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 №13**

|  |
| --- |
| **Шифрование методом гаммирования** |
| *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:**

Реализовать шифрование методом гаммирования на любом языке программирования.

**Теория:**

Гамми́рование — метод [симметричного](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BC%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%87%D0%BD%D1%8B%D0%B5_%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B) шифрования, заключающийся в «наложении» последовательности, состоящей из псевдослучайных чисел, на открытый текст. Последовательность псевдослучайных чисел называется гамма-последовательностью и используется для зашифровывания и расшифровывания данных. Суммирование, обычно, выполняется в каком-либо конечном поле.

**Исходный код:**

**Информация об алфавите:**

public string alf = "abcdefghijklmnopqrstuvwxyz" + "ABCDEFGHIJKLMNOPQRSTUVWXYZ" +

"абвгдеёжзийклмнопрстуфхцчшщъыьэюя" + "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ" +

"0123456789" + " .,?!~`@#№$;%^:&\*()-\_+=[]{}|/<>" + "„…†‡€‰‘’“”•—" +

"™¤¦§©Є«»¬®°±µ¶·£¥≠≤≥÷×∞αβπΩ∑←↑→↓↔↕↨∆∏√∟∩∫≡⌂⌠⌡◊♀♂♠♣♥♦♪♫ⱰⱭⱲⱳⱴꞈ¼½¾²³¿ˇ˅ˣˢΞΘΛΨλζεδγϕϞϬϼѡ⅓";

int bitsForOneSymbol = 8;

bool isFromFileOrig = false;

bool isFromFileCipher = false;

**Генерация ключа:**

private void f13\_ButtonGenKeyword\_Click(object sender, EventArgs e)

{

int previous = Convert.ToInt32(f13\_paramX0.Value);

int A = Convert.ToInt32(f13\_paramA.Value);

int B = Convert.ToInt32(f13\_paramB.Value);

string generated\_key = "";

int sourceLen = f13\_fieldOriginal.TextLength;

if (sourceLen < f13\_fieldCiphertext.TextLength) sourceLen = f13\_fieldCiphertext.TextLength;

if (sourceLen == 0)

{

MessageBox.Show("Сначала введите текст", "Предупреждение");

f13\_fieldOriginal.Text = null;

goto exitkey;

}

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

{

generated\_key = generated\_key + alf[(A \* previous + B) % alf.Length];

previous = (A \* previous + B) % alf.Length;

}

f13\_fieldKeyword.Text = generated\_key;

exitkey:;

}

**Функция перевода строки в последовательность битов:**

private byte [,] TextToBits (string text)

{

byte[] decimal\_code = new byte[text.Length];

for (int i = 0; i < decimal\_code.Length; i++)

decimal\_code[i] = (byte)alf.IndexOf(text[i]);

//от буквенной строки перейдем к десятичным числам - номеру символа в алфавите

byte[,] bin\_code = new byte[text.Length, bitsForOneSymbol];

//создадим ячейки под двоичный код - по 8 бит на букву

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

{

double curCode = decimal\_code[i];

for (int j = bitsForOneSymbol - 1; j >= 0; j--)

if (curCode - Math.Pow(2, j) >= 0)

{

curCode -= Math.Pow(2, j);

bin\_code[i,j] = 1;

}

else

{

bin\_code[i, j] = 0;

}

}

return bin\_code;

}

**Функция перевода последовательности битов в строку**

private string BitsToText (byte[,] bytes)

{

byte [] decimal\_code = new byte [bytes.Length / bitsForOneSymbol];

Array.Clear(decimal\_code, 0, decimal\_code.Length);

for (int i = 0; i < decimal\_code.Length; i++)

for (int j = bitsForOneSymbol - 1; j >= 0; j--)

{

decimal\_code[i] += (byte)(bytes[i,j] \* Math.Pow(2, j));

}

string result = "";

for (int i = 0; i < decimal\_code.Length; i++)

result += alf[decimal\_code[i]];

return result;

}

**Алгоритм зашифрования:**

private void f13\_ButtonEncrypt\_Click(object sender, EventArgs e)

{

string Source = f13\_fieldOriginal.Text;

string entered\_key = f13\_fieldKeyword.Text;

//Для удаления запрещенных символов составим строку из символов текста и ключа, отсутствующих в алфавите

//Символ для теста - ✆

string prohibited\_symbs = "";

for (int i = Source.Length - 1; i >= 0; i--)

if (alf.IndexOf(Source[i]) < 0)

prohibited\_symbs += Source[i];

for (int i = entered\_key.Length - 1; i >= 0; i--)

if (alf.IndexOf(entered\_key[i]) < 0)

prohibited\_symbs += entered\_key[i];

//Удалим все встретившиеся запрещенные символы

for (int i = 0; i < prohibited\_symbs.Length; i++)

{

Source = Regex.Replace(Source, Convert.ToString(prohibited\_symbs[i]), "");

entered\_key = Regex.Replace(entered\_key, Convert.ToString(prohibited\_symbs[i]), "");

}

//Убедимся, что размер ключа не меньше длины исходного текста

if (entered\_key.Length < Source.Length)

{

MessageBox.Show("Слишком короткий ключ", "Ошибка");

goto metka\_exit;

}

//Отразим очищенные от запрещенных символов данные в полях

f13\_fieldOriginal.Text = Source;

f13\_fieldKeyword.Text = entered\_key;

//Переведем данные в последовательность битов

byte[,] mes\_bin = TextToBits(Source);

byte[,] key\_bin = TextToBits(entered\_key);

//Соберем битовые строки просто посмотреть на них

string key\_bin\_str = "";

string mes\_bin\_str = "";

for (int i = 0; i < key\_bin.Length / bitsForOneSymbol; i++)

for (int j = bitsForOneSymbol - 1; j >= 0; j--)

{

key\_bin\_str += Convert.ToString(key\_bin[i, j]);

}

for (int i = 0; i < mes\_bin.Length / bitsForOneSymbol; i++)

for (int j = bitsForOneSymbol - 1; j >= 0; j--)

{

mes\_bin\_str += Convert.ToString(mes\_bin[i, j]);

}

//Побитовое сложение по модулю 2

byte[,] result = new byte[mes\_bin.Length / bitsForOneSymbol, bitsForOneSymbol];

if (isFromFileOrig) // Блоком

{

int i = 0;

for (int k = 0; k < (mes\_bin.Length / bitsForOneSymbol) / 64; k++) //Каждый блок

for (i = k \* 64; (i < mes\_bin.Length / bitsForOneSymbol) && (i < (k + 1) \* 64); i++) // Каждую букву в пределах блока

for (int j = bitsForOneSymbol - 1; j >= 0; j--) //Каждый бит

result[i, j] = (byte)((mes\_bin[i, j] + key\_bin[i, j]) % 2);

// Если есть последний неполный блок - шифруем хвост

for (int letter = i; letter < mes\_bin.Length / bitsForOneSymbol; letter++) // Каждую букву

for (int j = bitsForOneSymbol - 1; j >= 0; j--) // Каждый бит

result[letter, j] = (byte)((mes\_bin[letter, j] + key\_bin[letter, j]) % 2);

}

else // Потоком

{

for (int i = 0; i < mes\_bin.Length / bitsForOneSymbol; i++) // Каждую букву

for (int j = bitsForOneSymbol - 1; j >= 0; j--) // Каждый бит

result[i, j] = (byte)((mes\_bin[i, j] + key\_bin[i, j]) % 2);

}

f13\_fieldCiphertext.Text = BitsToText(result);

isFromFileCipher = false;

metka\_exit:;

}

**Алгоритм расшифрования:**

private void f13\_ButtonDecrypt\_Click(object sender, EventArgs e)

{

string Source = f13\_fieldCiphertext.Text;

string entered\_key = f13\_fieldKeyword.Text;

//Для удаления запрещенных символов составим строку из символов текста и ключа, отсутствующих в алфавите

//Символ для теста - ✆

string prohibited\_symbs = "";

for (int i = Source.Length - 1; i >= 0; i--)

if (alf.IndexOf(Source[i]) < 0)

prohibited\_symbs += Source[i];

for (int i = entered\_key.Length - 1; i >= 0; i--)

if (alf.IndexOf(entered\_key[i]) < 0)

prohibited\_symbs += entered\_key[i];

//Удалим все встретившиеся запрещенные символы

for (int i = 0; i < prohibited\_symbs.Length; i++)

{

Source = Regex.Replace(Source, Convert.ToString(prohibited\_symbs[i]), "");

entered\_key = Regex.Replace(entered\_key, Convert.ToString(prohibited\_symbs[i]), "");

}

//Убедимся, что размер ключа не меньше длины исходного текста

if (entered\_key.Length < Source.Length)

{

MessageBox.Show("Слишком короткий ключ", "Ошибка");

goto metka\_exit;

}

//Отразим очищенные от запрещенных символов данные в полях

f13\_fieldCiphertext.Text = Source;

f13\_fieldKeyword.Text = entered\_key;

//Переведем данные в последовательность битов

byte[,] mes\_bin = TextToBits(Source);

byte[,] key\_bin = TextToBits(entered\_key);

//Соберем битовые строки просто посмотреть на них

string key\_bin\_str = "";

string mes\_bin\_str = "";

for (int i = 0; i < key\_bin.Length / bitsForOneSymbol; i++)

for (int j = bitsForOneSymbol - 1; j >= 0; j--)

{

key\_bin\_str += Convert.ToString(key\_bin[i, j]);

}

for (int i = 0; i < mes\_bin.Length / bitsForOneSymbol; i++)

for (int j = bitsForOneSymbol - 1; j >= 0; j--)

{

mes\_bin\_str += Convert.ToString(mes\_bin[i, j]);

}

//Побитовое сложение по модулю 2

byte[,] result = new byte[mes\_bin.Length / bitsForOneSymbol, bitsForOneSymbol];

if (isFromFileCipher) // Блоком

{

int i = 0;

for (int k = 0; k < (mes\_bin.Length / bitsForOneSymbol) / 64; k++) //Каждый блок

for (i = k \* 64; (i < mes\_bin.Length / bitsForOneSymbol) && (i < (k + 1) \* 64); i++) // Каждую букву в пределах блока

for (int j = bitsForOneSymbol - 1; j >= 0; j--) //Каждый бит

result[i, j] = (byte)(((mes\_bin[i, j] - key\_bin[i, j]) + 2) % 2);

// Если есть последний неполный блок - шифруем хвост

for (int letter = i; letter < mes\_bin.Length / bitsForOneSymbol; letter++) // Каждую букву

for (int j = bitsForOneSymbol - 1; j >= 0; j--) // Каждый бит

result[letter, j] = (byte)(((mes\_bin[letter, j] - key\_bin[letter, j]) + 2) % 2);

}

else // Потоком

{

for (int i = 0; i < mes\_bin.Length / bitsForOneSymbol; i++) // Каждую букву

for (int j = bitsForOneSymbol - 1; j >= 0; j--) // Каждый бит

result[i, j] = (byte)(((mes\_bin[i, j] - key\_bin[i, j]) + 2) % 2);

}

f13\_fieldOriginal.Text = BitsToText(result);

metka\_exit:;

}

**Работа с файлами:**

private void f13\_ButtonReadOriginal\_Click(object sender, EventArgs e)

{

string textFromFile = "";

Stream myStream = null;

OpenFileDialog myDialog = new OpenFileDialog();

myDialog.InitialDirectory = "D:\\Учеба\\Крипта\\Labs\\Labs\\bin\\Debug\\TextFiles";

myDialog.DefaultExt = "txt";

myDialog.FileName = "OriginalIn.txt";

myDialog.Filter = "txt files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*";

myDialog.FilterIndex = 2;

myDialog.RestoreDirectory = true;

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

{

try

{

if ((myStream = myDialog.OpenFile()) != null)

{

textFromFile = new StreamReader(myStream, Encoding.GetEncoding(1251)).ReadToEnd();

}

}

catch (Exception ex)

{

MessageBox.Show("Error: Could not read file from disk. Original error: " + ex.Message);

}

}

f13\_fieldOriginal.Text = textFromFile;

isFromFileOrig = true;

}

private void f13\_ButtonSaveOriginal\_Click(object sender, EventArgs e)

{

StreamWriter myStream = null;

SaveFileDialog myDialog = new SaveFileDialog();

myDialog.InitialDirectory = "D:\\Учеба\\Крипта\\Labs\\Labs\\bin\\Debug\\TextFiles";

myDialog.DefaultExt = "txt";

myDialog.FileName = "OriginalOut.txt";

myDialog.Filter = "txt files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*";

myDialog.FilterIndex = 2;

myDialog.RestoreDirectory = true;

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

{

if ((myStream = new StreamWriter(myDialog.OpenFile(), Encoding.GetEncoding(1251))) != null)

{

myStream.WriteLine(Convert.ToString(f13\_fieldOriginal.Text));

myStream.Close();

}

}

}

**Результат работы программы:**

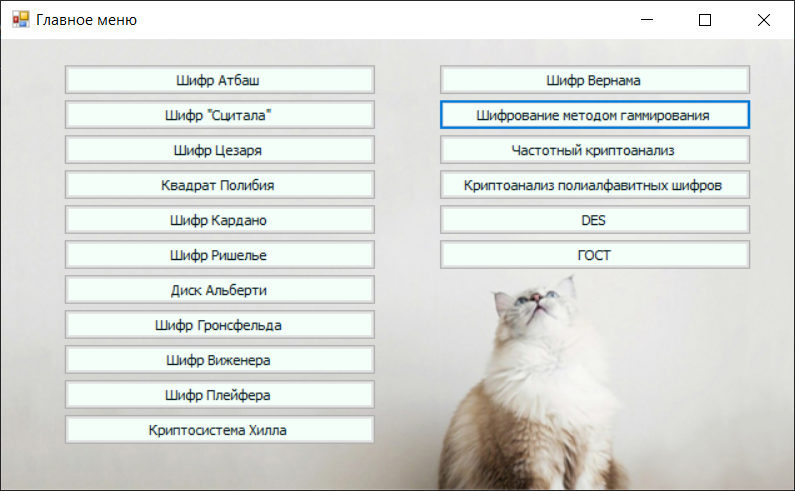


Рисунок 1. Главное меню программы.

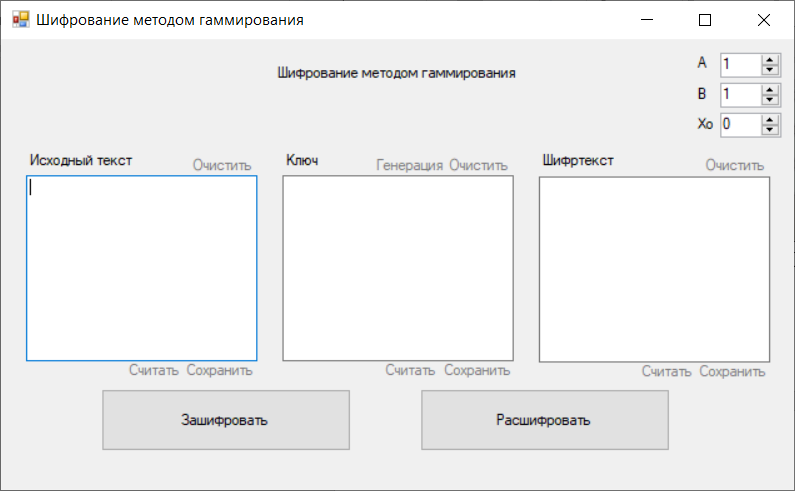


Рисунок 2. Окно шифрования методом гаммирования.

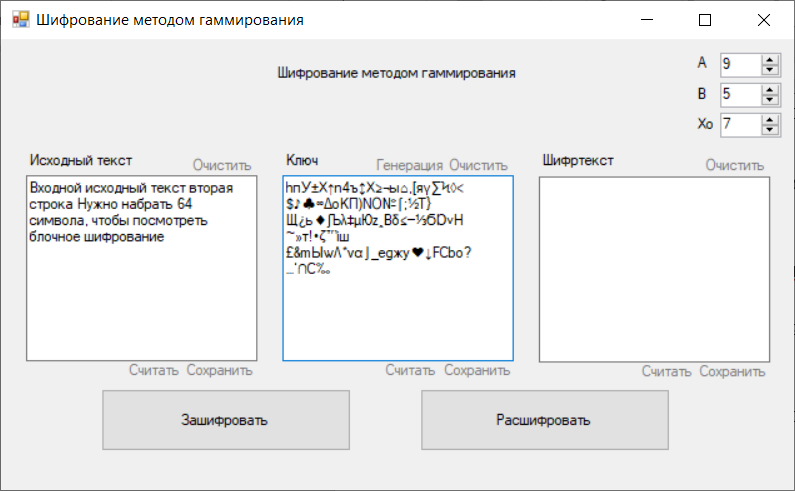


Рисунок 3. Исходный текст.

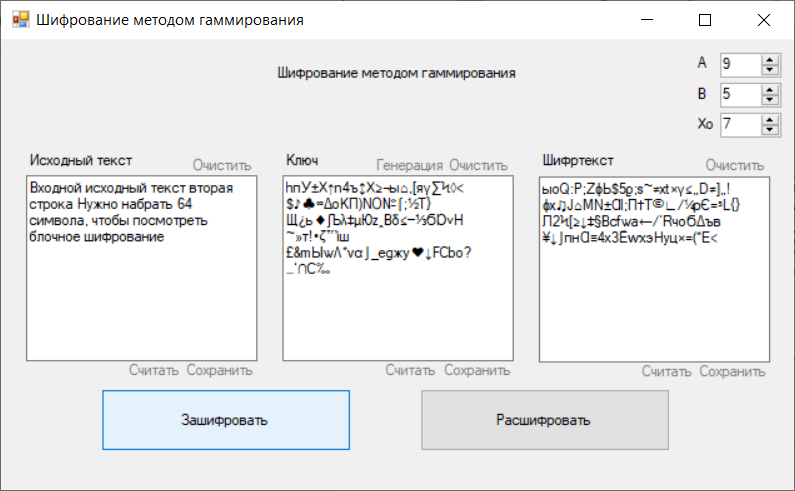


Рисунок 4. Шифртекст (результат зашифрования).

Очистим поле Исходный текст и расшифруем сообщение. Ключ необходимо оставить неизменным.

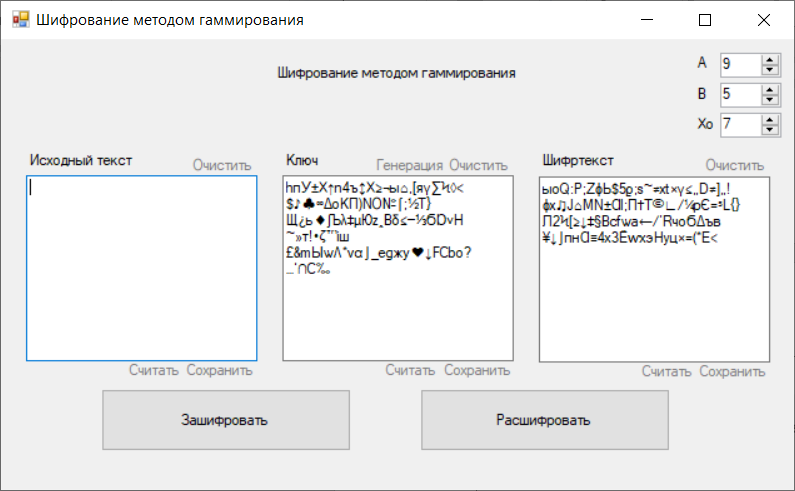


Рисунок 5. Очистка поля Исходный текст.

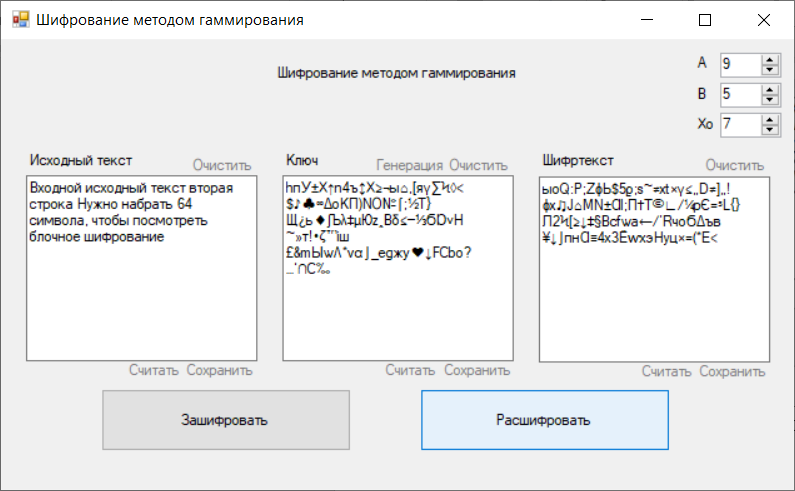


Рисунок 6. Исходный текст (результат расшифрования).

**Вывод:**

В ходе выполнения лабораторной работы я ознакомилась с алгоритмом шифрования методом гаммирования, 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 методом гаммирования.