Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий

институт

Кафедра «Информатика»

кафедра

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2**

Преподаватель \_\_\_\_\_\_\_\_\_\_\_ А.Н. Шниперов

подпись, дата инициалы, фамилия

Студент КИ15–17Б, 031510065 \_\_\_\_\_\_\_\_\_\_\_ Т.В. Радионов

номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2017

1. **Цели лабораторной работы**

* ознакомиться с основами ассиметричной криптографии;
* ознакомиться с элементами теории чисел, используемых в криптографии с открытым ключом;
* изучить особенности алгоритма с открытым ключом RSA;
* получить навыки разработки криптосистем с открытым ключом с
* использованием языка программирования высокого уровня.

1. **Задание лабораторной работы**

Разработать алгоритм шифрования/расшифровывания RSA (аббревиатура от фамилий Rivest, Shamir и Adleman) со следующими особенностями:

* объем исходного текста любой (в разумных пределах;
* исходный текст может состоять из русских и английских букв, цифр, а также знаков препинания;
* исходный текст находится в кодировке Unicode;
* выступающие в качестве модуля число – N, которое выбирается автоматически и состоит из 35-ти десятичных знаков. P и Q выбираются случайным образов, где P и Q – простые числа;
* исходный текст разбивается на K блоков, где K выбирается исходя из значений модуля N.

1. **Описание алгоритма шифрования**

Представляем своё сообщение в цифровом виде, используя коды символов Unicode (4-хзначное числовое представление символа) и разбиваем на блоки по 8 символов (), добавляем недостающие, не несущие никакого смысла числа, чтобы блок был равен . Соответственно, после шифрования зашифрованное сообщение будет состоять из блоков . Субъект, выполняющий шифрование, должен обладать открытым ключом .

Далее шифруем каждый блок своего сообщения по формуле .

1. **Описание алгоритма расшифрования**

Для расшифрования каждого блока и получения расшифрованного блока символов применяется формула , где – закрытый ключ. Затем расшифрованные блоки конвертируются в символы Unicode.

1. **Оценка алгоритма шифрования**

RSA – криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших целых чисел.

С точки зрения производительности условно можно выделить следующие этапы:

1. Формирование открытого и закрытого ключа
2. Процесс шифрования
3. Процесс дешифрования

На первом этапе выполняется генерация случайных чисел *P* и *Q* одинаковый длины 17 символов (наилучший вариант с точки зрения криптостойкости) так, чтобы произведение *P* на *Q* давало число, не превосходящее длиной 35 символов. Далее проверяется на простоту раздельно число *P* и число *Q* с помощью различных тестов на простоту: тест простоты (перебор делителей), тест Ферма, тест Миллера-Рабина. Если число не простое, выполняем его перегенерацию. В случае успеха на данном шаге выполняется формирование числа *N*, открытого ключа *S* и закрытого ключа *E*.

Скорость выполнения первого этапа зависит напрямую от скорости тестов на простоту значений *P* и *Q*, которые участвуют в формировании открытого и закрытого ключа.

Стоит отметить, что вероятностый тест Ферма показал наибыстрейшие результаты проверки числа на простоту по сравнению с тестами Миллера-Рабина, однако возникали и исключительные ситуации, где было наоборот (однако оба алгоритма работали менее, чем за 1 секунду). Медленнее всех оказалась стандартная проверка простоты, но именно она самая точная, так как проверяет на делимость все числа для установления простоты (скорость работы варьировалась от 30 до 60 секунд).

Скорость выполнения второго этапа зависит ненулевых битов, то есть время выполнения операций растет с увеличением количества ненулевых битов в двоичном представлении экспоненты *S*. Чтобы увеличить скорость шифрования, значение *s* часто выбирают равным 17, 257 или 65537. Также вычислительная сложность увеличивается из-за использования функции возведения в степень.

Скорость выполнения третьего этапа зависит лишь от скорости выполнения функции возведения в степень.

Стойкость алгоритма основывается на сложности вычисления обратной функции к функции шифрования {\displaystyle c=E(m)=m^{e}\mod n}.

Для вычисления {\displaystyle m} по известным {\displaystyle c,e,n} нужно найти такой  {\displaystyle d}, чтобы {\displaystyle e\cdot d\equiv 1{\pmod {\varphi (n)}},}, то есть злоумышленнику для вычисления необходимо знать значение. Для вычисления функции Эйлера от известного числа необходимо знать разложение этого числа на простые множители. Нахождение этих множителей является основой криптостойкости алгоритма шифрования RSA, так как знание этих множителей и есть «потайная дверь», которая используется для вычисления закрытого ключа .

1. **Код алгоритма**

public partial class Form1 : Form

{

static BigInteger N;

static BigInteger S;

static BigInteger E;

static Stopwatch sw = Stopwatch.StartNew();

public Form1()

{

InitializeComponent();

}

/// <summary>

/// Ф-ция генерации случайного числа типа Big Integer в диапазоне 100000000100000000 - 31000000031000000000

/// </summary>

/// <returns></returns>

private static BigInteger RandomGenerateBigInteger()

{

Random random = new Random();

BigInteger value = 0;

string valueString;

valueString = Convert.ToString(random.Next(100000000, 310000000));

valueString = valueString + Convert.ToString(random.Next(100000000, 310000000));

value = BigInteger.Parse(valueString);

return value;

}

/// <summary>

/// Ф-ция генерации случайного числа типа Big Integer в диапазоне 3 - 310000000

/// </summary>

/// <returns></returns>

private static BigInteger RandomGenerateBigIntegerForFerma()

{

Random random = new Random();

BigInteger value = 0;

string valueString;

valueString = Convert.ToString(random.Next(3, 310000000));

value = BigInteger.Parse(valueString);

return value;

}

/// <summary>

/// Ф-ция теста Миллера-Рабина на простоту

/// </summary>

/// <param name="value"></param>

/// <returns></returns>

private static bool MillerRabinTest(BigInteger value)

{

Thread.Sleep(10);

long totalWitness = 15;

BigInteger[] randomWitness = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 51001, 351011 };

BigInteger oddMultiplier;

long bitLength;

long i, j;

BigInteger result;

oddMultiplier = value - 1;

bitLength = 0;

while (oddMultiplier % 2 == 0)

{

oddMultiplier /= 2;

bitLength++;

}

for (i = 0; i < totalWitness; i++)

{

if (randomWitness[i] == value)

return true;

result = BigInteger.ModPow(randomWitness[i], oddMultiplier, value);

if (result == 1)

continue;

if (result == value - 1)

continue;

for (j = 1; j <= bitLength; j++)

{

result = BigInteger.ModPow(result, 2, value);

if (result == value - 1)

break;

}

if (result != value - 1)

return false;

}

return true;

}

/// <summary>

/// Ф-ция поиска НОД у двух чисел

/// </summary>

/// <param name="a"></param>

/// <param name="b"></param>

/// <returns></returns>

private static BigInteger GCB(BigInteger a, BigInteger b)

{

if (b == 0)

return a;

return GCB(b, a % b);

}

/// <summary>

/// Ф-ция проверки на простоту

/// </summary>

/// <param name="value"></param>

/// <returns></returns>

private static bool IsSimple(BigInteger value)

{

BigInteger end = (BigInteger)Math.Exp(BigInteger.Log(value) / 2);

for (BigInteger i = 3; i < end; i += 2)

{

if (value % i == 0)

return false;

}

return true;

}

/// <summary>

/// Ф-ция теста Ферма на простоту

/// </summary>

/// <param name="x"></param>

/// <returns></returns>

private static bool FermaTest(BigInteger value)

{

for (int i = 0; i < 1000; i++)

{

BigInteger a;

do

{

a = RandomGenerateBigIntegerForFerma();

} while (a > value - 1);

while (GCB(a, value) > 1)

{

do

{

a = RandomGenerateBigIntegerForFerma();

} while (a > value - 1);

}

if (BigInteger.ModPow(a, value - 1, value) != 1)

return false;

}

return true;

}

/// <summary>

/// Действия при нажатии на кнопку "Создание октрытого и закрытого ключей"

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void button1\_Click(object sender, EventArgs e)

{

Enabled = false;

// Вычисляем число P

BigInteger p;

sw.Restart();

do

{

p = RandomGenerateBigInteger();

if (p.IsEven)

{

p--;

}

if (FermaTest(p))

{

break;

}

} while (true);

sw.Stop();

textBoxTimeP.Text = sw.ElapsedMilliseconds + " ms";

textBoxP.Text = p.ToString();

Update();

// Вычисляем число Q

BigInteger q;

sw.Restart();

do

{

q = RandomGenerateBigInteger();

if (q.IsEven)

{

q--;

}

if (MillerRabinTest(q))

{

break;

}

} while (true);

sw.Stop();

textBoxTimeQ.Text = sw.ElapsedMilliseconds + " ms";

textBoxQ.Text = q.ToString();

Update();

// Вычисляем число N

BigInteger n = BigInteger.Multiply(p, q);

textBoxN.Text = n.ToString();

textBoxNLength.Text = textBoxN.Text.Length.ToString();

Update();

// Вычисляем число D

BigInteger d = (p - 1) \* (q - 1);

textBoxD.Text = d.ToString();

Update();

// Вычисляем число S

BigInteger s = q;

textBoxS.Text = s.ToString();

Update();

// Вычисляем число Е

BigInteger E = 0;

BigInteger a = s;

BigInteger b = d;

BigInteger dd = 1; // Остаток

while (a > 0)

{

BigInteger aa = b / a;

BigInteger bb = a;

a = b % a;

b = bb;

bb = dd;

dd = E - aa \* dd;

E = bb;

}

E = E % d;

if (E < 0)

{

E = (E + d) % d;

}

BigInteger eCheck = (E \* s) % d;

textBoxECheck.Text = eCheck.ToString();

Update();

textBoxE.Text = E.ToString();

Update();

// Передаем открытый и закрытый ключ

N = n;

S = s;

Form1.E = E;

Enabled = true;

}

/// <summary>

/// Действие при нажатии кнопки "Зашифровать"

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void button2\_Click(object sender, EventArgs e)

{

if (richTextBox1.Text == "")

{

MessageBox.Show("Введите какой-либо текст");

return;

}

richTextBox2.Text = "";

richTextBox3.Text = "";

// Получаем публичный ключ

BigInteger s = S;

BigInteger n = N;

// Получаем текст

char[] text = richTextBox1.Text.ToCharArray();

int size = richTextBox1.Text.Length;

string m = "1";

// Шифруем текст посимвольно

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

{

string ch = String.Format("{0:d4}", Convert.ToInt32(text[i]));

m = m + ch;

if ((i % 7 == 0) && (i != 0) || i == size - 1)

{

BigInteger c = BigInteger.ModPow(BigInteger.Parse(m), s, n);

richTextBox2.AppendText(c.ToString() + "\n");

m = "1";

}

}

}

/// <summary>

/// Действие при нажатие кнопки "Расшифровать"

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void button3\_Click(object sender, EventArgs e)

{

if (richTextBox2.Text == "")

{

MessageBox.Show("Отсутствует текст для расшифрования");

return;

}

richTextBox3.Text = "";

// Получаем зыкрытый ключ

BigInteger E = Form1.E;

BigInteger n = N;

// Получаем текст

string text = richTextBox2.Text;

int size = richTextBox2.Lines.Length - 1;

// Расшифровываем посимвольно

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

{

BigInteger c = BigInteger.Parse(richTextBox2.Lines[i]);

string m = BigInteger.ModPow(c, E, n).ToString();

for (int j = 1; j < m.Length; j++)

{

if (j % 4 == 0)

{

richTextBox3.Text = richTextBox3.Text +

Convert.ToChar(Convert.ToInt32(m.Substring(j - 3, 4))).ToString();

}

}

}

}

}

1. **Примеры работы программы**

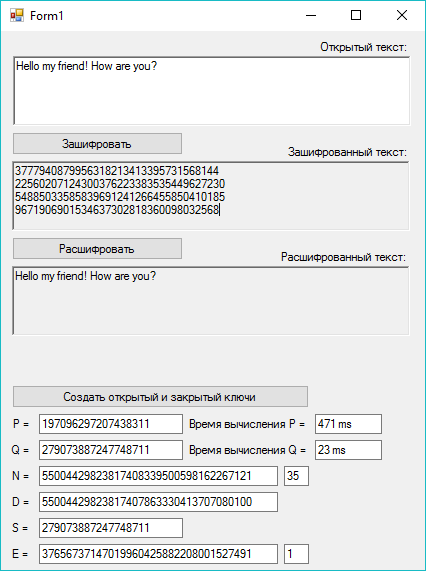


Рисунок 1 – тест Ферма(P) и Миллера-Рабина(Q)

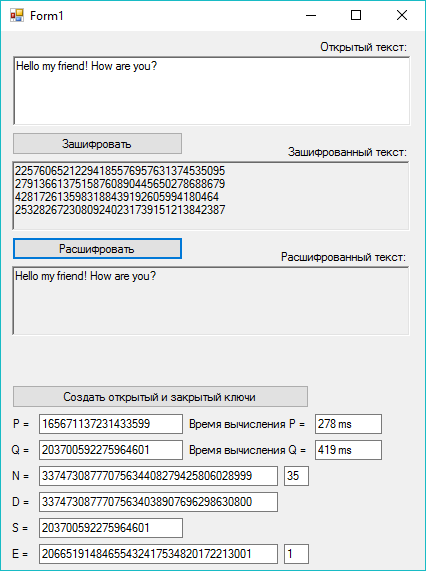


Рисунок 2 – тест Ферма(P) и Миллера-Рабина(Q)

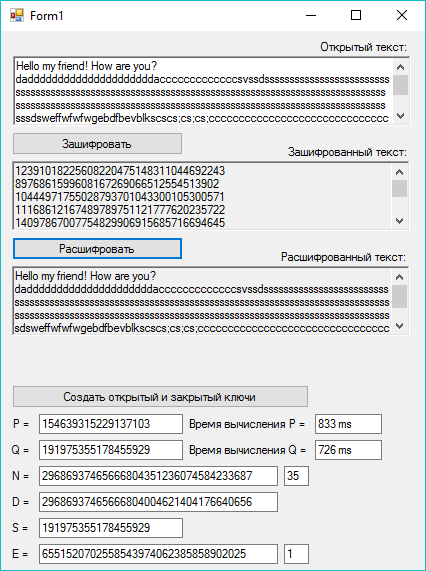


Рисунок 3 – тест Ферма(P) и Миллера-Рабина(Q)

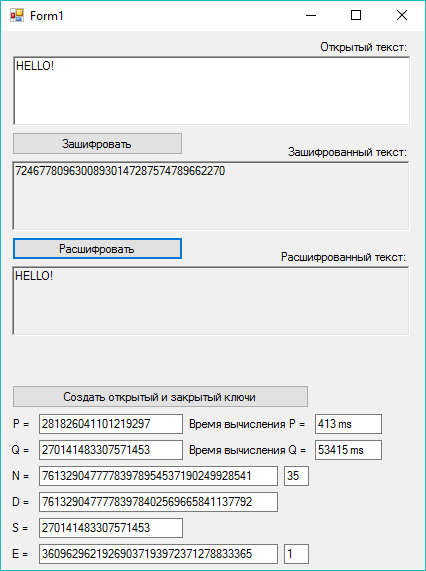


Рисунок 4 – тест Ферма(P) и тест простоты(Q)

1. **Выводы**

В результате проделанной работы был изучены основы ассиметричной криптографии и разработан на их основах алгоритм RSA, проанализирована эффективность алгоритма с различными тестами на простоту и установлены различия в скорости их работы. Также наблюдалась эффективность использования библиотек для длинной арифметики в вычислениях по сравнению с вычислениями без них. Установлена зависимость скорости расшифрования от длины шифротекста.