МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА — Российский технологический университет»

РТУ МИРЭА

Институт кибернетики
Базовая кафедра № 252 «Информационная безопасность»

КУРСОВАЯ РАБОТА

по дисциплине

«*Методы программирования*»

Тема курсовой работы

“Реализация ЭЦП на основе алгоритма *RSA* с хеш-функцией *SHA-2*”

Студент группы ККСО-03-20**,** Лытов Михаил Николаевич _____

Руководитель курсовой работы Чуваев Андрей Викторович _____
должность, звание, ученая степень (подпись руководителя)

Работа представлена к защите «8» декабря 2022 г.

Допущен к защите «_____» _____ 20__г.

Оценка_____

Москва, 2022

Оглавление

Введение

В данной курсовой работе будет рассмотрен алгоритм электронно-цифровой подписи RSA с хеш-функцией SHA-1.

Рональд Ривест, Ади Шамир, Леонард Адлеман в 1978 году предложили алгоритм, который имел интересные свойства для криптографии. На данном алгоритме была построена RSA.

Также в 1983 году руководство шифровальной службы Великобритании рассекретили документы, который глоссят, что в 1969 году был изобретён алгоритм, схожий с алгоритмом RSA, тремя ученными, а именно: Клиффорд Кокс, Малькольм Вильямсон и Джеймс Эллис, но их алгоритм отклонили, так как посчитали не безопасным.

RSA является первым широко используемым алгоритмом с ассиметричной криптографией и до сих пор актуальной, но с изменениями, которые будут рассмотрены в курсовой работе.

Так как RSA ассиметричный шифр, то возможно реализовать электронно-цифровую подпись, что будет показано в курсовой работе.

1. Теоретическая часть

1.1 Пример задачи

Пусть Алиса хочет передать посылку Бобу. Чтобы Боб мог быть уверен, что посылка от Алисы, она подписывает посылку. Далее рассмотрим алгоритм по пунктам.

1. Алиса создаёт два ключа, а именно открытый ключ (PK - public key) и закрытый ключ (SK - private key).
2. Открытый ключ Алиса отправляет Бобу.
3. Закрытым ключом Алиса подписывает посылку.
4. Алиса отправляет посылку с подписью.
5. Боб проверят закрытым ключом посылку Алисы.

1.2 Алгоритм RSA

1.2.1 Генерация ключей

Генерацию ключей в RSA можно разделить на 6 пунктов.

1.Генерация ключей в RSA зависит от двух случайно выбранных больших чисел p и q, для которых верна формула (1.2.1). В настоящий момент используют простые числа длинной 2048 бит

$$\log_2 p \sim \log_2 q > 1024 \quad (1.2.1)$$

2.Вычисление произведения простых чисел, формула (1.2.2):

$$n = pq \quad (1.2.2)$$

3.Вычисление функции Эйлера, по формуле (1.2.3):

$$\varphi(n) = (p-1)(q-1) \quad (1.2.3)$$

4.Выбрать случайное число e, взаимно простое с $\varphi(n)$ по формуле 1.2.5, где e – открытая экспонента.

$$[e \in \varphi(n) - 1] \quad (1.2.4)$$

$$\text{НОД}\big(e, \varphi(n)\big) = 1 \quad (1.2.5)$$

5.Вычислить число d по формуле китайской теоремы об остатках (1.2.6), где d – закрытая экспонента.

$$d * e = 1 \; mod \; \varphi(n), \tag{1.2.6}$$

$$d \equiv \; e^{-1} \; mod \; \varphi(n) \tag{1.2.7}$$

6.*Закрытым ключом* будет называть пару чисел (n, d), а *открытым ключом* пару (n, e).

1.2.3 Электронная подпись

Создание электронной подписи RSA производится так же, как и схема шифрования RSA, но должна соответствовать требованиям, а именно:

- вычисление подписи от сообщения является вычислительно лёгкой задачей;
- фальсификация подписи при неизвестном закрытом ключе – вычислительно трудная задача;
- подпись должна быть проверяемой открытым ключом.

Пусть Алиса имеет закрытый ключ $SK = (n, d)$, а Боб имеет открытый ключ $PK = (e, d)$.

1. Алиса вычисляет подпись сообщения m с помощью своего закрыто ключа по формуле (1.3.2), где s – подпись, d-закрытая экспонента, n-произведение простых чисел, а m – открытый текст.

$$m \in [1, n-1] \tag{1.3.1}$$

$$s = \; m^d \; \; mod(n) \tag{1.3.2}$$

2. Алиса посылает Бобу сообщение в виде (m, s).
3. Боб принимает сообщение, возводит подпись в степень e по модулю n. В результате Боб получает открытый текст.

$$s^e \; mod \; n = (m^d \; \; mod \; n)^e \; \; mod \; n = m \tag{1.3.3}$$

4. Боб сравнивает полученный текст с первой частью сообщения, если текст сошёлся, то подтверждена подлинность, в противном же случае это фальсификация.

Рисунок 1.3.1 – схема Электронной подписи.

Данный алгоритм имеет сильный недостаток, а именно длина подписи будет равна длине открытого сообщения.

Для уменьшения длинны подпись используется хеш-функция, обычно используют различные версии SHA.

Вместо подпись текста, пользователь подписывает хеш-функцию, заметим, что хеш-функция не добавляет безопасности, а именно уменьшает длину подписи, далее рассмотрим модифицированный вариант подписи, который представлен на рисунке 1.3.2.

1. Алиса посылает Бобу сообщение в виде (m, s), где

$$s = h(m)^d \ mod(n). \quad (1.3.4)$$

2. Боб принимает сообщение Алисы (m, s)
3. Вычисляет $h_2(m)$
4. Боб возводит подпись в степень e по модулю n. В результате Боб получает хеш-функцию.

$$s^e \ mod \ n = (h(m)^d \ mod \ n)^e \ mod \ n = h(m) \quad (1.3.5)$$

5. Боб сравнивает $h_2(m) = h(m)$, если хеш-функции сошлись, то подпись подлинна, в противном случае – фальсификация.

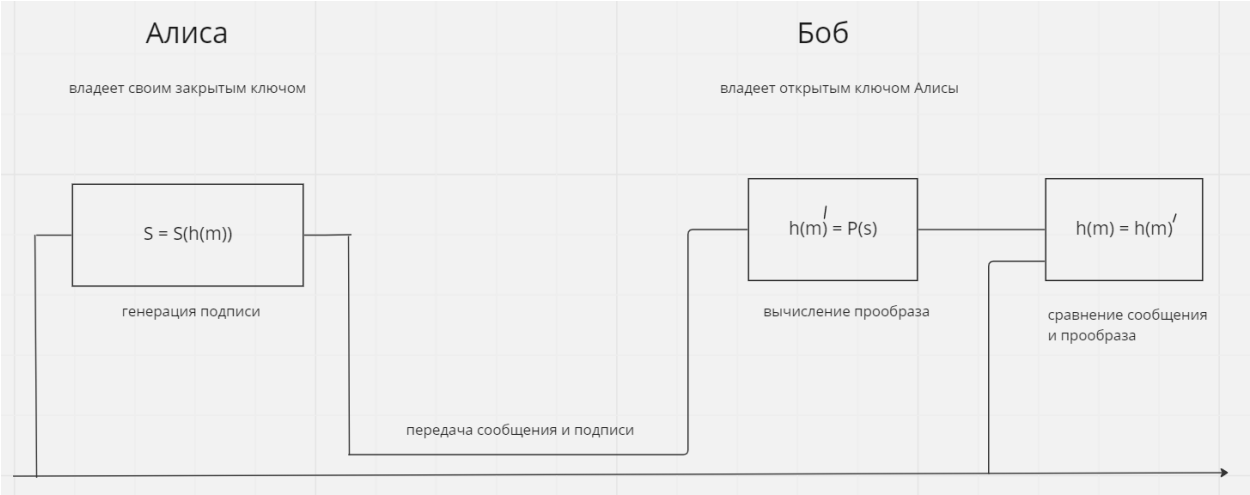

Рисунок 1.3.2 – схема Электронной подписи с хеш-функцией.

1.3 Алгоритм SHA – 2

1.3.1 Основные цели хеш-функции

Цели хеш-функции:

- Детерминировано шифровать данные, а то есть для одного и того же текста, хеш-функция всегда будет одинакова.
- Принимает ввод любой длинны, но вывод всегда одной длины.
- Невозможность получения вводных данных из вывода.

1.3.2 История SHA-2

Криптографический алгоритм SHA – 256 является частным случаем алгоритма семейства SHA - 2, был опубликован АНБ США в 2002 году, на данный момент алгоритм SHA – 2 не взломан, но в сети имеется несколько научных работ, которые утверждают, что алгоритм может быть взломан. На замену SHA – 2, в случае доказательства о ненадежности алгоритма, в 2015 году NIST утвердило алгоритм SHA -3 Keccak, являющимся принципиально другим алгоритмом.

SHA-3 хоть и является более стойким алгоритмом, но в то же время он является более долгим по времени работы.

Хеш-функции семейства SHA – 2 построена на основе структуры Меркла – Дамгарда.

1.3.2 Алгоритм SHA-256

Сначала постопают данные размером до 2^{64} бит, после чего дополняется нулями, пока сообщение не будет кратно 512 бит. Далее исходное сообщение делится на блоки(512 бит), в блоке 16 слов. Алгоритм пропускает каждый блок через цикл с 64 итерациями. На каждой итерации 2 слова преобразуются функцией, где функцию задают остальные слова. После чего все блоги складываются, и данная сумма является хеш-функцией.

Данную функцию нельзя ускорить параллельной обработкой блоков, так как блоки зависят от предыдущих блоков. Графическое представление обработки одной итерации представлено на рисунке 1.3.2.1.

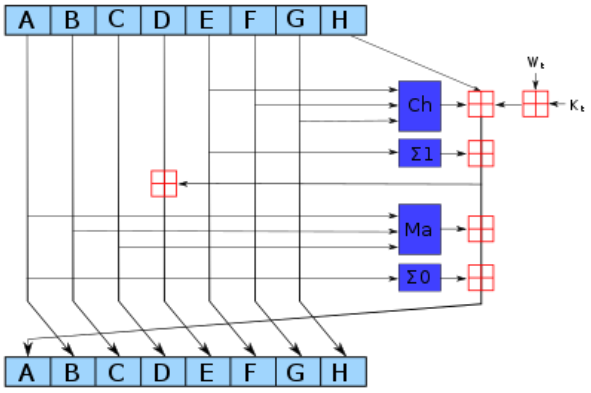

Рисунок 1.3.2.1 – Графическое представление обработки одной итерации.

2. Практическая часть

2.1 Реализация ЭЦП RSA

2.1.1 Используемые методы

В реализации кода используется класс *BigInteger* аналог целочисленных значений произвольной длины, при этом он потомок класса *Number*. *BigInteger* обладает множеством методов, таблица 2.1 показывает основные методы используемые в реализации кода.

Метод	свойства
add(BigInteger val)	This + val
divide(BigInteger val)	This/val
mod(BigInteger m)	This mod m
modPow(BigInteger exp, BigInteger m)	$this^{exp} \ mod\ m$
pow(BigInteget exp)	$this^{exp}$
probablePrime(int bitLength, Random rnd)	Возвращает простое число, которое вероятно, что простое
subtract(BigInteger val)	This - val

Таблица 2.1.1.1 – Методы класса *BigInteger*

2.1.2. Генерация ключей.

Реализацию генерации ключей можно поделить на 7 этапов.

1) Генерация двух простых чисел (листинг 2.1.1), размером 1024 бита или ~ 300 символом. Генерация проходит с помощью “Тест Миллера” и вероятность, что число будет составным не превышает 2^{-100}

```
first_number = probablePrime(1024,new Random());
second_number = probablePrime(1024, new Random());
```

Листинг 2.1.2.1 – Генерация простых чисел.

2) Вычисляется произведение простых чисел.

```
derivative = first_number.multiply(second_number);
```

Листинг 2.1.2.2 – $n = p * q$.

3) Вычисляется функция Эйлера.

```
BigInteger first_number_sub_1 =
first_number.subtract(BigInteger.ONE);//p-1
BigInteger second_number_sub_1 =
second_number.subtract(BigInteger.ONE);//q-1
euler_function =
first_number_sub_1.multiply(second_number_sub_1);//(p-1)(q-1)
```

Листинг 2.1.2.3 – Вычисление функции Эйлера.

4) Выбрать случайное число e, взаимно простое с $\varphi(n)$.

А) Сначала выбираем простое число равное 512 байтам.

Б) Если число взаимно просто с функцией Эйлера, то принимаем его за открытую экспоненту, в противном случае прибавляем единицу и возвращаемся к пункту Б.

```
open_exhibitor = BigInteger.probablePrime(maxleng / 2, new
Random());
while
(euler_function.gcd(open_exhibitor).compareTo(BigInteger.ONE) >
0 && open_exhibitor.compareTo(euler_function) < 0){
    open_exhibitor.add(BigInteger.ONE);
}
```

Листинг 2.1.2.4 – Поиск открытой экспоненты.

5) Вычислить число d такое, $d \equiv e^{-1} \, mod \, \varphi(n)$

```
close_exhibitor = open_exhibitor.modInverse(euler_function);
```

Листинг 2.1.2.5 – Выбор закрытой экспоненты.

6) Запись открытого ключа в файл open key.txt, где первая строчка открытая экспонента, а вторая строчка произведение простых чисел, пример записи показан на рисунке 2.1.2.7.

```
FileWriter output = new FileWriter("open key.txt");
String open_exhib = open_exhibitor.toString();
String der = derivative.toString();
output.write("open exhibitor:" + open_exhib + "\nderivative:" +
                    der);
output.close();
```

Листинг 2.1.2.6 – Запись открытого ключа.

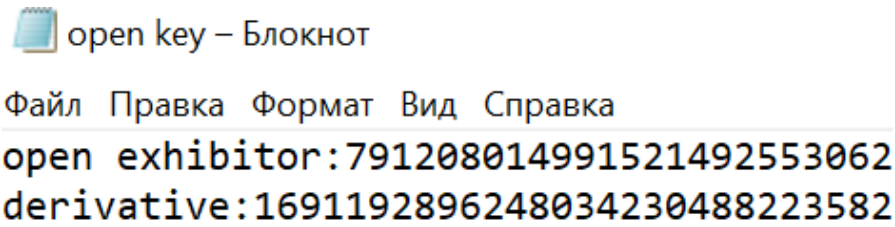

Рисунок 2.1.2.7 – Запись открытого ключа.

7) Запись закрытого ключа в файл *close key.txt*, где первая строчка закрытая экспонента, а вторая строчка произведение простых чисел, пример записи показан на рисунке 2.1.2.9.

```
FileWriter output = new FileWriter("close key.txt");
String close_exhib = close_exhibitor.toString();
String der = derivative.toString();
output.write("close exhibitor:" + close_exhib + "\nderivative:"
                    + der);
output.close();
```

Листинг 2.1.2.8 – Запись закрытого ключа.

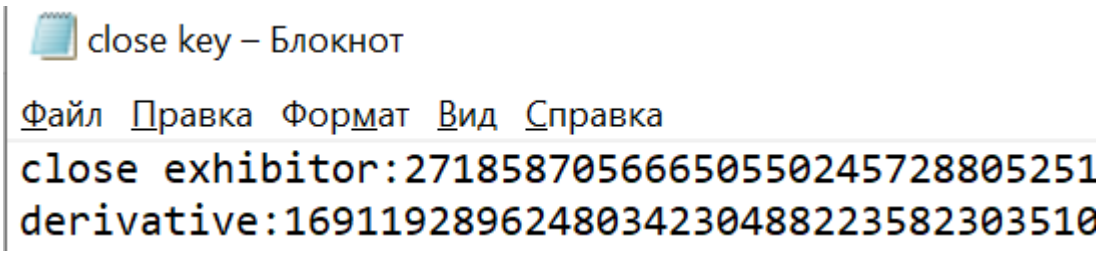

Рисунок 2.1.2.1 – Запись закрытого ключа.

Среднее время генерации ключей составляет 72 миллисекунды, данные получены при 100 экспериментов, часть данных показан на рисунке 2.1.10.

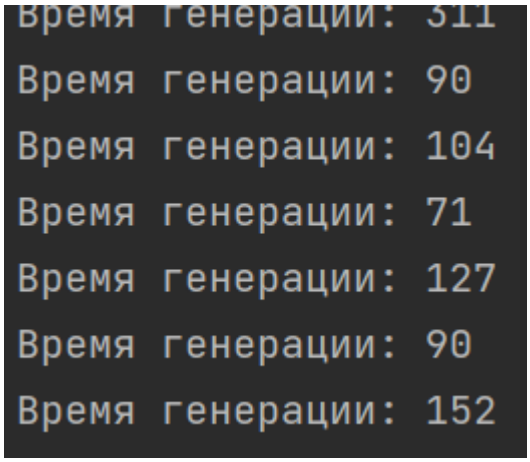

Рисунок 2.1.2.2 – Время генерации паролей.

2.1.3 Подпись

Подпись файла можно разделить на 4 пункта

1) Считываем файл с текстом построчно.

```
String line ;
String line_sum = "";
FileReader file_txt_ = new FileReader(name_file_txt);
BufferedReader buf_txt = new BufferedReader(file_txt_);
while ((line = buf_txt.readLine()) != null){
    line_sum += line + "\n";
}
file_txt_.close();
buf_txt.close();
```

Листинг 2.1.3.1 – Считывание текстового файла.

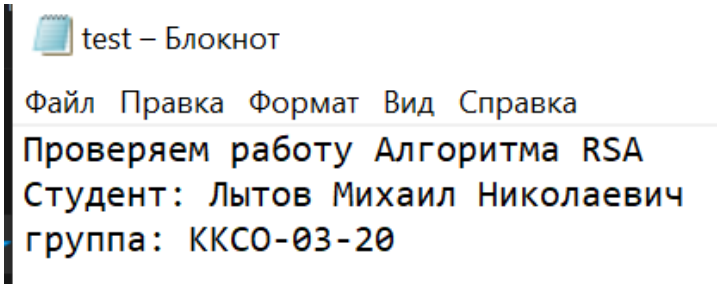

Рисунок 2.1. 3.1 – Пример подписываемого файла.

2) Берём хеш-функцию сообщения.

```
SHA sha = new SHA (line_sum);
BigInteger sha_int = sha.getInteger();
```

Листинг 2.1. 3.3 – взятие хеш-функции входного сообщения.

3) Считываем файл с закрытым ключом

А). Первая строчка файле является закрытой экспонентой, вторая строчка является произведением, что показано на рисунке 2.1.2.1

```
FileReader file_txt = new FileReader(name_file_close_key);
BufferedReader buf = new BufferedReader(file_txt);
String line_1 = buf.readLine();
String line_2 = buf.readLine();
line_1 = line_1.substring(16);
line_2 = line_2.substring(11);
close_key = new BigInteger(line_1);
derivative = new BigInteger(line_2);
buf.close();
file_txt.close();
```

Листинг 2.1.3.4 – Считывание файла с закрытым ключём.

4) Создаём подпись и добавляем в конец файла с открытым текстом.

А) Если хеш-функция меньше произведения простых чисел, то создаём подпись по формуле (1.3.2), в противном случае выводим, что ключ мал.

```
if(sha_int.compareTo(derivative) <= -1){
    signature = sha_int.modPow(close_key,derivative);
    FileWriter file_signature = new FileWriter(name_file_txt);
    file_signature.write(line_sum + "\n" +
                                    signature.toString());
    file_signature.close();

}else{System.out.println("Ключ мал");}
```

Листинг 2.1.3.5 – Подпись входного сообщения.

Сообщение может быть подписано за 80 миллисекунд, что показано на рисунке 2.1.3.3, подпись расположена на последней строчке в файле, что показано на рисунке 2.1.3.2

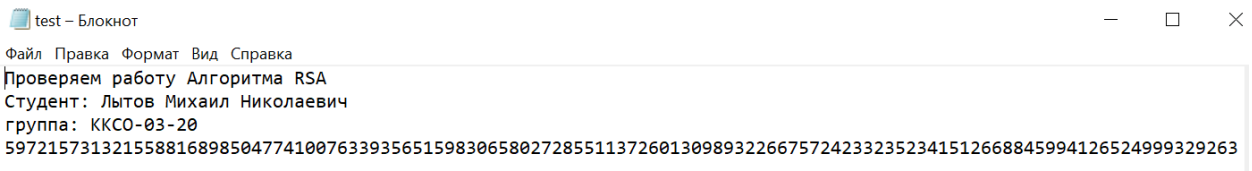

Рисунок 2.1.3.2 – Подписанное сообщение.

Рисунок 2. 1.3.3 – Время подписание файла.

2.1.4 Проверка подписи

Проверка подписи происходит в 5 этапов.

1) Считываем файл с открытым ключом, пример открытого ключа на рисунке 2.1.2.1

А) Первая строчка в файле является открытой экспонентой, вторая строчка в файле является произведением простых чисел.

```
FileReader file_open_key = new FileReader(name_file_open_key);
BufferedReader buf = new BufferedReader(file_open_key);
String open_exhibitor = buf.readLine();
String derivative = buf.readLine();
open_exhibitor = open_exhibitor.substring(15);
derivative = derivative.substring(11);
this.open_exhibitor = new  BigInteger(open_exhibitor);
this.derivative = new BigInteger(derivative);
```

Листинг 2.1.4.1 – Считывание открытого ключа.

2) Считываем подписываемый файл

А) Считываем файл до того момента пока не встретим null

Б) последняя считанная строчка является подписью, остальные же строчки являются сообщением, как показано на рисунке 2.1.3.2

```
FileReader read_signature_file = new
FileReader(name_file_signature_file);
BufferedReader buf_signature_file = new
BufferedReader(read_signature_file);
while ((line = buf_signature_file.readLine()) != null){
    message += line + "\n";
    signature = line;
}
message = message.substring(0, message.length() -
              signature.length() - 2);
this.signature = new BigInteger(signature);
```

Листинг 2. 1.4.2 – Считывание подписанного файла.

3) Берем хеш-функцию из полученного сообщения.

```
SHACustom sha = new SHACustom(message);
BigInteger sha_int = sha.toBiginteger_SHA();
```

Листинг 2.1.4.3 – Взятие хеш-функции сообщения.

4) Возводим подпись в степень *e* по модулю *n* и получаем хеш-функцию из подписи.

```
if ((sha_int.compareTo(derivative)) <= -1){
    check = signature.modPow(open_exhibitor, derivative);
```

Листинг 2. 1.4.4 – Открытие подписи.

5) Сравниваем хеш-функцию из сообщения и из подписи, если они равны, то подпись подлинна, что покажет сообщение, пример на рисунке 2.1.4.1, в противном случае фальсификация.

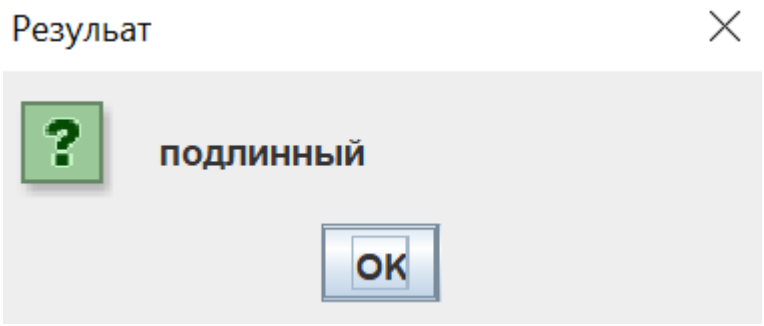

Рисунок 2.4.1 – Сообщение об подлинности подписи.

```
if((check.compareTo(sha_int)) == 0){
```

Листинг 2. 1.4.5 – Проверка на оригинальность подписи.

6) В исходный файл записывается сообщение, пример полученного файла на рисунке 2. 1.4.2.

```
FileWriter file = new FileWriter(name_file_signature_file);
file.write(message);
file.close();
```

Листинг 2. 1.4.6 –Запись исходного сообщения.

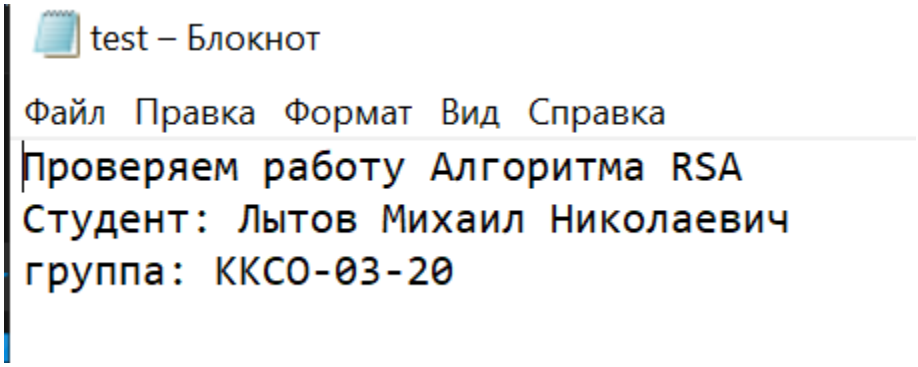

Рисунок 2. 1.4.2 – Исходный файл.

Время проверки подписи составляет 8 миллисекунд, что показывает рисунок 2.4.3.

```
Время проверки подписи 8 миллисекунд
```

Рисунок 2. 1.4.3 – Время проверки подписи.

2.2 Реализация SHA-2

2.2.1 Преобразования строки в двоичный код.

Преобразуем строку в двоичный код.

```
private byte[] String_in_Bytes(String text){
    byte[] byte_text = text.getBytes();
    return byte_text;
}
```

Листинг 2.2.1.1 – преобразование строки в массив байт

2.2.2 Отмечаем конец входных данных

После входных данных добавляем 0b10000000, показывая этим, что данные записаны.

```
private byte[] add_one(byte[] byte_text){
    byte_text = Arrays.copyOf(byte_text, byte_text.length + 1);
    byte_text[byte_text.length - 1] = (byte) 0b10000000;
    return byte_text;
}
```

Листинг 2.2.2.1 – добавления 0b10000000.

2.2.3 Дополнения кода

Дополняем код нулями, пока данные не будут сравнимы с 512, после чего в последние 64 бита будет записана длинна исходного сообщения и определяем количество блоков.

```
    long size_array = byte_text.length; // для изменения размера
массива
    long length_source_array = byte_text.length; // длина
входного массива
    while (size_array % 64 != 0){
        byte_text = Arrays.copyOf(byte_text, byte_text.length +
1);
        byte_text[byte_text.length - 1] = (byte) 0b00000000;
        size_array = byte_text.length;

    }
```

Листинг 2.2.3.1 – дополнения нулями.

```
byte[] line = new byte[8];
ByteBuffer line_array = ByteBuffer.allocate(line.length);
line_array.putLong((length_source_array - 1) * 8);
line = line_array.array();

for (int i = 0; i < line.length; i++){
    byte_text[Math.toIntExact(size_array - 1 - i)] =
line[line.length - 1 - i];
}
```

Листинг 2.2.3.2 – дополнение в конец длину исходных данных.

```
number_blocks = (int) (size_array / 64);
```

Листинг 2.2.3.3 –количество блоков.

2.2.4 Инициализация констант

Инициализируются 8 констант, которые представляют из себя первые 32 бита дробной части первых восьми простых чисел.

```
private int h0 = 0x6a09e667;
private int h1 = 0xbb67ae85;
private int h2 = 0x3c6ef372;
```

```
private int h3 = 0xa54ff53a;
private int h4 = 0x510e527f;
private int h5 = 0x9b05688c;
private int h6 = 0x1f83d9ab;
private int h7 = 0x5be0cd19;
```

Листинг 2.2.3.1 – дополнения нулями

Создаем 64 константы, где каждое значение представляет из себя 32 бита дробной части кубических корней первых 64 простых чисел.

```
private static int[] ARR_CONST =
{0x428a2f98,0x71374491,0xb5c0fbcf,0xe9b5dba5,0x3956c25b
,0x59f111f1,0x923f82a4,0xab1c5ed5,

0xd807aa98,0x12835b01,0x243185be,0x550c7dc3,0x72be5d74,
0x80deb1fe,0x9bdc06a7,0xc19bf174,

0xe49b69c1,0xefbe4786,0x0fc19dc6,0x240ca1cc,0x2de92c6f,
0x4a7484aa,0x5cb0a9dc,0x76f988da,

0x983e5152,0xa831c66d,0xb00327c8,0xbf597fc7,0xc6e00bf3,
0xd5a79147,0x06ca6351,0x14292967,

0x27b70a85,0x2e1b2138,0x4d2c6dfc,0x53380d13,0x650a7354,
0x766a0abb,0x81c2c92e,0x92722c85,

0xa2bfe8a1,0xa81a664b,0xc24b8b70,0xc76c51a3,0xd192e819,
0xd6990624,0xf40e3585,0x106aa070,

0x19a4c116,0x1e376c08,0x2748774c,0x34b0bcb5,0x391c0cb3,
0x4ed8aa4a,0x5b9cca4f,0x682e6ff3,

0x748f82ee,0x78a5636f,0x84c87814,0x8cc70208,0x90befffa,
0xa4506ceb,0xbef9a3f7,0xc67178f2};
```

Листинг 2.2.4.1 – дополнения нулями

2.2.5 Цикл фрагментов

Следующие шаги будут выполнятся для каждого блока

```
for (int i = 0; i < number_blocks; i++) {
```

Листинг 2.2.5.1 – Обработка каждого блока.

2.2.6 Создание слов

Копируем данные из 2.2.1 в новый массив по 32 бита, где 32 бита являются словом.

```
private int[] words(byte[] byte_text){
    int[] array_int = new int[16];
    for (int i = 0; i < 16; i++) {
        byte[] byte_word = Arrays.copyOfRange(byte_text, i * 4,
4 + i * 4);
        array_int[i] = ByteBuffer.wrap(byte_word).getInt();
    }
    return array_int;
}
```

Листинг 2.2.6.1 – создание слов.

Добавляем нулевые слова так, чтобы получилось 64 слова в итоге.

```
private int[] add_words_zero(int[] array_int){
    int size = array_int.length + 1;
    array_int = Arrays.copyOf(array_int, 64);
    Arrays.fill(array_int, size, 64, 0);

    return  array_int;
}
```

Листинг 2.2.6.2 – инициализация нулями.

2.2.7 Сжатие

Инициализируем копии h_0 ... h_1

```
int a = h0;
int b = h1;
int c = h2;
int d = h3;
int e = h4;
int f = h5;
int g = h6;
int h = h7;
```

Листинг 2.2.7.1 – копии h_0 ... h_1.

Далее проведем 64 итерации для копий, где каждую итерацию будут манятся переменные a-h

Сжатие происходит по формуле 2.2.7.1.

$$a = T_1 + T_2 \, mod \, 2^{32}$$

$$b = a$$

$$c = b$$

$$d = c$$

$$e = d + T_1$$

$$g = f$$

$$h = g$$

$$T_1 = h + S_1 + ch + k[i] + w[i] \; mod \; 2^{32}$$

$$T_2 = S_0 + Ma \; mod \; 2^{32}$$

$$S_1 = (e \; rotateRight \; 6) \oplus (e \; rotateRight \; 11) \oplus (e \gg 25)$$

$$ch = (e \; \& \; f) \oplus (\bar{e} \; \& \; g)$$

$$S_0 = (a \; rotateRight \; 2) \oplus (a \; rotateRight \; 13) \oplus (a \gg 22)$$

$$Ma = (a \; \& b) \oplus (a \; \& \; c) \oplus (b \; \& \; c)$$

(2.2.7.1)

```
for(int j = 0; j < 64; j++){
    int Sum_1 = Integer.rotateRight(e, 6) ^ Integer.rotateRight(e, 11) ^
Integer.rotateRight(e,25);
    int Ch = (e & f) ^ ((~e) & g );
    int T_1 = (int) (h + Sum_1 + Ch + ARR_CONST[j] + array_int[j] % twe_32);
    int Sum_0 = Integer.rotateRight(a, 2) ^ Integer.rotateRight(a, 13) ^
Integer.rotateRight(a, 22);
    int Ma = (a & b) ^ (a & c) ^ (b & c);
    int T_2 = (int) (Sum_0 + Ma % twe_32);
    h = g;
    g = f;
    f = e;
    e = (int) (d + T_1 % twe_32);
    d = c;
    c = b;
    b = a;
    a = (int) (T_1 +T_2 % twe_32);

}
```

Листинг 2.2.7.1 – реализация формулы 2.2.7.1.

2.2.8 Hex

Соединяем в одну совокупность h_0 ... h_7

```
public String gethex(){
    String hex = Integer.toHexString(h0) + Integer.toHexString(h1) +
Integer.toHexString(h2) + Integer.toHexString(h3) + Integer.toHexString(h4) +
Integer.toHexString(h5) + Integer.toHexString(h6) + Integer.toHexString(h7);
    return hex;
}
```

Листинг 2.2.8.1 – hexsum .

3. Список Литературы

1. Криптографические методы защиты информации, 3-е издание - Владимиров Сергей Михайлович
2. Java Полное руководство (Десятое издание) – Герберт Шилдт
3. https://habr.com/ru/post/470159/
4. https://javarush.com/groups/posts/2709-biginteger-i-bigdecimal
5. https://docs.oracle.com/en/java/javase/18/docs/api/java.base/java/math/BigInteger.html
6. https://codegym.cc/groups/posts/java-biginteger-class
7. https://en.wikipedia.org/wiki/RSA_(cryptosystem)#mw-head
8. https://habr.com/ru/post/534014/
9. https://habr.com/ru/company/selectel/blog/530262/
10. https://habr.com/ru/post/75193/
11. https://habr.com/ru/company/virgilsecurity/blog/459370/
12. https://habr.com/ru/post/224523/
13. https://www.iso.org/standard/17658.html
14. https://docs.oracle.com/en/java/javase/17/security/java-security-overview1.html
15. https://docs.oracle.com/javase/8/docs/api/java/security/MessageDigest.html
16. https://russianblogs.com/article/3080478024/
17. https://russianblogs.com/article/60561293587/
18. https://habr.com/ru/company/selectel/blog/530262/

4. Заключение

В ходе написание курсовой работы был реализован алгоритм электронной подписи RSA на языке JAVA. Можно сказать, что алгоритм RSA хоть и был изобретён давно, но до сих пор является актуальным.
Одним из главных недостатков RSA и в тот же момент преимуществом является то, чтобы сохранять безопасность алгоритма, стоит иметь большие входные данные, на данный момент используются простые числа размерностью 2048 бит, но в будущем придётся использовать ещё более большую длину простых чисел, что даст возможность алгоритму работать безопасно, но увеличит время генерации ключей.

5. Приложение

```
6. public class main {
       public static void main(String[] args){
           Frame app = new Frame();
           app.setVisible(true);
       }

   }
```

Листинг 5.1 – main.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.math.BigInteger;

public class Frame extends JFrame {
    private JLabel key_generation;
    private JButton button_generation;
    private JLabel file_generation;
    private JLabel emptiness;
    private String path_text_file;
    private String path_close_file_file;
    private String path_open_key;
    private String path_text_signature;
    public Frame() {
        super("RSA");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(3,4,2,2));

        key_generation = new JLabel("генерация ключей");
        button_generation = new JButton("генерировать");
        file_generation = new JLabel("");
        emptiness = new JLabel("");

        panel.add(key_generation);
        button_generation.addActionListener(new file_generation());
        panel.add(file_generation);
        panel.add(emptiness);
        panel.add(button_generation);



        JLabel signature = new JLabel("Подпись");
        JButton text_file = new JButton("Подписываемый файл");
        JButton close_key_file = new JButton("Закрытый ключ");
        JButton button_signature = new JButton("Подписать");

        text_file.addActionListener(new text_file());
        close_key_file.addActionListener(new close_key_file());
        button_signature.addActionListener(new sigianure_file());
        panel.add(signature);
```

```
        panel.add(text_file);
        panel.add(close_key_file);
        panel.add(button_signature);

        JLabel signature_verification_label = new JLabel("Проверка подписи");
        JButton text_signature_file = new JButton("Подписанный файл");
        JButton open_key_file = new JButton("Открытый ключ");
        JButton signature_verification_button  =new JButton("проверить");

        panel.add(signature_verification_label);
        panel.add(text_signature_file);
        panel.add(open_key_file);
        panel.add(signature_verification_button);

        signature_verification_button.addActionListener(new
signature verification());
        text_signature_file.addActionListener(new text_signature_verification
());
        open_key_file.addActionListener(new open_key());

        getContentPane().add(panel);
        setPreferredSize(new Dimension(800, 300));
        pack();
        setLocationRelativeTo(null);
        setVisible(true);
    }
    class signature_verification implements  ActionListener{

        @Override
        public void actionPerformed(ActionEvent e) {
            if(path_open_key == null || path_text_signature == null){
                if(path_open_key == null){
                    JOptionPane.showConfirmDialog(null, "Не выбран файл с
открытым ключём", "ERROR", JOptionPane.PLAIN_MESSAGE);
                } else if (path_text_signature == null) {
                    JOptionPane.showConfirmDialog(null, "Не выбран файл,
который нужно проверить", "ERROR", JOptionPane.PLAIN_MESSAGE);
                }
            }else {
                try {
                    FileReader file_open_key = new FileReader(path_open_key);
                    BufferedReader buf = new BufferedReader(file_open_key);
                    String open_exhibitor = buf.readLine();
                    open_exhibitor = open_exhibitor.substring(0,15);
                    if(open_exhibitor.equals("open exhibitor:")) {
                        Long start = System.currentTimeMillis();
                        Signature_verification check = new
Signature_verification(path_text_signature, path_open_key);
                        Long finish = System.currentTimeMillis();

                        System.out.println("Время проверки подписи " +
(finish - start) + " миллисекунд");

                        String text_check_file = check.check();
                        JOptionPane.showConfirmDialog(null, text_check_file,
"Результат", JOptionPane.PLAIN_MESSAGE);
                    }else{
                        JOptionPane.showConfirmDialog(null, "Не выбран файл,
с открытым ключём", "ERROR", JOptionPane.PLAIN_MESSAGE);
                    }
                } catch (IOException ex) {
                    throw new RuntimeException(ex);
```

```
                }
            }


        }
    }
    class text_signature_verification implements  ActionListener{

        @Override
        public void actionPerformed(ActionEvent e) {
            JFileChooser fileopen = new JFileChooser();
            int ret = fileopen.showDialog(null, "Открыть файл");
            if(ret == JFileChooser.APPROVE_OPTION){
                File file = fileopen.getSelectedFile();
                try {
                    path text signature = file.getCanonicalPath();
                } catch (IOException ex) {
                    throw new RuntimeException(ex);
                }
            }

        }
    }

    class open_key implements ActionListener{

        @Override
        public void actionPerformed(ActionEvent e) {
            JFileChooser fileopen = new JFileChooser();
            int ret = fileopen.showDialog(null, "Открыть файл");
            if(ret == JFileChooser.APPROVE_OPTION){
                File file = fileopen.getSelectedFile();
                try {
                    path_open_key = file.getCanonicalPath();
                } catch (IOException ex) {
                    throw new RuntimeException(ex);
                }
            }
        }
    }

    class close_key_file implements ActionListener{

        @Override
        public void actionPerformed(ActionEvent e) {
            JFileChooser fileopen = new JFileChooser();
            int ret = fileopen.showDialog(null, "Открыть файл");
            if (ret == JFileChooser.APPROVE_OPTION) {
                File file = fileopen.getSelectedFile();
                //System.out.println("тут");
                try {
                    //System.out.println("теперь тут");
                    path_close_file_file = file.getCanonicalPath();
                    //System.out.println(a);
                } catch (IOException ex) {
                    throw new RuntimeException(ex);
                }
            }
            //System.out.println(path_text_file);

        }
    }
```

```
    class sigianure_file implements ActionListener{

        @Override
        public void actionPerformed(ActionEvent e) {
            if(path_text_file == null || path_close_file_file == null) {
                if (path_text_file == null) {
                    JOptionPane.showConfirmDialog(null, "Не выбран файл,
который нужно подписать", "ERROR", JOptionPane.PLAIN_MESSAGE);
                } else if (path_close_file_file == null) {
                    JOptionPane.showConfirmDialog(null, "Не выбран файл, с
закрытым ключём", "ERROR", JOptionPane.PLAIN_MESSAGE);

                }
            }else{
                try {
                    FileReader file_txt = new
FileReader(path_close_file_file);
                    BufferedReader buf = new BufferedReader(file_txt);
                    String line_1 = buf.readLine();
                    try {

                        if (line_1.substring(0, 16).equals("close
exhibitor:")) {
                            buf.close();
                            file_txt.close();
                            Long start = System.currentTimeMillis();
                            Signature signature_class = new
Signature(path_text_file, path_close_file_file);
                            Long finish = System.currentTimeMillis();
                            System.out.println("Время подписи: " + (finish -
start) + " миллисекунд");
                            JOptionPane.showConfirmDialog(null, "Файл
подписан", "Резульат операции", JOptionPane.PLAIN_MESSAGE);
                        } else {
                            JOptionPane.showConfirmDialog(null, "Некоректный
файл с закрытым ключём", "Error", JOptionPane.PLAIN_MESSAGE);
                        }
                    }catch (StringIndexOutOfBoundsException ex){
                        JOptionPane.showConfirmDialog(null, "Некоректный файл
с закрытым ключём", "Error", JOptionPane.PLAIN_MESSAGE);
                    }
                } catch (IOException ex) {
                    throw new RuntimeException(ex);
                }
            }
        }
    }
    class text_file implements ActionListener{

        @Override
        public void actionPerformed(ActionEvent e) {
            JFileChooser fileopen = new JFileChooser();
            int ret = fileopen.showDialog(null, "Открыть файл");
            if (ret == JFileChooser.APPROVE_OPTION) {
                File file = fileopen.getSelectedFile();
                //System.out.println("тут");
                try {
                    //System.out.println("теперь тут");
                    path_text_file = file.getCanonicalPath();
                    //System.out.println(a);

                } catch (IOException ex) {
```

```
                    throw new RuntimeException(ex);
                }
            }
            //System.out.println(path_close_file_file);

        }
    }
    class file_generation implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            long start;
            long finish;
            start = System.currentTimeMillis();
            key_generation key = new key_generation();
            finish = System.currentTimeMillis();

            System.out.println("Время генерации: " + (finish - start));

            BigInteger open_exhibitor = key.getOpen_exhibitor();
            BigInteger close_exhibitor = key.getClose_exhibitor();
            BigInteger derivative = key.getDerivative();
            BigInteger first_number = key.getFirst_number();
            BigInteger second_number = key.getSecond_number();
            try {
                key_output output = new key_output(open_exhibitor,
close_exhibitor, derivative, first_number, second_number);
                String message = "адресс открытого ключа: " +
output.path_open_key() + "\nадрес закрытого ключа: " +
output.path_close_key();
                JOptionPane.showConfirmDialog(null,message, "Расположение
ключей", JOptionPane.PLAIN_MESSAGE);
            } catch (IOException ex) {
                throw new RuntimeException(ex);
            }


            //OptionPane.showConfirmDialog(null,"a", "out",
JOptionPane.PLAIN_MESSAGE);

        }
    }



}
```

Листинг 5.2 – Frame.java

```
import java.math.BigInteger;
import java.nio.ByteBuffer;
import java.util.Arrays;

public class SHA {
    private int number_blocks = 0;
    private int h0 = 0x6a09e667;
    private int h1 = 0xbb67ae85;
    private int h2 = 0x3c6ef372;
    private int h3 = 0xa54ff53a;
    private int h4 = 0x510e527f;
    private int h5 = 0x9b05688c;
    private int h6 = 0x1f83d9ab;
```

```
    private int h7 = 0x5be0cd19;
    private static int[] ARR_CONST =
{0x428a2f98,0x71374491,0xb5c0fbcf,0xe9b5dba5,0x3956c25b,0x59f111f1,0x923f82a4
,0xab1c5ed5,

0xd807aa98,0x12835b01,0x243185be,0x550c7dc3,0x72be5d74,0x80deb1fe,0x9bdc06a7,
0xc19bf174,

0xe49b69c1,0xefbe4786,0x0fc19dc6,0x240ca1cc,0x2de92c6f,0x4a7484aa,0x5cb0a9dc,
0x76f988da,

0x983e5152,0xa831c66d,0xb00327c8,0xbf597fc7,0xc6e00bf3,0xd5a79147,0x06ca6351,
0x14292967,

0x27b70a85,0x2e1b2138,0x4d2c6dfc,0x53380d13,0x650a7354,0x766a0abb,0x81c2c92e,
0x92722c85,

0xa2bfe8a1,0xa81a664b,0xc24b8b70,0xc76c51a3,0xd192e819,0xd6990624,0xf40e3585,
0x106aa070,

0x19a4c116,0x1e376c08,0x2748774c,0x34b0bcb5,0x391c0cb3,0x4ed8aa4a,0x5b9cca4f,
0x682e6ff3,

0x748f82ee,0x78a5636f,0x84c87814,0x8cc70208,0x90befffa,0xa4506ceb,0xbef9a3f7,
0xc67178f2};
    public SHA(String text){

        byte[] byte_text = String_in_Bytes(text);

        byte_text = add_one(byte_text);

        byte_text = add_zero(byte_text);

        for (int i = 0; i < number_blocks; i++) {
            int[] array_int  = words(Arrays.copyOfRange(byte_text, i * 64, 64
+ i * 64));
            //System.out.println(Arrays.toString(array_int));
            array_int = add_words_zero(array_int);
            array_int = changing_zero_indexes(array_int);
            compression(array_int);

        }

    }
    private byte[] String_in_Bytes(String text){
        byte[] byte_text = text.getBytes();
        return byte_text;
    }
    private byte[] add_one(byte[] byte_text){
        byte_text = Arrays.copyOf(byte_text, byte_text.length + 1);
        byte_text[byte_text.length - 1] = (byte) 0b10000000;
        return byte_text;
    }
    private byte[] add_zero(byte[] byte_text){
        long size_array = byte_text.length; // для изменения размера массива
        long length source array = byte text.length; // длина входного
массива
        while (size_array % 64 != 0){
            byte_text = Arrays.copyOf(byte_text, byte_text.length + 1);
            byte_text[byte_text.length - 1] = (byte) 0b00000000;
            size_array = byte_text.length;
```

```
        }
        byte[] line = new byte[8];
        ByteBuffer line_array = ByteBuffer.allocate(line.length);
        line_array.putLong((length_source_array - 1) * 8);
        line = line_array.array();
        //System.out.println(Arrays.toString(line));
        for (int i = 0; i < line.length; i++){
            byte_text[Math.toIntExact(size_array - 1 - i)] = line[line.length
- 1 - i];
        }

        number_blocks = (int) (size_array / 64);
        return byte_text;

    }
    private int[] words(byte[] byte_text){
        int[] array_int = new int[16];
        for (int i = 0; i < 16; i++) {
            byte[] byte_word = Arrays.copyOfRange(byte_text, i * 4, 4 + i *
4);
            array_int[i] = ByteBuffer.wrap(byte_word).getInt();
        }
        return array_int;
    }
    private int[] add_words_zero(int[] array_int){
        int size = array_int.length + 1;
        array_int = Arrays.copyOf(array_int, 64);
        Arrays.fill(array_int, size, 64, 0);

        return  array_int;
    }
    private int[] changing_zero_indexes(int[] array_int){
        long twe_32 = 4294967296l;

        for(int j = 16; j < 64; j++) {
            long s_0 = Integer.rotateRight(array_int[j-15], 7) ^
Integer.rotateRight(array_int[j-15], 18) ^ (array_int[j-15] >>> 3);
            long s_1 = Integer.rotateRight(array_int[j - 2], 17) ^
Integer.rotateRight(array_int[j - 2], 19) ^ (array_int[j - 2] >>> 10);
            long result = (array_int[j-16] + s_0 + array_int[j - 7] + s_1) %
twe_32;
            array_int[j] = (int) result;
        }
        return array_int;
    }
    private void compression(int[] array_int){
        long twe_32 = 4294967296l;
        int a = h0;
        int b = h1;
        int c = h2;
        int d = h3;
        int e = h4;
        int f = h5;
        int g = h6;
        int h = h7;
        for(int j = 0; j < 64; j++){
            int Sum_1 = Integer.rotateRight(e, 6) ^ Integer.rotateRight(e,
11) ^ Integer.rotateRight(e,25);
            int Ch = (e & f) ^ ((~e) & g );
            int T_1 = (int) (h + Sum_1 + Ch + ARR_CONST[j] + array_int[j] %
twe_32);
            int Sum_0 = Integer.rotateRight(a, 2) ^ Integer.rotateRight(a,
```

```
13) ^ Integer.rotateRight(a, 22);
            int Ma = (a & b) ^ (a & c) ^ (b & c);
            int T_2 = (int) (Sum_0 + Ma % twe_32);
            h = g;
            g = f;
            f = e;
            e = (int) (d + T_1 % twe_32);
            d = c;
            c = b;
            b = a;
            a = (int) (T_1 +T_2 % twe_32);

        }

        h0 += a;
        h1 += b;
        h2 += c;
        h3 += d;
        h4 += e;
        h5 += f;
        h6 += g;
        h7 += h;
    }
    public String gethex(){
        String hex = Integer.toHexString(h0) + Integer.toHexString(h1) +
Integer.toHexString(h2) + Integer.toHexString(h3) + Integer.toHexString(h4) +
Integer.toHexString(h5) + Integer.toHexString(h6) + Integer.toHexString(h7);
        return hex;
    }
    public int getH0(){
        return h0;
    }
    public int getH1(){
        return h1;
    }

    public int getH2() {
        return h2;
    }

    public int getH3() {
        return h3;
    }

    public int getH4() {
        return h4;
    }

    public int getH5() {
        return h5;
    }

    public int getH6() {
        return h6;
    }

    public int getH7() {
        return h7;
    }
    public BigInteger getInteger(){
        String hex =  gethex();
        BigInteger result = new BigInteger(String.valueOf(0));
```

```
        BigInteger element =  new BigInteger(String.valueOf(0));
        String element_str;
        Long SHA_int;
        BigInteger step_16 = new BigInteger(String.valueOf(16));
        for(int i = 0; i < hex.length(); i++){
            element_str = hex.substring(i, i+1);
            SHA_int = Long.parseLong(element_str, 16);
            element_str = SHA_int.toString();
            element = new BigInteger(element_str);
            element = element.multiply(step_16.pow(hex.length() - i - 1));
            result =  result.add(element);
        }
        return result;
    }
    public int getNumber_blocks(){
        return number_blocks;
    }
}
```

Листинг 5.3 – SHA.java

```
import java.math.BigInteger;
import java.util.Random;

import static java.math.BigInteger.probablePrime;

public class key_generation {
    private BigInteger open_exhibitor = BigInteger.ONE;
    private BigInteger close_exhibitor;
    private BigInteger euler_function;
    private int maxleng = 1024;
    private BigInteger second_number;
    private BigInteger derivative;
    private BigInteger first_number;

    public key_generation(){
        first_number = probablePrime(1024,new Random());
        second_number = probablePrime(1024, new Random());
        derivative = first_number.multiply(second_number);
        BigInteger first_number_sub_1 =
first_number.subtract(BigInteger.ONE);
        BigInteger second_number_sub_1 =
second_number.subtract(BigInteger.ONE);
        euler_function =
first_number_sub_1.multiply(second_number_sub_1);
        //System.out.println(second_number + "\n" +
first_number);
        gen();
        //BigInteger open_exhibitor = new
BigInteger(String.valueOf(65537));
        //System.out.println("open " + open_exhibitor);
        //System.out.println("close " + close_exhibitor);

    }
    private void gen(){
```

```
        open_exhibitor = BigInteger.probablePrime(maxleng / 2,
new Random());
        while
(euler_function.gcd(open_exhibitor).compareTo(BigInteger.ONE) >
0 && open_exhibitor.compareTo(euler_function) < 0){
            open_exhibitor.add(BigInteger.ONE);
        }

        close_exhibitor =
open_exhibitor.modInverse(euler_function);

        //System.out.println(open_exhibitor);

    }
    public BigInteger getOpen_exhibitor(){
        return open_exhibitor;
    }

    public BigInteger getClose_exhibitor(){
        return close_exhibitor;
    }
    public BigInteger getDerivative(){
        return derivative;
    }
    public BigInteger getFirst_number(){
        return first_number;
    }
    public BigInteger getSecond_number(){
        return second_number;
    }




}
```

Листинг 5.4 – key_generatiion.java

```
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.math.BigInteger;

public class key_output {
    private String path_close_key;
    private String path_open_key;
    public key_output(BigInteger open_exhibitor, BigInteger
close_exhibitor, BigInteger derivative, BigInteger first_number,
BigInteger second_number) throws IOException {
        output_open_key(open_exhibitor, derivative);
        output_close_key(close_exhibitor, derivative);
        simple_number(first_number, second_number);
```

```
    }
    private void output_open_key(BigInteger open_exhibitor,
BigInteger derivative) throws IOException {
        File file = new File("open key.txt");
        path_open_key = file.getCanonicalPath();
        FileWriter output = new FileWriter(file);
        String open_exhib = open_exhibitor.toString();
        String der = derivative.toString();
        output.write("open exhibitor:" + open_exhib +
"\nderivative:" + der);
        output.close();
    }
    private void output_close_key(BigInteger close_exhibitor,
BigInteger derivative) throws IOException {
        File file = new File("close key.txt");
        path_close_key = file.getCanonicalPath();
        //System.out.println(file.getCanonicalPath());
        FileWriter output = new FileWriter(file);
        String close_exhib = close_exhibitor.toString();
        String der = derivative.toString();
        output.write("close exhibitor:" + close_exhib +
"\nderivative:" + der);
        output.close();

    }
    private void simple_number(BigInteger first_number,
BigInteger second_number) throws IOException {
        FileWriter output = new FileWriter("simple_number.txt");
        String first = first_number.toString();
        String second = second_number.toString();
        output.write("first number: " + first + "\nsecond
number: " + second);
        output.close();
    }
    public String path_open_key(){
        return path_open_key;
    }
    public String path_close_key(){
        return path_close_key;
    }
}
```

Листинг 5.5 – key_output.java

```
import java.io.*;
import java.math.BigInteger;

public class Signature {
    private String name_file_txt;
    private String name_file_close_key;
    private BigInteger signature;
    private BigInteger close_key;
```

```
    private BigInteger derivative;
    Signature(String name_file_txt, String name_file_close_key)
throws IOException {
        this.name_file_txt = name_file_txt;
        this.name_file_close_key = name_file_close_key;
        new open_close_key_file();
        new open_text_file();

    }
    class open_text_file{
        private open_text_file() throws IOException {
            String line ;
            String line_sum = "";
            FileReader file_txt_ = new
FileReader(name_file_txt);
            BufferedReader buf_txt = new
BufferedReader(file_txt_);
            while ((line = buf_txt.readLine()) != null){
                line_sum += line + "\n";
            }
            file_txt_.close();
            buf_txt.close();
            line_sum = line_sum.substring(0,line_sum.length() -
1);

            long start = System.currentTimeMillis();

            SHA sha = new SHA(line_sum);

            long finish = System.currentTimeMillis();
            System.out.println("Время работы хэш-функции: " +
(finish - start));

            BigInteger sha_int = sha.getInteger();

            if(sha_int.compareTo(derivative) <= -1){
                signature =
sha_int.modPow(close_key,derivative);
                FileWriter file_signature = new
FileWriter(name_file_txt);
                file_signature.write(line_sum + "\n" +
signature.toString());
                file_signature.close();

            }else{System.out.println("Ключ мал");}
        }

    }

    class open_close_key_file{
        private open_close_key_file() throws IOException {
            FileReader file_txt = new
```

```
FileReader(name_file_close_key);
            BufferedReader buf = new BufferedReader(file_txt);
            String line_1 = buf.readLine();
            //System.out.println(line_1);
            String line_2 = buf.readLine();
            line_1 = line_1.substring(16);
            line_2 = line_2.substring(11);
            close_key = new BigInteger(line_1);
            derivative = new BigInteger(line_2);
            buf.close();
            file_txt.close();
        }
    }



}
```

Листинг 5.6 – Signature.java

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.math.BigInteger;

public class Signature_verification {
    private String name_file_signature_file;
    private String name_file_open_key;
    private BigInteger signature;
    private String message = "";
    private BigInteger open_exhibitor;
    private BigInteger derivative;

    public Signature_verification(String
name_file_signature_file, String name_file_open_key) throws
IOException {
        this.name_file_signature_file =
name_file_signature_file;
        this.name_file_open_key = name_file_open_key;
        data_file_signature_file();
        data_name_file_open_key();

    }
    public String check() throws IOException {
        SHA sha = new SHA(message);
        BigInteger sha_int = sha.getInteger();
        BigInteger check = new BigInteger("0");
        if ((sha_int.compareTo(derivative)) <= -1){
            check = signature.modPow(open_exhibitor,
derivative);
            if((check.compareTo(sha_int)) == 0){
```

```
                FileWriter file = new
FileWriter(name_file_signature_file);
                file.write(message);
                file.close();
                return "подлинный";
            }
        }
        return "фальсификация";
    }
    private void data_name_file_open_key()throws IOException{
        FileReader file_open_key = new
FileReader(name_file_open_key);
        BufferedReader buf = new BufferedReader(file_open_key);
        String open_exhibitor = buf.readLine();
        String derivative = buf.readLine();
        open_exhibitor = open_exhibitor.substring(15);
        derivative = derivative.substring(11);
        this.open_exhibitor = new  BigInteger(open_exhibitor);
        this.derivative = new BigInteger(derivative);

        buf.close();
        file_open_key.close();
    }
    private void data_file_signature_file()throws IOException {
        String signature = "";
        String line = "";
        FileReader read_signature_file = new
FileReader(name_file_signature_file);
        BufferedReader buf_signature_file = new
BufferedReader(read_signature_file); // чек buf
        while ((line = buf_signature_file.readLine()) != null){
            message += line + "\n";
            signature = line;
        }
        message = message.substring(0, message.length() -
signature.length() - 2);

        this.signature = new BigInteger(signature);
        buf_signature_file.close();
        read_signature_file.close();
    }
}
```

Листинг 5.6 – Signature_verification.java