Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

[Кафедра информационных](https://www.belstu.by/fakultety/fit/vm) систем и технологий

Специальность 1-40 01 01 Программное обеспечение информационных технологий

**Отчёт по лабораторной работе №9**

по дисциплине Информационная безопасность

Тема: Исследование ассиметричных шифров

        Исполнитель:

Студент 3 курса группы 4

Станчик Максим Андреевич

Руководитель:

Ассистент Савельева М.Г.

Минск, 2025

**Лабораторная работа №9**

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

**Задачи:**

1. Закрепить теоретические знания по алгебраическому описанию, алгоритмам реализации операций зашифрования/расшифрования и оценке криптостойкости ассиметричных шифров.

2. Разработать приложение для реализации указанных преподавателем методов генерации ключевой информации и ее использования для ассиметричного зашифрования/расшифрования.

3. Выполнить анализ криптостойкости ассиметричных 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

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 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*.

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.

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*.

Суть метода для шифрования состоит в том, что существуют две различные задачи укладки ранца: одна из них решается легко и характеризуется линейным ростом трудоемкости, а другая решается трудно. Легкий для укладки ранец можно трансформировать в трудный. Трудный для укладки ранец применяется в качестве открытого ключа, который легко использовать для зашифрования, но невозможно – для расшифрования. В качестве закрытого ключа применяется легкий для укладки ранец, который предоставляет простой способ расшифрования сообщения.

В качестве закрытого ключа *d* (легкого для укладки ранца) используется сверхвозрастающая последовательность, состоящая из *z* элементов: *d1*, *d2*, …, *dz*: *d* = {*di*}, *i* = 1, …, *z*.

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

Ранцевые криптосистемы не являются криптостойкими. А. Шамир и Р. Циппел обнаружили, что зная числа *а*, *a-1* и *n*, можно восстановить сверхвозрастающую последовательность по нормальной последовательности.

# Ход работы

В начале, нужно было разработать приложение, реализующее генерацию приватного и публичного ключа, а также осуществляющее зашифрование и расшифрование сообщения, состоящего из собственных фамилии, имени и отчества.

Метод generatePrivateKey предназначен для генерации последовательности чисел, которые в качестве приватного ключа в ассиметричном шифровании. Он используется для создания приватного ключа. Принимает в качестве параметров начальное значение для генерации последовательности (первый элемент в возвращаемом списке) и количество членов (элементов) в создаваемой последовательности. Метод возвращает список объектов, представляющий сгенерированную последовательность чисел. Код метода представлен в листинге 2.1.

public static List<BigInteger> generatePrivateKey (BigInteger initialNumber, int z) {

List<BigInteger> sequence = new ArrayList<>();

BigInteger element = initialNumber;

BigInteger sum = initialNumber;

for (int i = 0; i < z; i++) {

sequence.add(element);

element = sum.add(BigInteger.valueOf(z));

sum = sum.add(element);

}

return sequence;

}

Листинг 2.1 – Код метода генерации приватного ключа

Метод generateRandomNumber предназначен для генерации случайного числа, представленное в двоичном формате. Это начальное число потом используется в создании приватных и публичных ключей. Метод принимает количество бит, которые нужно сгенерировать. Это определяет длину создаваемого случайного числа. Метод возвращает объект, который представляет собой случайное целое число, сформированное из сгенерированных двоичных бит. Код метода представлен в листинге 2.2.

public static List<BigInteger> generatePrivateKey (BigInteger initialNumber, int z) {

List<BigInteger> sequence = new ArrayList<>();

BigInteger element = initialNumber;

BigInteger sum = initialNumber;

for (int i = 0; i < z; i++) {

sequence.add(element);

element = sum.add(BigInteger.valueOf(z));

sum = sum.add(element);

}

return sequence;

}

Листинг 2.2 – Код метода генерации случайного числа в двоичном формате

Метод getPublicKeyParams предназначен для вычисления параметров публичного ключа на основе предоставленного приватного ключа. Эти параметры потом используются для шифрования и расшифрования данных. Принимает список целых чисел, представляющий приватный ключ. Он используется для вычисления суммы и определения других параметров публичного ключа. Код метода представлен в листинге 2.3.

public static BigInteger[] getPublicKeyParams(List<BigInteger> privateKey) {

BigInteger sum = privateKey.stream().reduce(BigInteger.ZERO, BigInteger::add);

BigInteger n = sum.add(BigInteger.ONE);

BigInteger a = generateCoprime(n);

return new BigInteger[]{a, n};

}

Листинг 2.3 – Код метода генерации случайного числа в двоичном формате

Метод encrypt предназначен для шифрования строки (текста) с использованием публичного ключа и заданной кодировки (ASCII или Base64). Он преобразует текст в зашифрованный формат, который может быть использован для безопасной передачи данных. Принимает в качестве параметров публичный ключ, используемый для шифрования (список чисел, которые представляют криптографические значения), исходный текст, который необходимо зашифровать, и перечисление, указывающее, какую кодировку использовать (ASCII или Base64). Код метода представлен в листинге 2.4.

public static List<BigInteger> encrypt(List<BigInteger> publicKey, String plaintext, Encoding encoding) {

List<BigInteger> encryptedList = new ArrayList<>();

if (encoding == Encoding.BASE64) {

plaintext = base64Encode(plaintext);

}

for (int index = 0; index < plaintext.length(); index++) {

char b = plaintext.charAt(index);

String binaryString;

if (encoding == Encoding.ASCII) {

binaryString = String.format("%8s", Integer.toBinaryString(b)).replace(' ', '0');

} else {

binaryString = convertBase64ToBinary(String.valueOf(b));

}

List<Integer> positions = new ArrayList<>();

for (int i = 0; i < binaryString.length(); i++) {

if (binaryString.charAt(i) == '1') {

positions.add(i);

}

}

BigInteger sum = BigInteger.ZERO;

for (int position : positions) {

if (position < publicKey.size()) {

sum = sum.add(publicKey.get(position));

}

}

encryptedList.add(sum);

}

return encryptedList;

}

Листинг 2.4 – Код метода шифрования текста

Метод decrypt предназначен для расшифровки зашифрованного текста, используя приватный ключ и параметры, полученные при шифровании. Он преобразует зашифрованные данные обратно в исходный текст. Принимает в качестве параметров приватный ключ, используемый для расшифровки (список чисел, которые представляют криптографические значения), список зашифрованных чисел, представляющий текст, который необходимо расшифровать, *a*, параметр, используемый в процессе расшифровки, который должен быть взаимно простым с *n*, и сам *n*, модуль, используемый в процессе расшифровки, который также определяется во время шифрования. Код метода представлен в листинге 2.5.

public static Object[] decrypt(List<BigInteger> privateKey, List<BigInteger> encryptedText, BigInteger a, BigInteger n) {

StringBuilder binaryResult = new StringBuilder();

BigInteger inverse = getInverseNumber(a, n);

for (BigInteger cipher : encryptedText) {

BigInteger decryptedValue = cipher.multiply(inverse).mod(n);

String binaryString = getDecryptedBinary(decryptedValue, privateKey);

binaryResult.append(binaryString);

}

byte[] byteArray = binaryStringToByteArray(binaryResult.toString());

return new Object[]{byteArray, binaryResult.toString()};

}

Листинг 2.5 – Код метода расшифрования текста

Метод getDecryptedBinary предназначен для преобразования заданного целого числа в двоичное представление, используя приватный ключ. Этот метод используется в процессе расшифровки, чтобы получить двоичную строку, соответствующую расшифрованному числу. Принмиает в качестве параметров целое число, которое необходимо преобразовать в двоичное представление. Это число обычно получается в результате расшифровки, приватный ключ, который используется для определения значений, которые будут представлены в двоичном формате (этот список содержит числа, которые помогают формировать двоичное представление). Метод возвращает строку, представляющую двоичное значение числа, сформированное на основе приватного ключа. Код метода представлен в листинге 2.6.

public static String getDecryptedBinary(BigInteger number, List<BigInteger> privateKey) {

StringBuilder binaryString = new StringBuilder();

for (int i = privateKey.size() - 1; i >= 0; i--) {

if (number.compareTo(privateKey.get(i)) >= 0) {

binaryString.append('1');

number = number.subtract(privateKey.get(i));

} else {

binaryString.append('0');

}

}

return binaryString.reverse().toString();

}

Листинг 2.6 – Код метода для преобразования целого числа в двоичное представление с использованием приватного ключа

Перечисление Encoding предназначено для определения доступных типов кодировки, которые могут использоваться при обработке текстовых данных в криптографических алгоритмах. Оно упрощает управление различными способами представления данных. Содержит в себе такие элементы перечисления, как ASCII и BASE64. Код перечисления представлен в листинге 2.7.

enum Encoding {

ASCII,

BASE64

}

Листинг 2.7 – Код представления для типа кодировки

Метод binaryStringToByteArray предназначен для преобразования двоичной строки в массив байтов. Нужен для конвертации текстовых данных, представленных в двоичном формате, в формат, подходящий для хранения или передачи. В качестве параметров принимает двоичную строку, которая будет преобразована в массив байтов. Строка должна состоять из символов '0' и '1'. Метод возвращает массив байтов, полученный из двоичной строки. Код метода представлен в листинге 2.8.

public static byte[] binaryStringToByteArray(String binaryString) {

int byteCount = (binaryString.length() + 7) / 8;

byte[] bytes = new byte[byteCount];

for (int i = 0; i < byteCount; i++) {

int startIndex = i \* 8;

int endIndex = Math.min(startIndex + 8, binaryString.length());

String byteString = binaryString.substring(startIndex, endIndex);

while (byteString.length() < 8) {

byteString = "0" + byteString;

}

bytes[i] = (byte) Integer.parseInt(byteString, 2);

}

return bytes;

}

Листинг 2.8 – Код метода для преобразования двоичной строки в массив байтов

Метод convertBase64ToBinary предназначен для преобразования строки, закодированной в формате Base64, в двоичное представление. Это позволяет работать с данными в бинарном формате. В качестве параметров принимает строку, закодированную в формате Base64, которую необходимо преобразовать в двоичное представление. Метод возвращает строку, представляющую двоичное значение, соответствующее входной строке Base64. Код метода представлен в листинге 2.9.

public static byte[] binaryStringToByteArray(String binaryString) {

int byteCount = (binaryString.length() + 7) / 8;

byte[] bytes = new byte[byteCount];

for (int i = 0; i < byteCount; i++) {

int startIndex = i \* 8;

int endIndex = Math.min(startIndex + 8, binaryString.length());

String byteString = binaryString.substring(startIndex, endIndex);

while (byteString.length() < 8) {

byteString = "0" + byteString;}

bytes[i] = (byte) Integer.parseInt(byteString, 2);}return bytes;}

Листинг 2.9 – Код метода для преобразования строки, закодированной в формате Base64, в двоичное представление

Метод getRandomBetween предназначен для генерации случайного числа в заданном диапазоне, определяемом минимальным и максимальным значениями. Код метода представлен в листинге 2.10.

public static byte[] binaryStringToByteArray(String binaryString) {

int byteCount = (binaryString.length() + 7) / 8;

byte[] bytes = new byte[byteCount];

for (int i = 0; i < byteCount; i++) {

int startIndex = i \* 8;

int endIndex = Math.min(startIndex + 8, binaryString.length());

String byteString = binaryString.substring(startIndex, endIndex);

while (byteString.length() < 8) {

byteString = "0" + byteString;

}

bytes[i] = (byte) Integer.parseInt(byteString, 2);

}

return bytes;}

Листинг 2.10 – Код метода для генерации случайного числа в заданном диапазоне

Метод generateCoprime предназначен для генерации случайного числа, которое является взаимно простым с заданным числом *n*. В качестве параметров принимает число *n*, с которым будет проверяться взаимная простота. Метод возвращает случайное число, которое является взаимно простым с *n*. Код метода представлен в листинге 2.11.

public static BigInteger generateCoprime(BigInteger n) {

BigInteger min = n.add(BigInteger.ONE);

BigInteger max = n.multiply(BigInteger.TWO);

BigInteger coprime;

Random random = new Random();

do {

coprime = getRandomBetween(min, max, random);

} while (!gcd(n, coprime).equals(BigInteger.ONE));

return coprime;

}

Листинг 2.11 – Код метода для генерации случайного взаимно простого числа

Метод getInverseNumber предназначен для вычисления обратного элемента числа number по модулю modulus. В качестве параметров принимает число, для которого нужно найти обратный элемент, модуль, относительно которого вычисляется обратный элемент. Метод возвращает обратное число number по модулю modulus. Если обратного элемента не существует, возвращается null. Код метода представлен в листинге 2.12.

public static BigInteger getInverseNumber(BigInteger number, BigInteger modulus) {

BigInteger m0 = modulus;

BigInteger y = BigInteger.ZERO;

BigInteger x = BigInteger.ONE;

if (modulus.equals(BigInteger.ONE)) {

return BigInteger.ZERO;

}

while (number.compareTo(BigInteger.ONE) > 0) {

BigInteger quotient = number.divide(modulus);

BigInteger temp = modulus;

modulus = number.mod(modulus);

number = temp; temp = y; y = x.subtract(quotient.multiply(y)); x = temp;

}

if (x.compareTo(BigInteger.ZERO) < 0) {

x = x.add(m0);

} return x;}

Листинг 2.12 – Код метода для генерации случайного взаимно простого числа

Метод gcd предназначен для вычисления наибольшего общего делителя (НОД) двух больших целых чисел *a* и *b*. Этот метод основан на алгоритме Евклида и используется в различных математических и криптографических приложениях. Принимает в качестве параметров *a* (первое целое число, для которого вычисляется НОД), *b* (второе целое число, для которого вычисляется НОД). Метод возвращает наибольший общий делитель двух чисел *a* и *b*. Код метода представлен в листинге 2.13.

public static BigInteger gcd(BigInteger a, BigInteger b) {

while (!b.equals(BigInteger.ZERO)) {

BigInteger temp = b;

b = a.mod(b);

a = temp;

}

return a;

}

Листинг 2.13 – Код метода для вычисления НОД

Метод base64Encode предназначен для кодирования строки в формат Base64. Это часто используется для безопасной передачи данных в текстовом формате, особенно в веб-технологиях и при работе с бинарными данными. В качестве параметров принимает строку, которую необходимо закодировать в формат Base64. Метод возвращает закодированную в Base64 строку. Код метода представлен в листинге 2.14.

public static String base64Encode(String text) {

return Base64.getEncoder().encodeToString(text.getBytes());

}

Листинг 2.14 – Код метода для кодировки строки в формат Base64

Константа BASE64\_CHARS содержит все символы, используемые в кодировке Base64. Она представляет собой строку, состоящую из 64 символов, которые используются для представления бинарных данных в текстовом формате. Код константы представлен в листинге 2.15.

static final String BASE64\_CHARS = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";

Листинг 2.15 – Код константы, которая содержит все символы, используемы в кодировке Base64

Далее, на основе сгенерированного приватного ключа подбираются параметры *n* и *a* для публичного ключа. Значение модуля *n* должно быть больше суммы всех чисел последовательности; кроме того, НОД (*а*, *n*) = 1. Метод generatePublicKey предназначен для генерации открытого ключа на основе предоставленного закрытого ключа и двух других параметров: *a* и *n*. Это может быть полезно в криптографических системах, где открытый и закрытый ключи связаны математически. Метод принимает в качестве параметров список значений закрытого ключа, который используется для вычисления открытого ключа, коэффициент, который используется в формуле для вычисления открытого ключа, модуль, который используется для вычисления открытого ключа по модулю. Метод возвращает список значений открытого ключа, вычисленных на основе закрытого ключа. Код константы представлен в листинге 2.16.

public static List<BigInteger> generatePublicKey(List<BigInteger> privateKey, BigInteger a, BigInteger n) {

List<BigInteger> sequence = new ArrayList<>();

for (BigInteger d : privateKey) {

BigInteger e = d.multiply(a).mod(n);

sequence.add(e);

}

return sequence;

}

Листинг 2.16 – Код метода для генерации публичного ключа

Результат генерации публичного и приватного ключей (для кодов ASCII) представлен на рисунке 2.1.

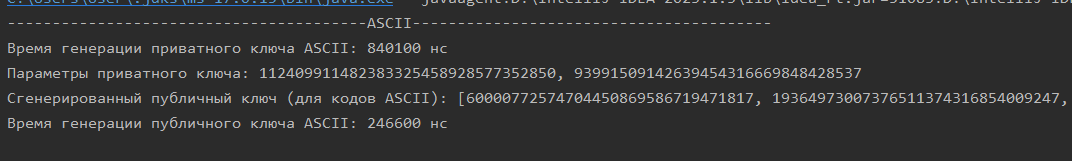


Рисунок 2.1 – Результат генерации ключей (для кодов ASCII)

Результат генерации публичного и приватного ключей (для Base64) представлен на рисунке 2.2.

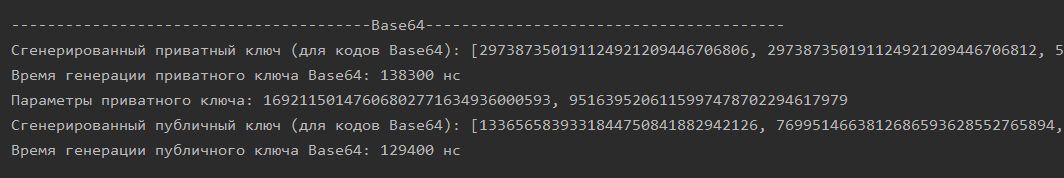


Рисунок 2.2 – Результат генерации ключей (для Base64)

Результат шифрования и расшифрования текста в кодах ASCII приведен на рисунке 2.3.

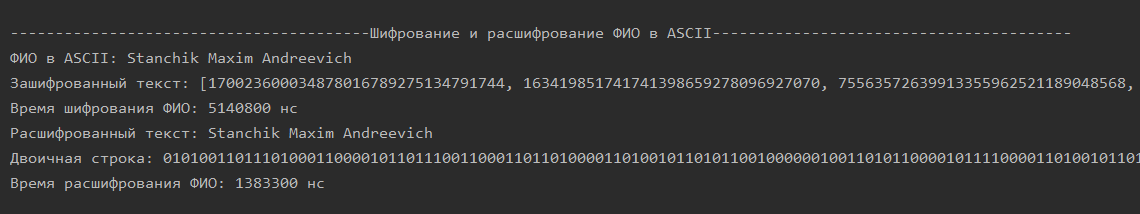


Рисунок 2.3 – Результат шифрования и расшифрования ФИО в ASCII

Результат шифрования и расшифрования текста в Base64 приведен на рисунке 2.4.

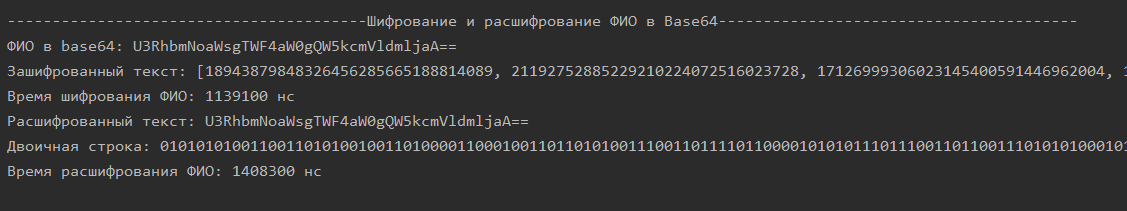


Рисунок 2.4 – Результат шифрования и расшифрования ФИО в Base64

# Время выполнения зашифрования и расшифрования

Оценку времени операций зашифрования и расшифрования при использовании разных таблиц кодировки можно увидеть на рисунке 3.1. Зашифровка ASCII занимает значительно больше времени (в среднем около 3–4 миллионов наносекунд). Время шифрования Base64 существенно меньше (в пределах 1–1.6 миллионов наносекунд). Время расшифровки ASCII варьируется от ~1 до 2 миллионов наносекунд. Время расшифровки Base64 также занимает примерно от ~0.6 до 1.5 миллионов наносекунд. Следовательно, Base64 выигрывает по производительности как в шифровании, так и в расшифровке по сравнению с ASCII. Это делает его предпочтительным выбором, если критично время обработки данных

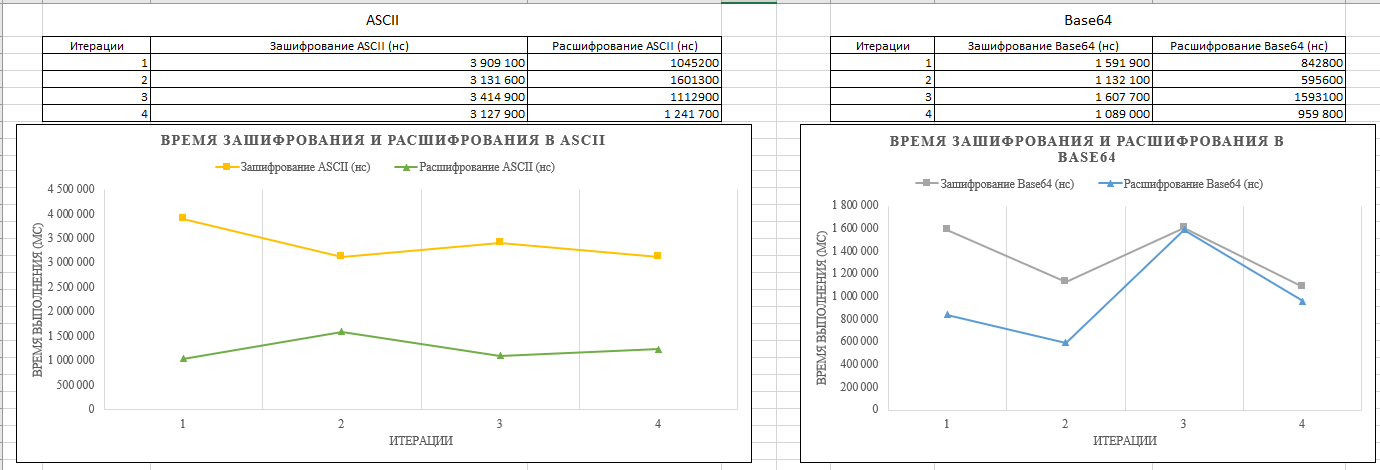


Рисунок 3.1 – Время зашифрования в ASCII и Base64

График, отображающий изменение времени зашифрования и расшифрования при разном количестве членов ключевой последовательности, представлен на рисунке 3.2.

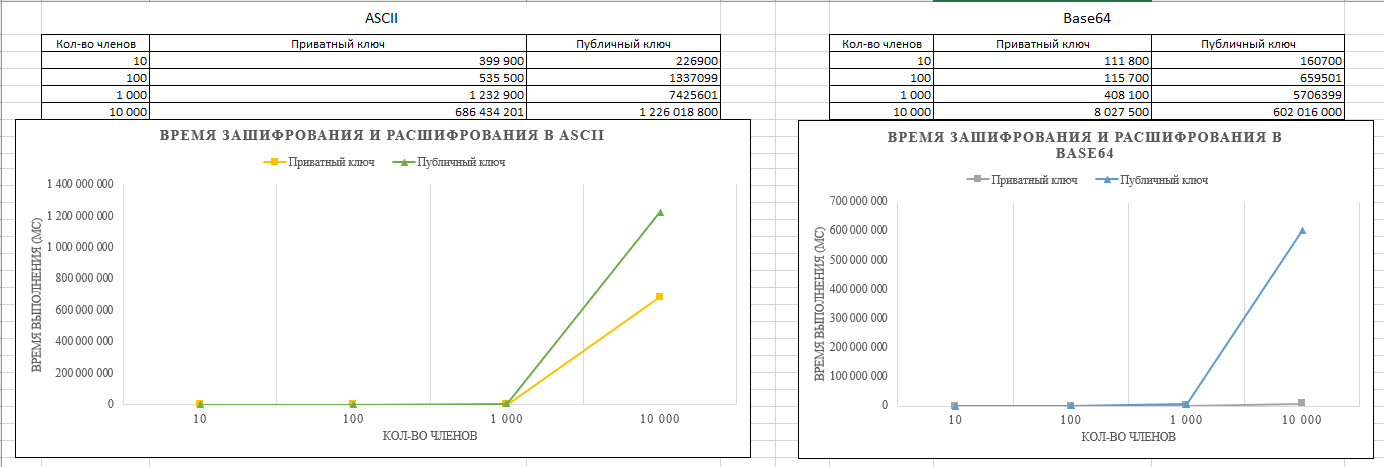


Рисунок 3.2 – Время зашифрования при разных количествах членов ключевой последовательности

Обе схемы (ASCII и Base64) показывают экспоненциальный рост времени выполнения операций при увеличении количества членов с 1 000 до 10 000. Особенно резко возрастает время публичного ключа, что видно как в ASCII, так и в Base64. При увеличении количества данных время работы публичного ключа растёт значительно быстрее, чем приватного. Формат Base64 снова показывает лучшую производительность, особенно с приватным ключом. В задачах, чувствительных к производительности, использование приватного ключа и Base64-кодирования предпочтительнее.

**Вывод:** в ходе выполнения лабораторной работы был изучен принцип реализации асимметричных шифров. Также было разработано приложение, выполняющее шифрование и расшифрование с помощью шифра, основанного на алгоритме об укладке ранца. Была оценена скорость шифрования и дешифрования сообщений, представленных в кодах ASCII и base64.