Учреждение образования

«Белорусский государственный технологический университет»

**Отчет по лабораторной работе №7**

по дисциплине: «Защита информации и надежность информационных систем»

Исполнитель:

Воликов Д. А., ФИТ 4-7

Руководитель:

Асс. Савельева М. Г.

Минск 2024

# Лабораторная работа №7. Исследование асимметричных шифров

**Цель**: изучение и приобретение практических навыков разработки и использования приложений для реализации асимметричных шифров.

**Задачи**:

1. Закрепить теоретические знания по алгебраическому описанию, алгоритмам реализации операций зашифрования/расшифрования и оценке криптостойкости асимметричных шифров.
2. Разработать приложение для реализации указанных преподавателем методов генерации ключевой информации и ее использования для асимметричного зашифрования/расшифрования.
3. Выполнить анализ криптостойкости асимметричных шифров.
4. Оценить скорость зашифрования/расшифрования реализованных шифров.
5. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

## Теоретические сведения

Две известные проблемы, связанные с практическим использованием симметричных криптосистем, стали важными побудительными мотивами для разработки принципиально нового класса методов шифрования: криптографии с открытым ключом, или асимметричной криптографии.

Концепция нового подхода предложена Уитфилдом Диффи (Whitfield Diffie) и Мартином Хеллманом (Martin Hellman) и, независимо, Ральфом Мерклом (Ralph Merkle).

В основу асимметричной криптографии положена идея использовать ключи парами: один – для зашифрования (открытый, или публичный, ключ), другой – для расшифрования (тайный ключ). Отметим, что указанная пара ключей принадлежит получателю зашифрованного сообщения. Все алгоритмы шифрования с открытым ключом основаны на использовании односторонних функций, к числу которых, как известно, относится вычисление дискретного логарифма.

Односторонней функцией (one-way function) называется математическая функция, которую относительно легко вычислить, но трудно найти по значению функции соответствующее значение аргумента, т. е. зная *х*, легко вычислить *f*(*x*), но по известному *f*(*x*) трудно найти подходящее значение *x*.

Практически первой реализацией идеи Диффи – Хеллмана стал алгоритм согласования по открытому каналу тайного ключа между абонентами *А* и *В*.

Алгоритмы шифрования с открытым ключом можно использовать для решения следующих задач:

* зашифрования/расшифрования передаваемых и хранимых данных в целях их защиты от несанкционированного доступа;
* формирования цифровой подписи под электронными документами;
* распределения секретных ключей, используемых далее при шифровании документов симметричными методами.

Криптоалгоритм на основе задачи об укладке ранца. Алгоритм разработан Р. Мерклом и М. Хеллманом. Это первый алгоритм шифрования с открытым ключом широкого назначения.

Ранцевый (рюкзачный) вектор *S* = (*s*1, ..., *s*z ) – это упорядоченный набор из *z*, *z* ≥ 3, различных натуральных чисел *s*i. Входом задачи о ранце (рюкзаке) называем пару (*S*’, *S*), где *S*’ – рюкзачный вектор, а *S* – натуральное число.

Решением для входа (*S*’, *S*) будет такое подмножество из *S*’, сумма элементов которого равняется *S*.

В наиболее известном варианте задачи о ранце требуется выяснить, обладает или нет данный вход (*S*’, *S*) решением. В варианте, используемом в криптографии, нужно для данного входа (*S*’, *S*) построить решение, зная, что такое решение существует. Оба эти варианта являются NP-полными.

В качестве закрытого ключа *d* (легкого для укладки ранца) используется сверхвозрастающая последовательность, состоящая из *z* элементов: *d*1, *d*2, …, *d*z: *d* = {*d*i}, *i* = 1, …, *z*. Например, пусть *d* = { 2, 3, 6, 13, 27, 52, 105, 210 }.

Сверхвозрастающей последовательностью называется последовательность, в которой каждый последующий член больше суммы всех предыдущих.

Открытый ключ *e* представляет собой нормальную (не сверхвозрастающую) последовательность. Он формируется на основе закрытого ключа и не позволяет легко решить задачу об укладке ранца.

Для получения открытого ключа *e* (*e* = {*e*i}, *i* = 1, …, *z*) все значения закрытого ключа умножаются на некоторое число *a* по модулю *n*:

*e*i ≡ *d*i ⋅ *a* mod *n*.

Значение модуля *n* должно быть больше суммы всех чисел последовательности; кроме того, НОД (*а*, *n*) = 1.

Для зашифрования сообщения (*М*) оно сначала разбивается на блоки, по размерам равные числу (*z*) элементов последовательности в ранце. Затем, считая, что 1 указывает на присутствие элемента последовательности в ранце, а 0 – на его отсутствие, вычисляются полные веса рюкзаков (*S*i, *i* = 1, …, *z*): по одному ранцу для каждого блока сообщения с использованием открытого ключа получателя *e*.

Возьмём для примера следующий открытый ключ *е*: {62, 93, 186, 403, 417, 352, 315, 210}.

Результат зашифрования (упаковки ранца) каждого блока (буквы) сообщения с помощью открытого ключа представлен в правом столбце на рисунке ниже.

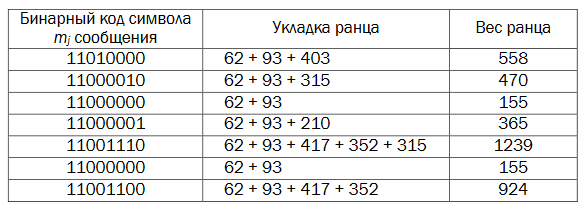


Рисунок 1 – Пояснение к примеру зашифрования сообщения укладкой ранца

Таким образом, зашифрованное сообщение С = «558 470 155 365 1239 155 924»: *с*1 = 558, *с*2 = 470 и т. д.

Для расшифрования сообщения получатель (используя свой тайный ключ *d*: сверхвозрастающую последовательность) должен сначала определить такое обратное к *а* число *а*–1, что

*а ⋅ а*–1 mod *n* ≡ 1.

Для вычисления обратных чисел по модулю можно использовать известный нам расширенный алгоритм Евклида.

После определения обратного числа каждое значение шифрограммы (*c*i) преобразуется в соответствии со следующим соотношением:

*S*i ≡ *c*i ⋅ *а*–1 mod *n*.

Полученное на основании последней формулы для каждого блока число далее рассматривается как заданный вес ранца, который следует упаковать по изложенному выше алгоритму, используя сверхвозрастающую последовательность (тайный ключ получателя).

Вспомним, что сверхвозрастающая последовательность равна *d*: *d* = {2, 3, 6, 13, 27, 52, 105, 210}, а также *n* = 420, *a* = 31; шифртекст: С = «155 365 558 155 924 1239 470».

Расшифрование первого блока шифртекста. Сначала вычисляем вес первого ранца (при *c*1 = 155):

*S*1 = *c*1 ⋅ *а*–1 mod *n* = 155 · 271 mod 420 = 5.

Используем *S*1 = 5 и с помощью сверхвозрастающей последовательности ({2, 3, 6, 13, 27, 52, 105, 210}) и известного алгоритма упаковки ранца получаем *m*1 = 11000000. Понятно, что последней бинарной последовательности должен соответствовать некоторый символ алфавита в используемой таблице кодировки. Расшифрование остальных блоков шифртекста производится аналогично.

## Практическое задание

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:

* генерация сверхвозрастающей последовательности (тайного ключа); старший член последовательности – 100-битное число; в простейшем случае принимается *z* = 6 (для кодировки Base64) и *z* = 8 (для кодировки ASCII);
* вычисление нормальной последовательности (открытого ключа);
* зашифрование сообщения, состоящего из собственных фамилии, имени и отчества;
* расшифрование сообщения;
* оценка времени выполнения операций зашифрования и расшифрования.

Оценим работу зашифрования и расшифрования сообщения на основе кодировки ASCII (*z* = 8).

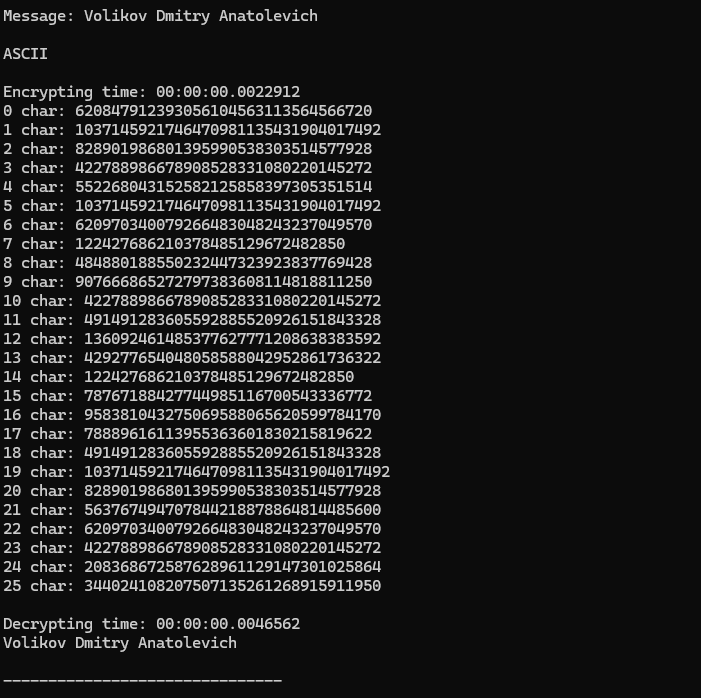


Рисунок 2 – Зашифрование/расшифрование по кодировке ASCII

Как можно заметить, каждый символ сообщения после зашифрования представлен большим числом (ибо числа возрастающей последовательности имеют размер больше 64 битов).

Оценим работу зашифрования и расшифрования сообщения на основе кодировки Base64 (*z* = 6).

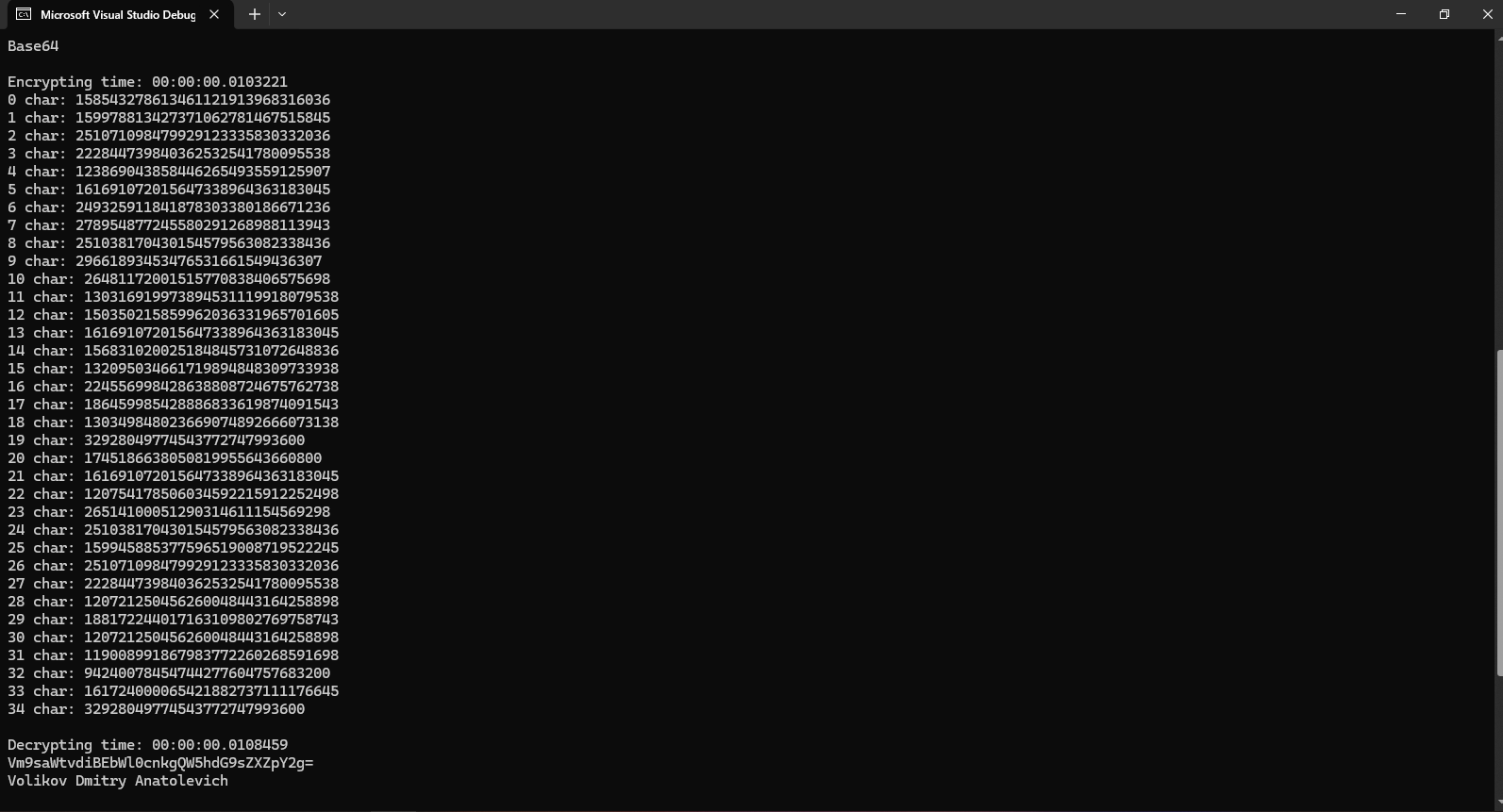


Рисунок 3 – Зашифрование/расшифрование по кодировке Base64

Легко заметить, что зашифрованное сообщение стало больше. Это всё из-за того, что каждый символ в Base64 занимает 6 битов. Соответственно, количество символов в зашифрованном сообщении будет больше.

Теперь оценим время зашифрования каждой из кодировок.

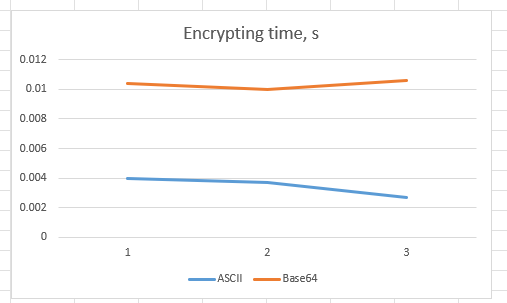


Рисунок 4 – Время зашифрования сообщения алгоритмом

Теперь оценим время расшифрования каждой из кодировок.

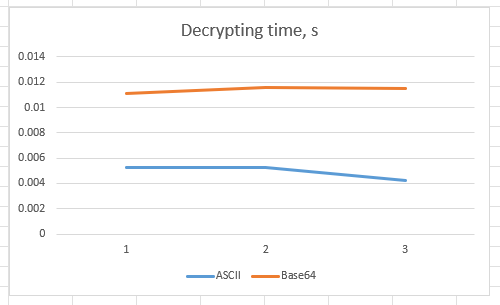


Рисунок 5 – Время расшифрования сообщения алгоритмом

Вывод: в ходе этой лабораторной работы я изучил основы асимметричных шифров. Изучил алгоритм работы асимметричного шифрования на основе задачи об укладке ранца. Реализовал программу, которая зашифровывает/расшифровывает сообщение, а также оценивает время работы алгоритма.

# Листинг «Лабораторная работа №5»

using Lab7.Encoders;

using Lab7.Encryptors;

const string message = "Volikov Dmitry Anatolevich";

var asciiEncryptor = new AsciiBackpack();

Console.WriteLine($"Message: {message}\n");

Console.WriteLine("ASCII\n");

var watch = System.Diagnostics.Stopwatch.StartNew();

var asciiEncryptedMessage = asciiEncryptor.Encrypt(message);

watch.Stop();

Console.WriteLine($"Encrypting time: {watch.Elapsed}");

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

{

Console.WriteLine($"{i} char: {asciiEncryptedMessage[i]}");

}

watch.Start();

var asciiDecryptedMessage = asciiEncryptor.Decrypt(asciiEncryptedMessage);

watch.Stop();

Console.WriteLine($"\nDecrypting time: {watch.Elapsed}");

Console.WriteLine(asciiDecryptedMessage);

Console.WriteLine("\n-------------------------------\n");

Console.WriteLine("\nBase64\n");

var base64Encryptor = new Base64Backpack();

watch.Start();

var base64EncryptedMessage = base64Encryptor.Encrypt(message);

watch.Stop();

Console.WriteLine($"Encrypting time: {watch.Elapsed}");

for (int i = 0; i < base64EncryptedMessage.Length; i++)

{

Console.WriteLine($"{i} char: {base64EncryptedMessage[i]}");

}

watch.Start();

var base64DecryptedMessage = base64Encryptor.Decrypt(base64EncryptedMessage);

watch.Stop();

Console.WriteLine($"\nDecrypting time: {watch.Elapsed}");

Console.WriteLine(base64DecryptedMessage);

Console.WriteLine(Base64.Decode(base64DecryptedMessage));

Листинг 6.1 – Код программы

using System.Numerics;

using System.Text;

namespace Lab7.Encryptors

{

public class AsciiBackpack : Backpack

{

public AsciiBackpack() : base(8) { }

public override BigInteger[] Encrypt(string message)

{

BigInteger[] encryptedMessage = new BigInteger[message.Length];

byte[] bytes = Encoding.ASCII.GetBytes(message);

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

{

string bits = Convert.ToString(bytes[i], 2).PadLeft(8, '0');

for (int j = 0; j < 8; j++)

{

if (bits[j] == '1')

{

encryptedMessage[i] += PublicKey![j];

}

}

}

return encryptedMessage;

}

public override string Decrypt(BigInteger[] message)

{

byte[] bytes = new byte[message.Length];

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

{

string byt = "";

BigInteger sNumber = (message[i] \* InversedA) % N;

for (int j = 7; j >= 0; j--)

{

byt += sNumber >= \_privateKey![j] ? '1' : '0';

sNumber = sNumber >= \_privateKey![j] ? sNumber -= \_privateKey![j] : sNumber;

}

byt = new string(byt.Reverse().ToArray());

foreach (var bit in byt)

{

bytes[i] = (byte)((bytes[i] << 1) + (bit == '1' ? 1 : 0));

}

}

return Encoding.ASCII.GetString(bytes);

}

}

}

Листинг 6.2 – Код шифратора на основе ASCII

using Lab7.Encoders;

using System.Numerics;

namespace Lab7.Encryptors

{

public class Base64Backpack : Backpack

{

private readonly int Base64ByteSize = 6;

public Base64Backpack() : base(6) { }

public override BigInteger[] Encrypt(string message)

{

var bits = Base64.Utf8ToBase64Binary(message);

BigInteger[] encryptedMessage = new BigInteger[bits.Length / Base64ByteSize];

for (int i = 0; i < bits.Length / Base64ByteSize; i++)

{

var base64Byte = new string(bits.Skip(i \* Base64ByteSize).Take(Base64ByteSize).ToArray());

for (int j = 0; j < Base64ByteSize; j++)

{

if (base64Byte[j] == '1')

{

encryptedMessage[i] += PublicKey![j];

}

}

}

return encryptedMessage;

}

public override string Decrypt(BigInteger[] message)

{

string decodedBits = "";

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

{

string base64Byte = "";

BigInteger sNumber = (message[i] \* InversedA) % N;

for (int j = 5; j >= 0; j--)

{

base64Byte += sNumber >= \_privateKey![j] ? '1' : '0';

sNumber = sNumber >= \_privateKey![j] ? sNumber -= \_privateKey![j] : sNumber;

}

decodedBits += new string(base64Byte.Reverse().ToArray());

}

return Base64.EncodeBase64Binary(decodedBits);

}

}

}

Листинг 6.3 – Код шифратора на основе Base64