Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

«Исследование асимметричных шифров

RSA и Эль-Гамаля»

Студент: Белицкий В.Д.

ФИТ 3 курс 5 группа

Преподаватель:

Савельева Маргарита Геннадьевна

Минск 2023

1. **Описание приложения**

Приложение написано на языке программирования C# и позволяет выполнить следующие задачи:

* генерация ключевой информации;
* зашифрование текстовых документов на основе алгоритма RSA;
* расшифрование на основе алгоритма RSA;
* зашифрование на основе алгоритма Эль-Гамаля;
* расшифрование на основе алгоритма Эль-Гамаля;
* оценка времени выполнения операций зашифрования и расшифрования.

1. **Методика выполнения поставленных задач**

**2.1. Алгоритм Эль-Гамаля**

Для реализации генерации ключевой информации были выполнены следующие шаги:

В первом этапе пользователь выбирает простое число *p* и параметр *x*, который находится в диапазоне [1; *p*-1].

Затем приложение автоматически выбирает число *g*, которое должно быть меньше *p* и являться первообразным по модулю *p*. Это означает, что степени числа *g* дают все возможные остатки по модулю *p*, которые являются взаимно простыми с *p*.

После этого производится вычисление открытого ключа y по формуле: *y = mod* *p*. Теперь у нас есть полный набор ключевой информации *(p, g, x, y*), необходимой для выполнения операций шифрования и расшифрования.

Для шифрования каждого отдельного блока исходного сообщения (где в нашем случае 1 блок равен 1 символу), используется случайное число *k,* которое должно находиться в диапазоне от 1 до .

На рисунке 2.1 изображена функция зашифрования.

public static string Encrypt\_El(int p, int g, int x, string inString)

{

var result = "";

var y = Power(g, x, p);

var rand = new Random();

Console.WriteLine($"Открытый ключ (p,g,y)=({p},{g},{y})");

Console.WriteLine($"Закрытый ключ x={x}");

Console.WriteLine("\nИсходный текст: ");

foreach (int code in inString)

if (code > 0)

{

Console.Write((char)code);

var k = rand.Next() % (p - 2) + 1;

var a = Power(g, k, p);

var b = Mul(Power(y, k, p), code, p);

result += a + " " + b + " ";

}

return result;

Рис. 2.1 – Реализация зашифрования

Для дешифрования полученного шифротекста мы вычисляем каждый блок исходного сообщения с использованием формулы: *m = (b × ())* *mod p*.

Процесс расшифрования реализован в соответствии с представленным на рисунке 2.2 алгоритмом.

public static string Decrypt\_El(int p, int x, string inText)

{

var result = "";

var arr = inText.Split(' ').Where(xx => xx != "").ToArray();

for (var i = 0; i < arr.Length; i += 2)

{

var a = int.Parse(arr[i]);

var b = int.Parse(arr[i + 1]);

if (a != 0 && b != 0)

{

var deM = Mul(b, Power(a, p - 1 - x, p), p);

var m = (char)deM;

result += m;

}

}

return result;

Рис. 2.2 – Реализация расшифрования

**2.2. Алгоритм *RSA***

Затем мы рассмотрим программную реализацию алгоритма *RSA* для шифрования и расшифрования. Этот алгоритм гораздо более прост в понимании и реализации, чем предыдущий.

Действия алгоритма состоят из следующих шагов:

1. Пользователь вводит большие простые числа p и q. Желательно, чтобы они были одинаковой длины, чтобы алгоритм был более криптостойким. Если пользователь вводит число, которое не является простым, система предупреждает его об этом.
2. Выбирается число e, взаимно простое с функцией Эйлера *φ(n) = (p-1)(q-1*). Пара *(e, n)* становится открытым ключом алгоритма.
3. Вычисляется число *d* с использованием алгоритма Евклида. Пара *(d, n)* становится закрытым ключом алгоритма.

Рассмотрим алгоритм шифрования сообщения. Каждый блок шифротекста вычисляется отдельно с использованием формулы *c = () mod n*, где *m* - блок исходного сообщения. Программный код, реализующий эти вычисления, представлен на рисунке 2.3.

public static string EncryptRSA(string str, long e, long n)

{

string result = "";

BigInteger bi;

foreach (int code in str)

if (code > 0)

{

bi = new BigInteger((char)code);

bi = BigInteger.Pow(bi, (int)e);

BigInteger n\_ = new BigInteger((int)n);

bi = bi % n\_;

result += (bi.ToString() + " ");

}

return result;

Листинг. 2.1 – Реализация зашифрования

Расшифрование проводится подобным образом по формуле *m = (c)d mod n*, с использованием закрытого ключа *d*. Функция расшифрования изображена на рисунке 2.3.

public static string DecryptRSA(string str, int d, int n)

{

string result = "";

var arr = str.Split(' ').ToArray();

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

{

int b = (int)(Convert.ToDouble(arr[i]));

b = (int)Math.Pow(b, d);

b = b % n;

var m = (char)b;

result += m;

}

return result;

Рис. 2.3 – Реализация расшифрования

Результат работы первого алгоритма представлен на рисунке 2.5.

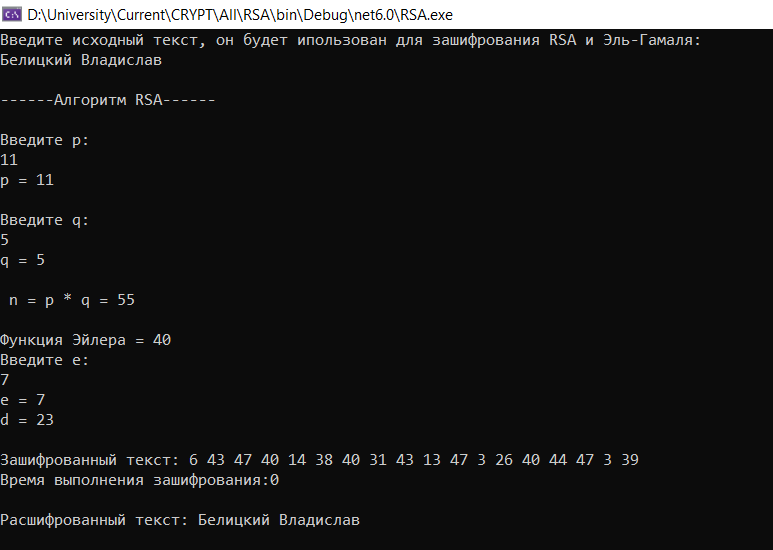


Рис 2.4 – Результат работы RSA

Результат работы первого алгоритма представлен на рисунке 2.5.

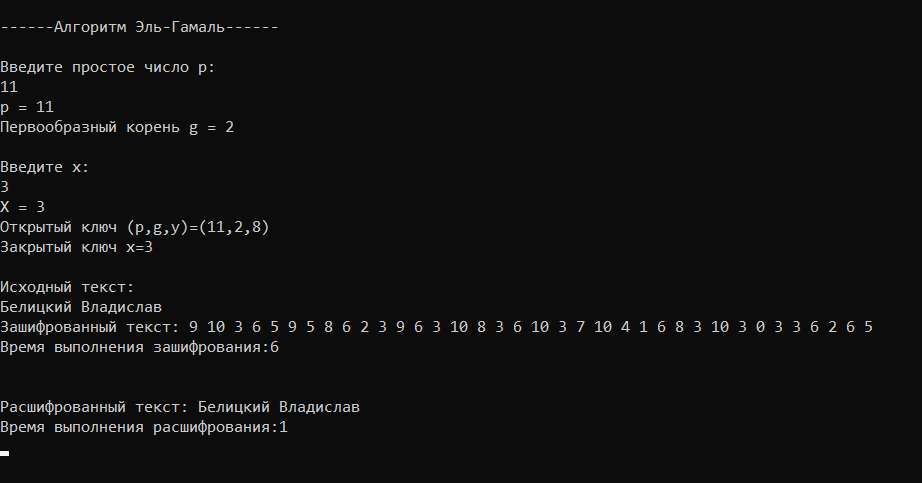


Рис 2.5 – Результат работы RSA

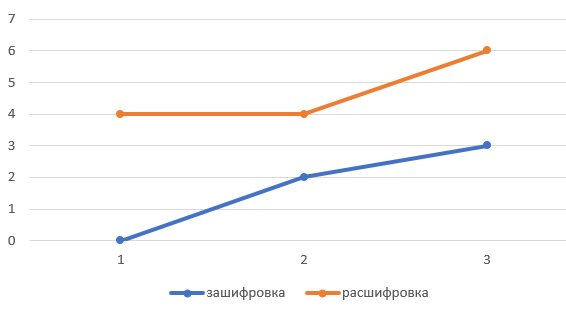


Рис 2.6 – График среднего времени работы

1. **Используя примерно одинаковый порядок ключевой информации, оценить производительность обоих алгоритмов и относительное изменение объемов криптотекстов**

Если добавить к исходному тексту дополнительные символы, например, отчество, то при использовании алгоритма *RSA* количество чисел, необходимых для шифрования, будет зависеть от количества символов. На рисунке 2.9. было подсчитано количество символов при шифровании текста «*Bialitski Vladislav*» с использованием RSA: получилось 19 чисел для 19 букв. Однако, если зашифровать текст *«Bialitski Vladislav Dmitrievich*», то получится 31 букв и 31 чисел. Таким образом, каждой букве соответствует одно число.

В случае алгоритма Эль-Гамаля, при шифровании 19 символов получается 38 чисел. Также, при шифровании 31 символов получается 64 чисел. Можно сделать вывод, что при использовании алгоритма Эль-Гамаля количество чисел, необходимых для шифрования, примерно в два раза больше, чем количество исходных символов.

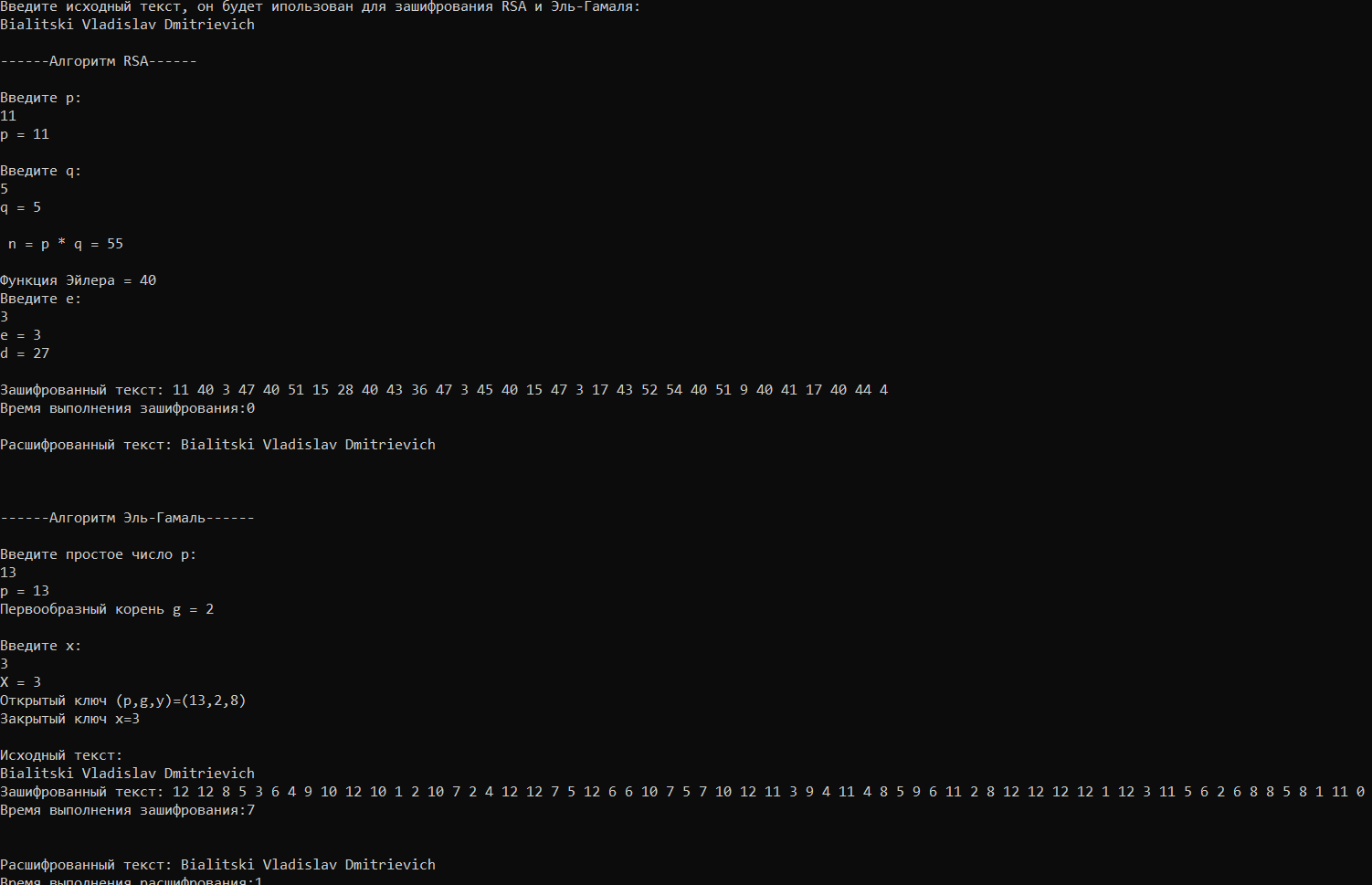
****

Рис 2.7– Результат работы приложения при увеличении текста

**Вывод**

В процессе выполнения лабораторной работы мы получили практические навыки разработки и использования приложений для реализации двух алгоритмов асимметричного шифрования - *RSA* и Эль-Гамаля. Было создано приложение, которое позволяет генерировать ключевую информацию и использовать ее для шифрования и расшифрования данных. Кроме того, мы провели оценку скорости выполнения операций шифрования и расшифрования для оценки эффективности и производительности данных алгоритмов.