Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Отчет по лабораторной работе №9

«Исследование асимметричных шифров»

Студентка: Пунько А.А,

ФИТ 3 курс 5 группа

Преподаватель: Берников В. О.

Минск 2020

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 нам 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.

В данной работе мы будем работать над аспектами решения первой из указанных задач. По мнению Диффи и Хеллмана алгоритм шифрования с открытым ключом, должен:

* вычислительно легко создавать пару (открытый ключ, e – закрытый ключ, d),
* вычислительно легко зашифровывать сообщение Mi открытым ключом,
* вычислительно легко расшифровывать сообщение Ci, используя закрытый ключ,
* обеспечивать непреодолимую вычислительную сложность определения соответствующего закрытого ключа при известном открытом ключе,
* обеспечивать непреодолимую вычислительную сложность восстановления исходного (открытого сообщения, Mi) зная только открытый ключ и зашифрованное сообщение, Ci.

Ранцевый (рюкзачный) вектор S = (s1, . . ., sz) – это упорядоченный набор из z, z ≥ 3, различных натуральных чисел si. 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. Имеются также варианты этой задачи, которые не лежат даже в классе NP.

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

Необходимо по очереди анализировать некоторый «текущий вес» S предметов, составляющих сверхвозрастающую последовательность; в результате анализа нужно упаковать (доупаковать) ранец.

1. В качестве текущего выбирается число S, которое сравнивается с «весом» самого тяжелого предмета (dz); если текущий вес меньше веса данного предмета, то его в ранец не кладут (0), в противном случае его укладывают (bz = 1) в ранец и переходят к анализу очередного (в общем случае – i-го предмета).

2. Если на предыдущем (i-м шаге) предмет пополнил ранец, то текущий вес уменьшают на вес положенного предмета (S = S – di); переходят к следующему по весу предмету в последовательности: di-1.

Шаги повторяются до тех пор, пока процесс не закончится. Если текущий вес уменьшится до нуля (S = 0), то решение найдено. В противном случае – нет.

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 = {ei}, i = 1,…, z) все значения закрытого ключа умножаются на некоторое число a по модулю n: ei = di \* 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 (Si, i = 1, . . ., z): по одному ранцу для каждого блока сообщения с использованием открытого ключа получателя, e.

Для расшифрования сообщения получатель (использует свой тайный ключ, d: сверхвозрастающую последовательность) должен сначала определить обратное к а число: а-1, такое 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 (ci) преобразуется в соответствии со следующим соотношением: Si = ci 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).

# Практическая часть

Целью данной работы является разработка приложения, осуществляющего шифрование и расшифрование текста с использованием ранцевого алфавита. Стартовое окно приложения изображено на рисунке 1.

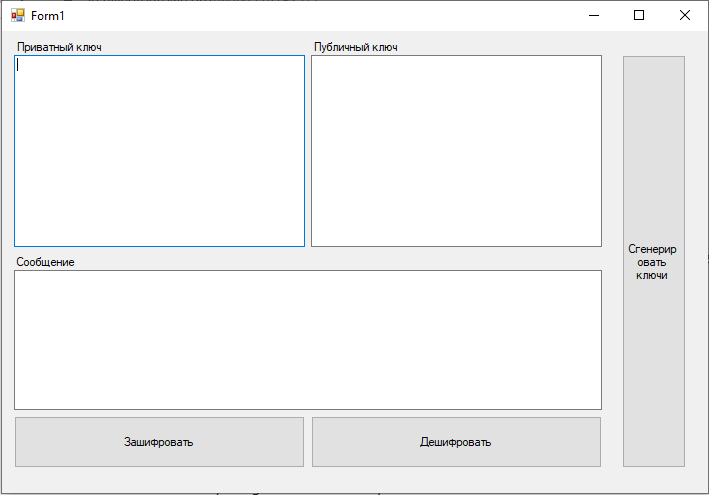


Рисунок 1 – Стартовое окно приложения

Результат генерации ключей представлен на рисунке 2.

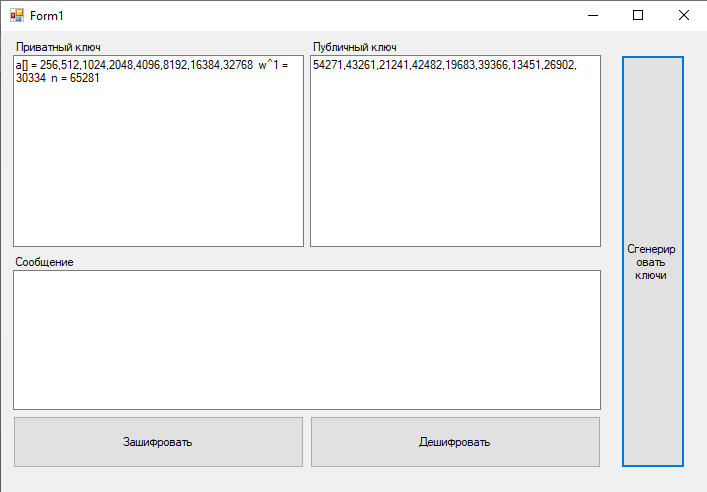


Рисунок 2 – Результат генерации ключей

Результат шифрования представлен на рисунке 3.

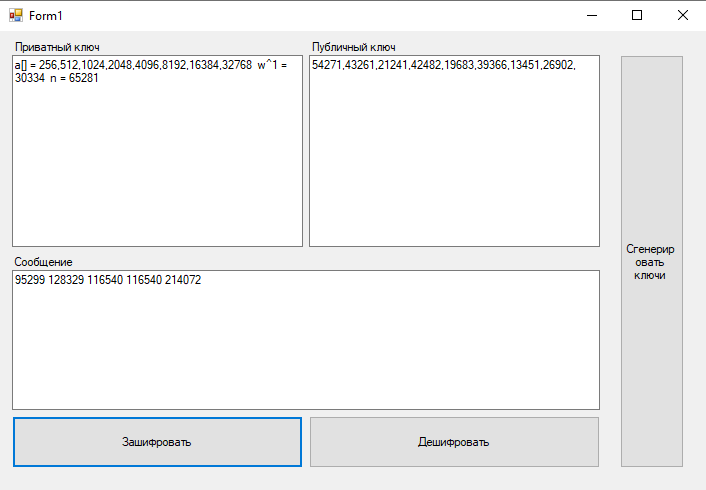


Рисунок 3 – Результат шифрования

Результат дешифрования представлен на рисунке 4.

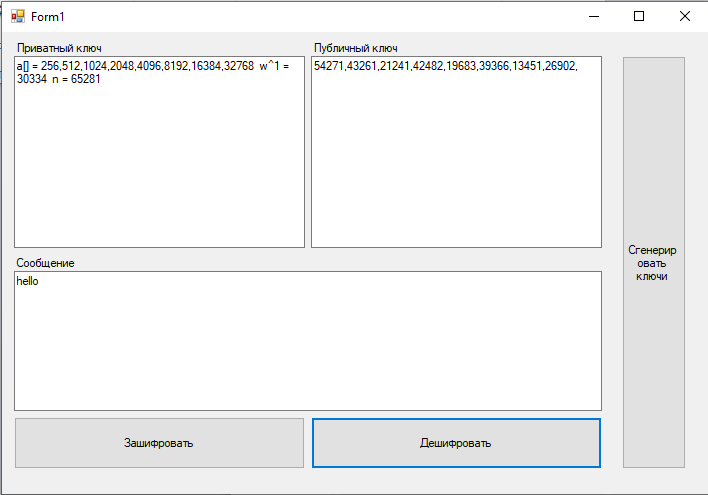


Рисунок 4 – Результат дешифрования

Функция для генерации ключей представлена в листинге 1. При этом генерируется сверхвозрастающая последовательность a, и два больших взаимообратных числа e и w. Эти числа являются закрытым ключом. Открытый ключ e вычисляется с использованием формулы ei= w\*ai mod n

public MHKey()

{

generator gen = new generator(8);

BigInteger w = gen.w;

e = new List<BigInteger>();

for (int i = 0; i < gen.a.Count; i++)

{

e.Add(w.Multiply(gen.a.ElementAt(i)).Mod(gen.n)); // w\*a Mod n ..

}

BigInteger w1 = BigInteger.ValueOf(1);

BigInteger one = BigInteger.ValueOf(2);

while (one.IntValue != 1)

{

w1 = w1.Add(BigInteger.ValueOf(1)).Mod(gen.n);

one = w.Multiply(w1).Mod(gen.n);

}

privateKey = new MHPrivateKey(gen.a, w.ModInverse(gen.n), gen.n);

}

Листинг 1 – Алгоритм генерации ключей

Функция для шифрования представлена в листинге 2. В процессе работы функции каждый символ переводится в бинарный вид (8 бит). Далее ищутся биты с единицей и складываются соответствующие числа из открытого ключа.

public List<BigInteger> encipher(String plain)

{

String binary;

BigInteger tmp;

List<BigInteger> enciphered = new List<BigInteger>();

char[] charArray = plain.ToCharArray();

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

{

tmp = BigInteger.ValueOf(0);

binary = charToBinary(charArray[i]);

char one = '1';

for (int j = binary.Length - 1; j > -1; j--)

{

if (binary.ElementAt(j) == one)

{

tmp = tmp.Add((BigInteger)e.ElementAt(7 - j));

}

}

enciphered.Add(tmp);

}

return enciphered;

}

Листинг 2 – Алгоритм шифрования

Функция для дешифрования представлена в листинге 3

public String decipher(List<BigInteger> cipher, MHPrivateKey key)

{

String decrypted = "";

BigInteger temp = BigInteger.ValueOf(0);

int tmp = 0;

BigInteger bits = BigInteger.ValueOf(0);

for (int i = 0; i < cipher.Count; i++)

{

temp = cipher.ElementAt(i);

int bitlen = temp.BitLength;

int ff = 0;

while (bitlen < (int)Math.Pow(2, ff))

{

ff++;

}

if (ff > bitlen)

bitlen = ff;

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

{

if (temp.Mod(BigInteger.ValueOf(2)).CompareTo(BigInteger.ValueOf(1)) == 0)

{

bits = bits.Add(key.w1.Multiply(BigInteger.ValueOf((long)Math.Pow(2, j))));

}

temp = temp.ShiftRight(1);

}

bits = bits.Mod(key.n);

List<BigInteger> list = (List<BigInteger>)key.a;

BigInteger temper;

int k = key.a.Count - 1;

while (k >= 0)

{

temper = list.ElementAt(k);

if (bits.CompareTo(temper) > -1)

{

tmp += (int)Math.Pow(2, k);

bits = bits.Subtract(temper);

}

k--;

}

decrypted += (binaryToChar(Convert.ToString(tmp, 2))).ToString();

bits = BigInteger.ValueOf(0);

tmp = 0;

}

return decrypted;

}.

Листинг 3 – Алгоритм дешифрования

# Вывод

В процессе выполнения данной работы был изучен ранцевый метод шифрования как один из методов ассиметричного шифрования, было создано приложение, реализующее генерацию ключей, шифрование и дешифрование текста.