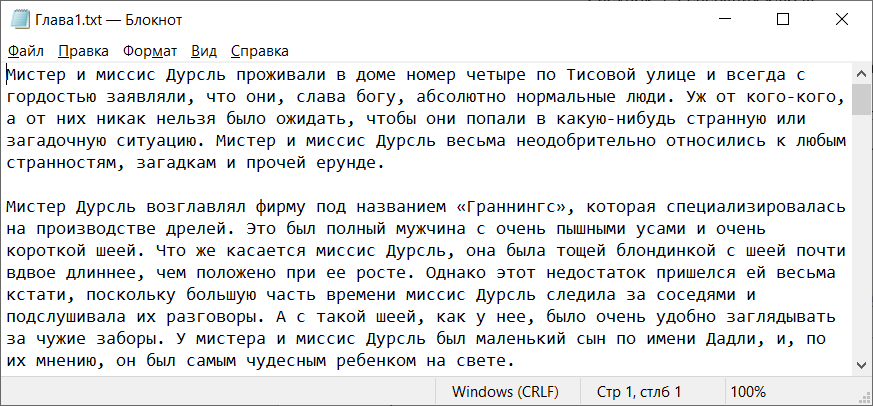


Рисунок 4. Содержимое исходного файла «Глава1.txt».

Нажмем «Зашифровать», выберем для зашифрования данный файл.

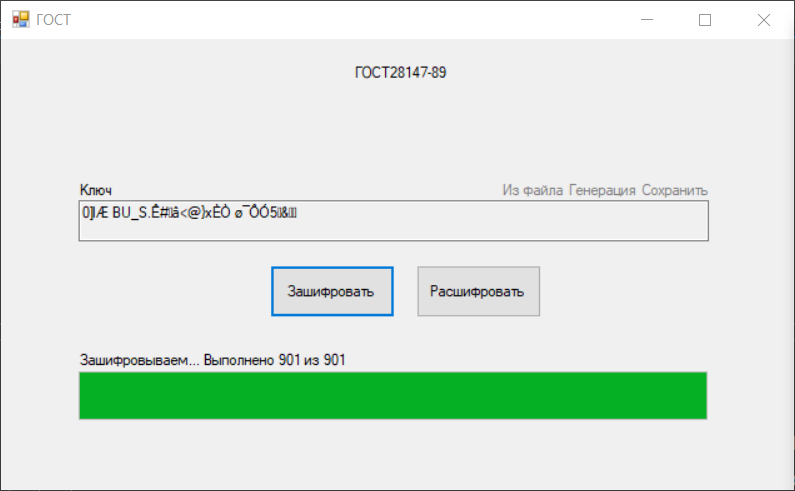


Рисунок 5. Процесс зашифрования проходит мгновенно на небольших текстах.

Сохраним результат в файл «Глава1-2.txt».

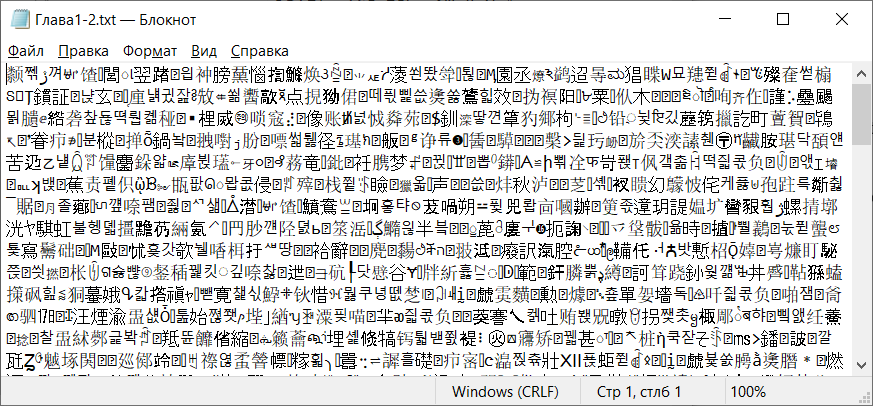


Рисунок 6. Содержимое файла «Глава1-2.txt».

Расшифруем файл «Глава1-2.txt» и сохраним результат в файл «Глава1-3.txt».

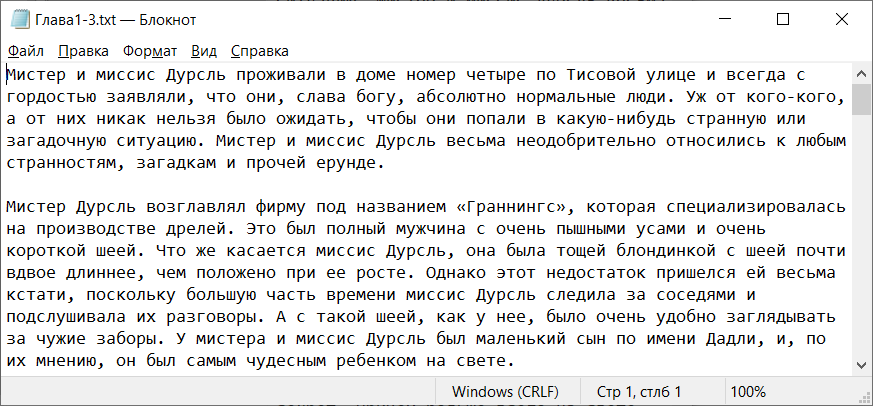


Рисунок 7. Содержимое файла «Глава1-3.txt».

**Вывод:**

В ходе выполнения лабораторной работы я ознакомилась с алгоритмом шифрования ГОСТ28147-89, 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 данным алгоритмом.