Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Исследование ассиметричных шифров RSA и Эль-Гамаля**

Студент: Рубашек А. А.

ФИТ 3 курс 5 группа

Преподаватель:

Савельева Маргарита Геннадьевна

1. **Цель работы**

Изучение и приобретение практических навыков разработки и использования приложений для реализации асимметричных шифров RSA и Эль-Гамаля.

1. **Задание**

1. С помощью простого консольного приложения составить табличную или графическую форму зависимости времени вычисления параметра *у*, функционально заданного выражением вида: *у* ≡ *ax* mod *n*, от параметров: *а* (десятичные числа от 5 до 35; можно взять 1 или 2 числа), *х* (числа, желательно простые, из диапазона от 103 до 10100; для примера взять 5–10 чисел, равномерно распределенных в указанном диапазоне), *n* (для примера взять числа, в двоичном виде состоящие из 1024 и 2048 битов).

2. Разработать авторское оконное приложение в соответствии с целью лабораторной работы. При этом можно воспользоваться доступными библиотеками либо программными кодами. В основе вычислений – кодировочные таблицы Base64 и ASCII. Приложение должно реализовывать следующие операции:

• зашифрование и расшифрование текстовых документов на основе алгоритмов RSA и Эль-Гамаля;

• определение времени выполнения операций.

Исходный текст для зашифрования – собственные фамилия, имя, отчество. Для численного представления блоков текста можно в том числе пользоваться указанными выше кодировочными таблицами. Ключевую информацию для обоих алгоритмов можно сгенерировать самостоятельно либо воспользоваться, например, одной из утилит криптографической библиотеки OpenSSL, с помощью которой, в частности, можно сгенерировать ключевую информацию для алгоритма RSA.

3. Используя примерно одинаковый порядок ключевой информации, оценить производительность обоих алгоритмов и относительное изменение объемов криптотекстов (по отношению к объемам открытых текстов).

1. **Ход работы**

Для первого задания в качестве параметров были выбраны следующие значения:

*a* – [5, 15];

*x* – [1000, 3000, 5000, 10000, 100000];

*n* – [21024, 22048].

Далее, по формуле *у* ≡ *ax* mod *n* были вычислены значения *y*, а также было оценено время подсчёта этих значений, которое мы можем увидеть на графике 3.1.

Рисунок 3.1 – Графическая форма зависимости времени вычисления параметра *у*

Для второго задания были разработаны классы ElGamal и Rsa для реализации соответствующих алгоритмов шифрования. Их коды представлены в Приложении 1 и Приложении 2.

Шифруемые строки в обоих случаях – «Rubashek Alexandr Alexandrovich». Тут же была оценена производительность обоих алгоритмов и относительное изменение объемов криптотекстов (по отношению к объемам открытых текстов). В результате при использовании двух шифров получим следующий вывод, представленный на рисунке 3.2.

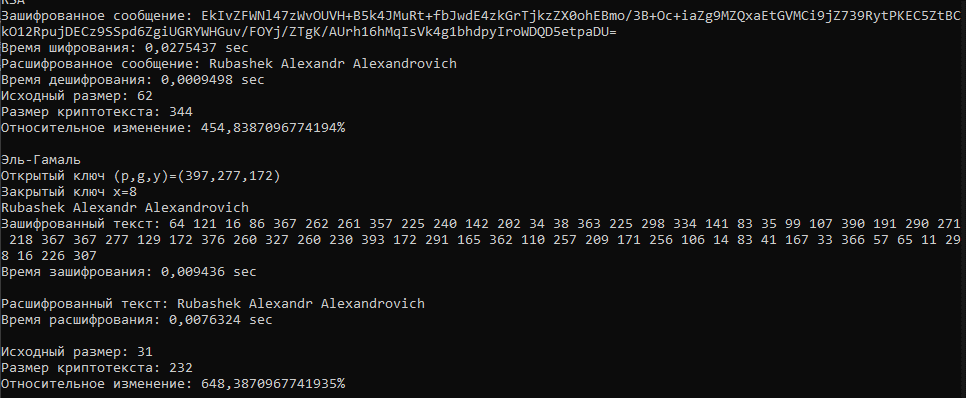
r 

Рисунок 3.2 – Результаты шифрования и расшифрования для Base64

Данный вывод был получен для кодировки Base64. Теперь сравним его с выводом для кодировки ASCII, представленном на рисунке 3.3.

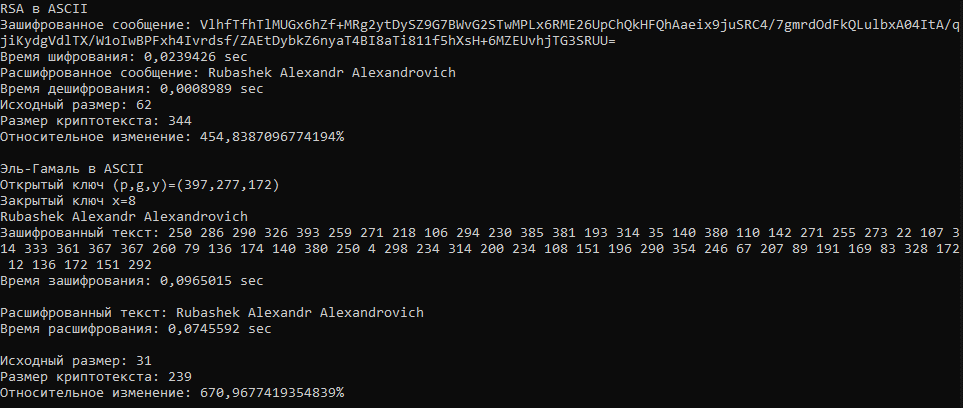


Рисунок 3.3 – Результаты шифрования и расшифрования для ASCII

Исходя из полученных результатов, можно увидеть, что кодировка ASCII справляется быстрее. Это обусловлено тем, что ASCII представляет символы непосредственно в виде однобайтовых значений, тогда как Base64 кодирует данные в последовательность символов, используя шесть битов из каждого байта и добавляя символы заполнения. Расшифровка ASCII-кодированного текста требует простого сопоставления каждого символа с его однобайтовым значением, что делает процесс очень быстрым и эффективным.

Кроме того, посмотрим на графики 3.4 – 3.7, демонстрирующие производительность обоих алгоритмов в разных кодировках:

Рисунок 3.4 – Оценка скорости шифрования и дешифрования для RSA с использованием Base64

Рисунок 3.5 – Оценка скорости шифрования и дешифрования для RSA с использованием Base64

Рисунок 3.6 – Оценка скорости шифрования и дешифрования для RSA с использованием Base64

Рисунок 3.7 – Оценка скорости шифрования и дешифрования для RSA с использованием Base64

Рисунок 3.8 – Оценка эффективности относительно изменения объемов криптотекста

**Вывод**

В ходе лабораторной работы были изучены и приобретены практические навыки разработки и использования приложений для реализации асимметричных шифров RSA и Эль-Гамаля.

Также было разработано авторское приложение в соответствии с целью лабораторной работы.

**Приложение 1**

Класс ElGamal

using System.Numerics;

using System.Text;

public class ElGamalASCII

{

private static BigInteger Exponentiation(BigInteger a, BigInteger b, BigInteger n)

{

var tmp = a;

var sum = tmp;

for (var i = 1; i < b; i++)

{

for (var j = 1; j < a; j++)

{

sum += tmp;

if (sum >= n)

{

sum -= n;

}

}

tmp = sum;

}

return tmp;

}

private static BigInteger Multiplication(BigInteger a, BigInteger b, BigInteger n)

{

var sum = BigInteger.Zero;

for (var i = BigInteger.Zero; i < b; i++)

{

sum += a;

if (sum >= n)

{

sum -= n;

}

}

return sum;

}

public static string Encrypt(string str)

{

return Crypt(397, 277, 8, str);

}

public static string Decrypt(string str)

{

return Decrypt(397, 8, str);

}

private static string Crypt(BigInteger p, BigInteger g, BigInteger x, string inString)

{

var result = "";

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

var rand = new Random();

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

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

foreach (byte code in Encoding.ASCII.GetBytes(inString))

{

if (code > 0)

{

Console.Write((char)code);

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

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

var b = Multiplication(Exponentiation(y, k, p), code, p);

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

}

}

Console.WriteLine("");

return result;

}

private static string Decrypt(BigInteger p, BigInteger x, string inText)

{

var result = "";

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

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

{

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

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

if (a != BigInteger.Zero && b != BigInteger.Zero)

{

var deM = Multiplication(b, Exponentiation(a, p - 1 - x, p), p);

var m = (char)deM;

result += m;

}

}

return result;

}

}

public class ElGamal

{

private static int Exponentiation(int a, int b, int n)

{

var tmp = a;

var sum = tmp;

for (var i = 1; i < b; i++)

{

for (var j = 1; j < a; j++)

{

sum += tmp;

if (sum >= n)

{

sum -= n;

}

}

tmp = sum;

}

return tmp;

}

private static int Multiplication(int a, int b, int n)

{

var sum = 0;

for (var i = 0; i < b; i++)

{

sum += a;

if (sum >= n)

{

sum -= n;

}

}

return sum;

}

public static string Encrypt(string str)

{

return Crypt(397, 277, 8, str);

}

public static string Decrypt(string str)

{

return Decrypt(397, 8, str);

}

private static string Crypt(int p, int g, int x, string inString)

{

var result = "";

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

var rand = new Random();

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

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

foreach (int code in inString)

{

if (code > 0)

{

Console.Write((char)code);

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

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

var b = Multiplication(Exponentiation(y, k, p), code, p);

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

}

}

Console.WriteLine("");

return result;

}

private static string Decrypt(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 = Multiplication(b, Exponentiation(a, p - 1 - x, p),

p);

var m = (char)deM;

result += m;

}

}

return result;

}

}

**Приложение 2**

Класс Rsa

using System.Text;

public static class RsaASCII

{

public static string Encrypt(string str, string publicKeyXml)

{

var testData = Encoding.ASCII.GetBytes(str);

using (var rsa = RSA.Create())

{

rsa.FromXmlString(publicKeyXml);

var encryptedData = rsa.Encrypt(testData, RSAEncryptionPadding.Pkcs1);

var base64Encrypted = Convert.ToBase64String(encryptedData);

return base64Encrypted;

}

}

public static string Decrypt(string str, string privateKeyXml)

{

var resultBytes = Convert.FromBase64String(str);

using (var rsa = RSA.Create())

{

rsa.FromXmlString(privateKeyXml);

var decryptedBytes = rsa.Decrypt(resultBytes, RSAEncryptionPadding.Pkcs1);

var decryptedData = Encoding.ASCII.GetString(decryptedBytes);

return decryptedData;

}

}

}

public static class Rsa

{

public static string GenerateKey()

{

using (var rsa = RSA.Create())

{

rsa.KeySize = 1024;

return rsa.ToXmlString(true);

}

}

public static string Encrypt(string str, string publicKeyXml)

{

var testData = Encoding.UTF8.GetBytes(str);

using (var rsa = RSA.Create())

{

rsa.FromXmlString(publicKeyXml);

var encryptedData = rsa.Encrypt(testData, RSAEncryptionPadding.Pkcs1);

var base64Encrypted = Convert.ToBase64String(encryptedData);

return base64Encrypted;

}

}

public static string Decrypt(string str, string privateKeyXml)

{

using (var rsa = RSA.Create())

{

rsa.FromXmlString(privateKeyXml);

var resultBytes = Convert.FromBase64String(str);

var decryptedBytes = rsa.Decrypt(resultBytes, RSAEncryptionPadding.Pkcs1);

var decryptedData = Encoding.UTF8.GetString(decryptedBytes);

return decryptedData;

}

}

}