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 №11**

|  |
| --- |
| **Криптосистема Хилла** |
| *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 криптосистему Хилла на любом языке программирования.

**Теория:**

### Шифрование

Для того, чтобы зашифровать какой-либо текст по алгоритму Хилла необходимо проделать следующие шаги:

1. Создаем кодированный алфавит. Допустим мы хотим шифровать русский текст. Тогда длина алфавита будет 33 буквы. Целесообразно добавить к алфавиту еще 4 символа на выбор, я добавлю такие: "?", ".", ","," ". Это делается для того, чтобы длина алфавита была простым числом, т.е. числом, которое делится нацело только на себя и на 1. Это, конечно, не обязательно, но очень удобно, потому что для расшифровки необходимо, чтобы детерминант ключа и длина алфавита были взаимно простыми, т.е. не имели общих делителей кроме 1. Если длина алфавита – простое число, то таких ключей, для которых выполняется это условие значительно больше. Каждому символу нашего алфавита ставим в соответствие целочисленный код. Удобнее всего использовать просто номера букв. Таким образом получаем кодированный алфавит (справа).
2. Теперь берем текст, который хотим зашифровать и кодируем его с помощью нашего алфавита. Возьмем для примера слово **«ШИФР»**, его код будет таким: **25 9 21 17**.
3. Теперь выбираем ключевое слово, или просто набор букв, который будем использовать в качестве ключа. Тут важно, чтобы длина этого ключевого слова была равна квадрату целого числа, т.е. **4, 9, 16, 25** и т.д. Только тогда мы сможем сделать из него квадратную матрицу, необходимую для шифрования. Я выбрал слово **«АЛЬПИНИЗМ»**. Кодируем его с помощью нашего алфавита. Получаем: **0 12 29 16 9 14 9 8 13**. Запишем ключ в виде матрицы 3х3:

https://habrastorage.org/web/2a3/e6f/65c/2a3e6f65c17144528edbcad30720de91.PNG  
  
Ключ можно задавать сразу матрицей.

1. Теперь надо разбить текст на блоки по n символов в каждом, где n-размерность матрицы, в моем случае – 3. Начнем разбивать:

Первый блок: **(25 9 21)**

На второй блок у нас осталось всего одно число – 17. Самое простое решение в таком случае: добавить столько символов, чтобы образовать целый блок.

Тогда второй блок: **(17 35 35)**

1. Для шифрования текста требуется провести матричное умножение каждого блока на матрицу ключа. Тут стоит заметить, что блоки можно было бы записывать не в строки, а в столбцы. Тогда мы бы умножали ключ на столбец, это не существенное различие.

Также важным фактором для данного шифра является определитель матрицы ключа: он должен быть отличным от нуля, иначе расшифровку зашифрованного текста будет невозможно осуществить.

Итак, умножаем первый блок на ключ:

https://habrastorage.org/web/5e4/f48/379/5e4f483794104a2c9f4499ea84f3167c.PNG  
  
Умножаем второй блок на ключ:

https://habrastorage.org/web/809/870/49c/80987049cc7340e68c9acec8d18fe1af.PNG  
  
Теперь нам нужно получившиеся матрицы разделить по модулю на 37, т.е. взять остаток от деления на 37.

Делим первую матрицу:

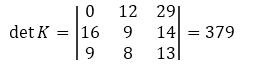
https://habrastorage.org/web/267/fa5/095/267fa5095dae47a292e9fec3cc93f7f0.PNG  
  
Делим вторую матрицу:

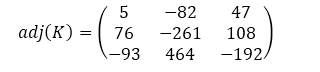
https://habrastorage.org/web/536/c95/75a/536c9575a0164813bb1a43c1dc2c85d7.PNG  
  
Почему делим на 37? Потому что это длина нашего алфавита, будь у вас алфавит другой длины, вы бы делили на другое число. Например, для английского алфавита делим на 26, или 29, если вы добавили какие-то символы.

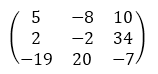
1. Теперь декодируем полученные матрицы с помощью нашего алфавита.  
     
   Первая матрица: АЮН  
   Вторая матрица: ЧХЯ  
     
   Склеиваем две матрицы и получаем зашифрованный текст: АЮНЧХЯ

### Расшифрование

Теперь переходим к расшифрованию. Расшифрование производим по следующему алгоритму:

1. Обратно кодируем шифртекст в цифры и разбиваем на блоки.
2. Находим определитель матрицы ключа:  
     
   
3. Теперь по расширенному алгоритму Евклида находим **d**, **x**, **y**.
4. Найдем **обратный детерминанту элемент в кольце по модулю 37**.   
     
   Итак, наш детерминант равен **379**, он положительный, а **x** равен **-4** – отрицательный. Тогда обратный детерминанту элемент находим по формуле **37+x=37+(-4)=37-4=33**.
5. Надо найти **матрицу обратную матрице ключа по модулю 37**. Для того чтобы найти эту матрицу нам необходимо найти матрицу алгебраических дополнений ключа и обратный детерминант матрицы ключа (уже нашли в предыдущем пункте). В нашем случае матрица алгебраических дополнений ключа выглядит так:

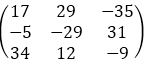
  
Теперь эту матрицу делим по модулю на **37**. Получаем такую матрицу, тут важно не терять знаки у элементов (некоторые выполняют деление по модулю с потерей минусов, в данном алгоритме это недопустимо):

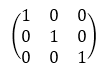
  
  
Умножаем матрицу алгебраических дополнений на обратный детерминанту элемент. Получаем такую матрицу:

https://habrastorage.org/web/997/a65/2f9/997a652f9f6642f3a485f85971f690b7.PNG  
  
Делим данную матрицу по модулю на **37**:

https://habrastorage.org/web/2ba/1e3/2ee/2ba1e32ee75a4e1c823b998a25101eb6.PNG

Транспонируем ее (меняем строки и столбцы местами):

  
  
Теперь если элемент матрицы отрицательный, меняем его на другой, вычисленный по формуле **37+<элемент>**:  
  
https://habrastorage.org/web/ef3/662/ced/ef3662ced77243e0bd0e8562a7cd4d93.PNG  
  
Последняя полученная матрица является обратной по модулю к матрице ключа. Если перемножить матрицу ключа и эту матрицу, а потом результат разделить по модулю на 37, мы получим единичную матрицу, т.е. матрицу вида:



1. Для расшифровки шифртекста умножаем строки шифртекста на матрицу обратную ключу.  
   Умножаем первую строку:

https://habrastorage.org/web/9c9/e44/134/9c9e441347784c6c80e341b2c58b2f71.PNG  
  
Умножаем вторую строку:

https://habrastorage.org/web/a7f/824/3bb/a7f8243bbddf4b64b6ccb1b6137c046b.PNG  
  
Делим полученные строки на 37 по модулю:

https://habrastorage.org/web/85f/bd9/0f9/85fbd90f91f941c796e10693abd54bea.PNG  
  
Склеиваем матрицы **(25 9 21 13 35 35)** и декодируем с помощью нашего алфавита: ШИФР.  
  
В итоге мы получили исходный текст с двумя лишними пробелами в конце, которые никакой роли не играют.

**Исходный код:**

**Используемый алфавит:**

public string alf\_ru = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя.,! "; //Length = 33+4=37

public string ALF\_RU = "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ:;?\*"; //Length = 33+4=37

public string alf\_en = "abcdefghijklmnopqrstuvwxyz.,!' "; //Length = 26+5=31

public string ALF\_EN = "ABCDEFGHIJKLMNOPQRSTUVWXYZ:;?^\*"; //Length = 26+5=31

**Класс для работы с матрицами:**

public class Matrix

{

static Random rnd = new Random();

public Decimal[,] array;

public int row, column;

public int Row { get { return row; } }

public int Column { get { return column; } }

public void Random()

{

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

{

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

{

array[i, j] = rnd.Next(10);

}

}

}

public void Random(int min, int max)

{

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

{

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

{

array[i, j] = rnd.Next(min, max);

}

}

}

public Matrix(int row, int colunm)

{

this.row = row;

this.column = colunm;

array = new Decimal[row, column];

}

public Matrix Transpose()

{

Matrix m = new Matrix(column, row);

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

{

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

{

m.array[j, i] = array[i, j];

}

}

return m;

}

public void TransposeMyself()

{

array = Transpose().array;

}

public Matrix Inverse()

{

Decimal det = Determinant();

if (det == 0)

{

throw new Exception("Матрица вырождена");

}

Matrix m = new Matrix(row, column);

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

{

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

{

m.array[i, j] = Cofactor(array, i, j) / det;

}

}

return m.Transpose();

}

public Matrix Inverse(int finiteFieldSize)

{

Decimal det = Determinant();

if (det == 0)

{

throw new Exception("Матрица вырождена");

}

Matrix m = new Matrix(row, column);

int antiDet = Foo(Convert.ToInt32(det), finiteFieldSize); //Элемент, обратный к определителю

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

{

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

{

m.array[i, j] = (((Cofactor(array, i, j)) % finiteFieldSize \* antiDet) % finiteFieldSize + finiteFieldSize) % finiteFieldSize;

}

}

return m.Transpose();

}

public Decimal Determinant()

{

if (column != row)

{

throw new Exception("Расчет определителя невозможен");

}

return Determinant(array);

}

private Decimal Determinant(Decimal[,] array)

{

int n = (int)Math.Sqrt(array.Length);

if (n == 1)

{

return array[0, 0];

}

Decimal det = 0;

for (int k = 0; k < n; k++)

{

det += array[0, k] \* Cofactor(array, 0, k);

}

return det;

}

private Decimal Cofactor(Decimal[,] array, int row, int column)

{

return Convert.ToDecimal(Math.Pow(-1, column + row)) \* Determinant(Minor(array, row, column));

}

private Decimal[,] Minor(Decimal[,] array, int row, int column)

{

int n = (int)Math.Sqrt(array.Length);

Decimal[,] minor = new Decimal[n - 1, n - 1];

int \_i = 0;

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

{

if (i == row)

{

continue;

}

int \_j = 0;

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

{

if (j == column)

{

continue;

}

minor[\_i, \_j] = array[i, j];

\_j++;

}

\_i++;

}

return minor;

}

public static Matrix operator +(Matrix m1, Matrix m2)

{

if (m1.row != m2.row || m1.column != m2.column)

{

throw new Exception("Сложение невозможно");

}

Matrix m = new Matrix(m1.row, m1.column);

for (int i = 0; i < m1.row; i++)

{

for (int j = 0; j < m1.column; j++)

{

m.array[i, j] = m1.array[i, j] + m2.array[i, j];

}

}

return m;

}

public static Matrix operator -(Matrix m1, Matrix m2)

{

if (m1.row != m2.row || m1.column != m2.column)

{

throw new Exception("Вычитание невозможно");

}

Matrix m = new Matrix(m1.row, m1.column);

for (int i = 0; i < m1.row; i++)

{

for (int j = 0; j < m1.column; j++)

{

m.array[i, j] = m1.array[i, j] - m2.array[i, j];

}

}

return m;

}

public static Matrix operator \*(Matrix m1, Matrix m2)

{

if (m1.column != m2.row)

{

throw new Exception("Умножение невозможно");

}

Matrix m = new Matrix(m1.row, m2.column);

for (int i = 0; i < m1.row; i++)

{

for (int j = 0; j < m2.column; j++)

{

decimal sum = 0;

for (int k = 0; k < m1.column; k++)

{

sum += m1.array[i, k] \* m2.array[k, j];

}

m.array[i, j] = sum;

}

}

return m;

}

public override string ToString()

{

string str = "";

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

{

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

{

str += array[i, j] + "\t";

}

str += "\n";

}

return str;

}

}

**Функция поиска элемента, обратного данному, в поле**

public static int Foo(int element, int fSize)

{

int x, y;

int g = GCD(element, fSize, out x, out y);

if (g != 1)

if (g == -1)

{

g = 1;

x = -x;

y = -y;

}

return (x % fSize + fSize) % fSize;

} //поиск обратного элемента в поле

**Функция нахождения НОД:**

public static int GCD(int a, int b, out int x, out int y)

{

if (a == 0)

{

x = 0;

y = 1;

return b;

}

int x1, y1;

int d = GCD(b % a, a, out x1, out y1);

x = y1 - (b / a) \* x1;

y = x1;

return d;

}

**Алгоритм зашифрования:**

private void f11\_ButtonEncrypt\_Click(object sender, EventArgs e)

{

//Подготовка сообщения и ключа

string Source = f11\_fieldOriginal.Text;

string entered\_key = f11\_fieldKeyword.Text;

string alf, ALF;

if (f11\_radioButtonRus.Checked)

{

Source = Regex.Replace(Source, "[^а-яА-ЯёЁ .,!:;?\*]", "");

entered\_key = Regex.Replace(entered\_key, "[^а-яА-ЯёЁ .,!:;?\*]", "");

alf = alf\_ru;

ALF = ALF\_RU;

}

else

{

Source = Regex.Replace(Source, "[^a-zA-Z .,!':;?\*^]", "");

entered\_key = Regex.Replace(entered\_key, "[^a-zA-Z .,!':;?\*^]", "");

alf = alf\_en;

ALF = ALF\_EN;

}

//Проверка длины ключа на квадратность

if (!isSquare(entered\_key.Length))

{

MessageBox.Show("Некорректный ключ. Ожидаемая длина ключа - полный квадрат>1.", "Ошибка");

goto metka\_exit;

}

int matrixSize = Convert.ToInt32(Math.Sqrt(entered\_key.Length));

//Приведение сообщения и ключа к корректному виду

while (Source.Length % matrixSize != 0) Source = Source + " ";

f11\_fieldOriginal.Text = Source;

f11\_fieldKeyword.Text = entered\_key;

entered\_key = entered\_key.ToLower().Replace(":", ".").Replace(";", ",").Replace("?", "!")

.Replace("\*", " ").Replace("^", "'");

//Сохранение регистра сообщения

bool[] isUpper = new bool[Source.Length];

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

if (alf.IndexOf(Source[i]) >= 0)

isUpper[i] = false;

else

isUpper[i] = true;

Source = Source.ToLower().Replace(":", ".").Replace(";",",").Replace("?","!")

.Replace("\*"," ").Replace("^","'");

//Заполнение матрицы-ключа

Matrix keyMatrix = new Matrix(matrixSize, matrixSize);

for (int i = 0; i < keyMatrix.row; i++)

{

for (int j = 0; j < keyMatrix.column; j++)

{

keyMatrix.array[i, j] = alf.IndexOf(entered\_key[i \* keyMatrix.column + j]);

}

}

//Проверка матрицы-ключа на обратимость

if (keyMatrix.Determinant() == 0)

{

MessageBox.Show("Некорректный ключ. Матрица не обратима. Расшифрование невозможно.", "Ошибка");

goto metka\_exit;

}

//Зашифрование

string result = "";

int finiteFieldSize = alf.Length;

int amount\_of\_crypted\_symbs = 0;

while (amount\_of\_crypted\_symbs < Source.Length)

{

Matrix message = new Matrix(1, matrixSize);

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

message.array[0,i] = alf.IndexOf(Source[i + amount\_of\_crypted\_symbs]);

Matrix res = message \* keyMatrix;

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

if (isUpper[i + amount\_of\_crypted\_symbs])

result += ALF[Convert.ToInt32((res.array[0, i]) % finiteFieldSize)];

else

result += alf[Convert.ToInt32((res.array[0,i]) % finiteFieldSize)];

amount\_of\_crypted\_symbs += matrixSize;

}

f11\_fieldCiphertext.Text = result;

metka\_exit:;

}

**Алгоритм расшифрования:**

private void f11\_ButtonDecrypt\_Click(object sender, EventArgs e)

{

//Подготовка сообщения и ключа

string Source = f11\_fieldCiphertext.Text;

string entered\_key = f11\_fieldKeyword.Text;

string alf, ALF;

if (f11\_radioButtonRus.Checked)

{

Source = Regex.Replace(Source, "[^а-яА-ЯёЁ .,!\*:;?]", "");

entered\_key = Regex.Replace(entered\_key, "[^а-яА-ЯёЁ .,!\*:;?]", "");

alf = alf\_ru;

ALF = ALF\_RU;

}

else

{

Source = Regex.Replace(Source, "[^a-zA-Z .,!'\*:;?^]", "");

entered\_key = Regex.Replace(entered\_key, "[^a-zA-Z .,!'\*:;?^]", "");

alf = alf\_en;

ALF = ALF\_EN;

}

//Проверка длины ключа на квадратность

if (!isSquare(entered\_key.Length) || entered\_key.Length < 4)

{

MessageBox.Show("Некорректный ключ. Ожидаемая длина ключа - полный квадрат>1", "Ошибка");

goto metka\_exit;

}

int matrixSize = Convert.ToInt32(Math.Sqrt(entered\_key.Length));

//Проверка шифртекста на соответствие длины

if (Source.Length % matrixSize != 0)

{

MessageBox.Show("Некорректный шифртекст. Ожидаемая длина шифртекста кратна корню из длины ключа", "Ошибка");

goto metka\_exit;

}

//Приведение сообщения и ключа к корректному виду

f11\_fieldCiphertext.Text = Source;

f11\_fieldKeyword.Text = entered\_key;

entered\_key = entered\_key.ToLower().Replace(":", ".").Replace(";", ",").Replace("?", "!")

.Replace("\*", " ").Replace("^", "'");

//Сохранение регистра сообщения

bool[] isUpper = new bool[Source.Length];

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

if (alf.IndexOf(Source[i]) >= 0)

isUpper[i] = false;

else

isUpper[i] = true;

Source = Source.ToLower().Replace(":", ".").Replace(";", ",").Replace("?", "!")

.Replace("\*", " ").Replace("^", "'");

//Заполнение матрицы-ключа

Matrix keyMatrix = new Matrix(matrixSize, matrixSize);

for (int i = 0; i < keyMatrix.row; i++)

{

for (int j = 0; j < keyMatrix.column; j++)

{

keyMatrix.array[i, j] = alf.IndexOf(entered\_key[i \* keyMatrix.column + j]);

}

}

//Проверка матрицы-ключа на обратимость

if (keyMatrix.Determinant() == 0)

{

MessageBox.Show("Некорректный ключ. Матрица не обратима. Расшифрование невозможно.", "Ошибка");

goto metka\_exit;

}

//Обратная матрица

Matrix antiKeyMatrix = keyMatrix.Inverse(alf.Length);

//Расшифрование

string result = "";

int finiteFieldSize = alf.Length;

int amount\_of\_crypted\_symbs = 0;

while (amount\_of\_crypted\_symbs < Source.Length)

{

Matrix message = new Matrix(1, matrixSize);

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

message.array[0, i] = alf.IndexOf(Source[i + amount\_of\_crypted\_symbs]);

Matrix res = message \* antiKeyMatrix;

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

if (isUpper[i + amount\_of\_crypted\_symbs])

result += ALF[Convert.ToInt32((res.array[0, i]) % finiteFieldSize)];

else

result += alf[Convert.ToInt32((res.array[0, i]) % finiteFieldSize)];

amount\_of\_crypted\_symbs += matrixSize;

}

f11\_fieldOriginal.Text = result;

metka\_exit:;

}

**Результат работы программы:**

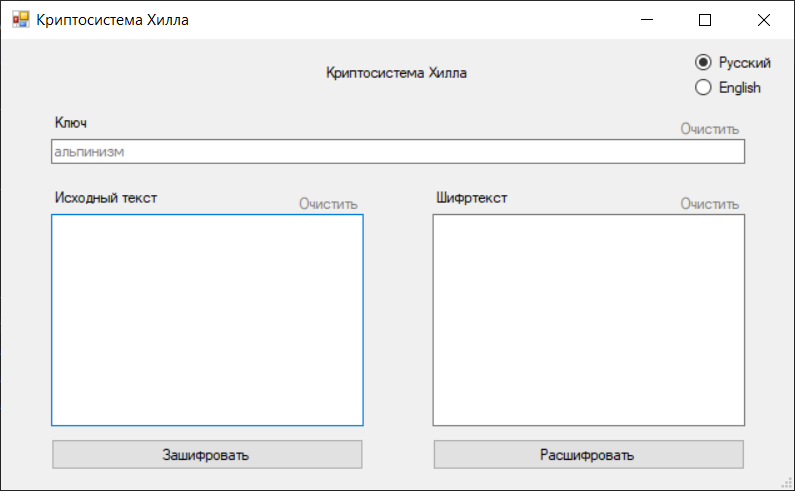


Рисунок 1. Главное меню программы.

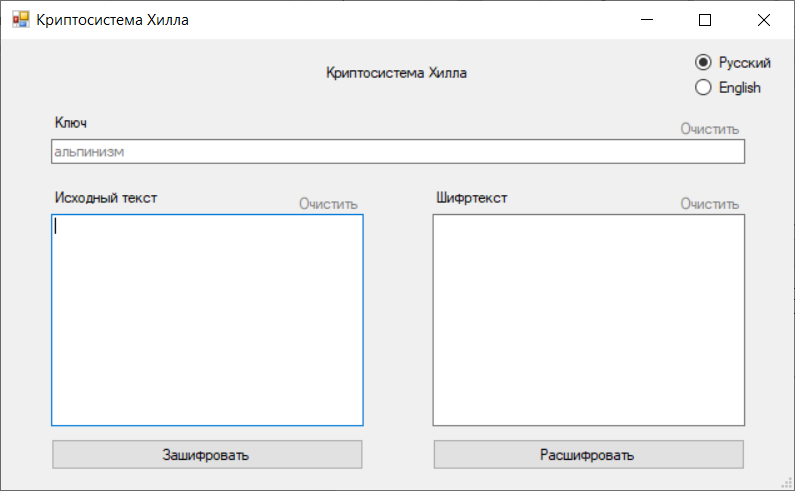


Рисунок 2. Окно криптосистемы Хилла.

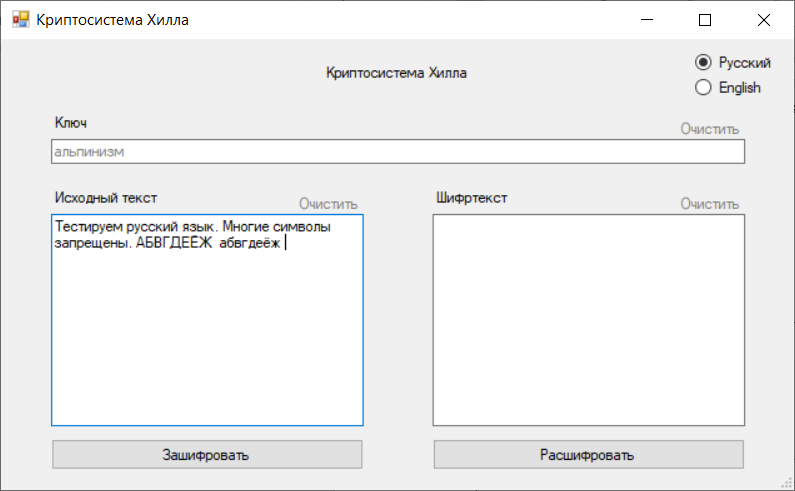


Рисунок 3. Исходный текст.

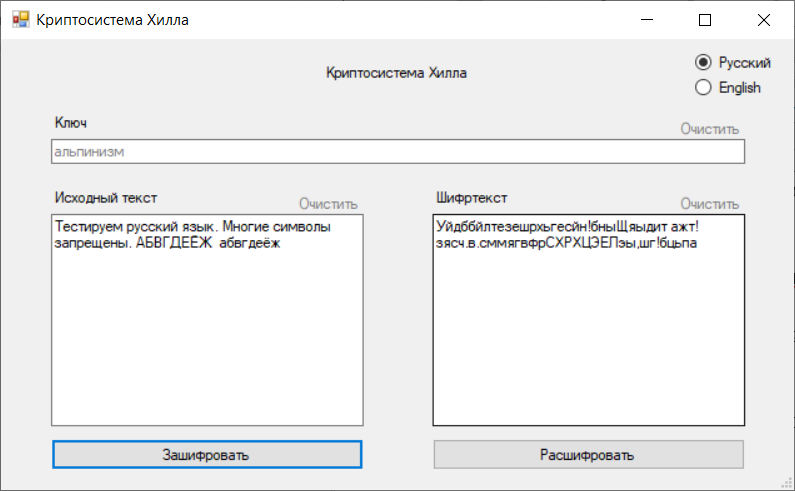


Рисунок 4. Шифртекст (результат зашифрования).

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

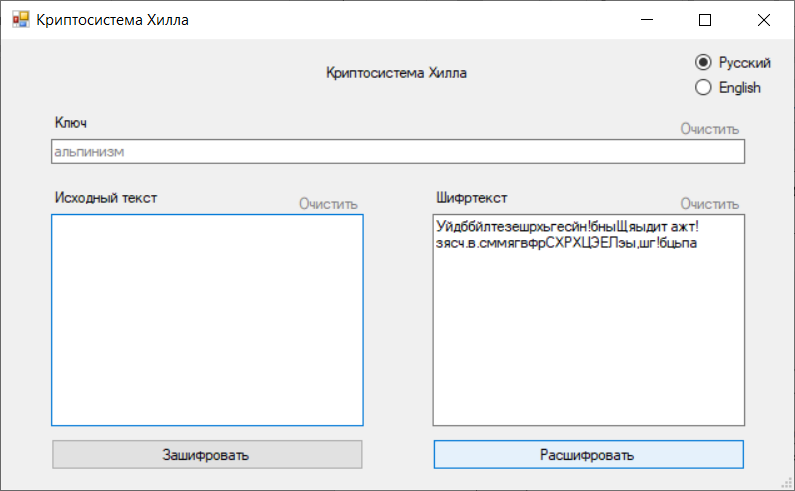


Рисунок 5. Очистка поля Исходный текст.

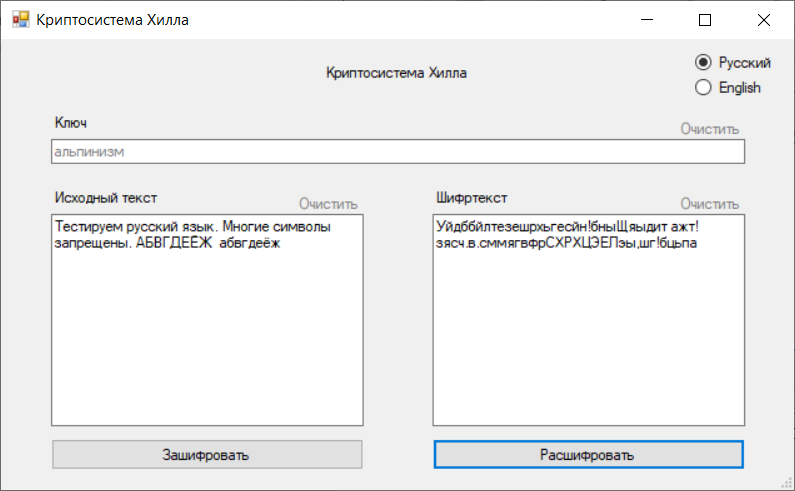


Рисунок 6. Исходный текст (результат расшифрования).

**Вывод:**

В ходе выполнения лабораторной работы я ознакомилась с алгоритмом работы криптосистемы Хилла, 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 криптосистемы Хилла.