Московский государственный университет информационных технологий, радиотехники и электроники

Институт Кибернетики

Кафедра Информационной безопасности

КУРСОВАЯ РАБОТА

По дисциплине

«Методы программирования»

«Реализация ЭЦП на основе алгоритма RSA [с хэш-функцией MD5]»

Студент группы ККСО-1-13

Чистяков Вадим Евгеньевич

Руководители курсовой работы:

Лавреньтев А.В., Кирюхин В.А.

Москва, 2015

Оглавление

1. Введение

1.1 Техническое задание.

Реализовать программу для ЭЦП файла методом RSA. Программа по запросу пользователя должна генерировать пару файлов: с открытым ключом — для проверки подписи, с закрытым ключом — для создания подписи. В режиме подписи файла на вход поступают: имя подписываемого файла и файл с закрытым ключом; выход — подписанный файл (файл дополненный блоком, содержащим подпись). В режиме проверки подписи на вход поступают: подписанный файл и файл с открытым ключом; выход — исходный файл и сообщение о наличии или отсутствии искажения.

2. Основные теоретические положения об ЭЦП

Технология применения системы ЭЦП предполагает наличие сети абонентов, обменивающихся подписанными электронными документами. При обмене электронными документами по сети значительно снижаются затраты, связанные с их обработкой, хранением и поиском.
Одновременно при этом возникает проблема, как аутентификации автора электронного документа, так и самого документа, т.е. установление подлинности автора и отсутствия изменений в полученном электронном сообщении. В алгоритмах ЭЦП как и в асимметричных системах шифрования используются однонаправленные функции. ЭЦП используется для аутентификации текстов, передаваемых по телекоммуникационным каналам. ЭЦП представляет собой относительно небольшой объём дополнительной цифровой информации, передаваемой вместе с подписанным текстом. Концепция формирования ЭЦП основана на обратимости асимметричных шифров, а также на взаимосвязанности содержимого сообщения, самой подписи и пары ключей. Изменение хотя бы одного из этих элементов сделает невозможным подтверждение подлинности подписи, которая реализуется при помощи асимметричных алгоритмов шифрования и хэш-функций.

Система ЭЦП включает две процедуры:

- формирование цифровой подписи;
- проверку цифровой подписи.

В процедуре формирования подписи используется секретный ключ отправителя сообщения, в процедуре проверки подписи – открытый ключ отправителя.

2.1 ЭЦП на базе RSA

RSA (аббревиатура от фамилий Rivest, Shamir и Adleman) — криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших целых чисел.

Обобщённая схема формирования и проверки электронной цифровой подписи приведена на рис.1.

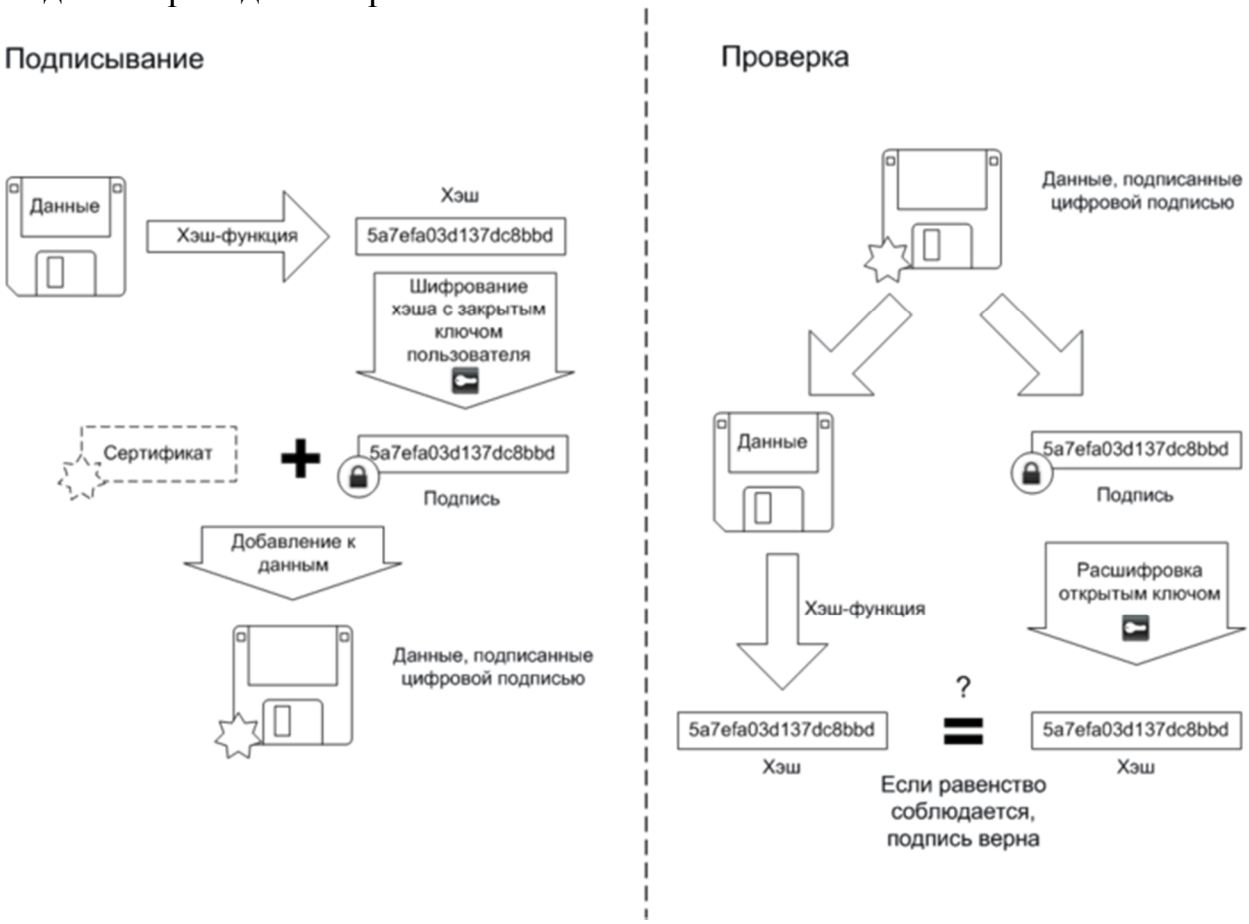

Рис. 1. Схема электронной цифровой подписи RSA

2.2Алгоритм электронной цифровой подписи (ЭЦП) RSA

2.2.1 Определение открытого *« e»* и секретного « *d*» ключей

(действия отправителя)

1. Выбор двух взаимно простых больших чисел ***p*** и ***q***

2. Определение их произведения $n = p \cdot q$

3. Определение функции Эйлера: $\phi(n) = (p-1)(q-1)$

4. Выбор секретного ключа d с учетом условий: $1 < d \leq \phi(n)$,

$$НОД(d, \phi(n)) = 1$$

5. Определение значения открытого ключа e: $e < n$,

$$e \cdot d \equiv 1 (mod\ \phi(n))$$

2.2.2 Формирование ЭЦП

1. Вычисление хэш-значения сообщения M: $m = h(M)$

2. Для получения ЭЦП шифруем хэш-значение m с помощью секретного ключа d и отправляем получателю цифровую подпись $S = m^d \ (mod\ n)$ и открытый текст сообщения M

3.Аутентификация сообщения - проверка подлинности подписи

1. Расшифровка цифровой подписи S с помощью открытого ключа e и вычисление её хэш-значения $m' = S^e \ (mod\ n)$

2. Вычисление хэш-значения принятого открытого текста M

$$m = h(M)$$

3. Сравнение хэш-значений m и m', если $m=m'$, то цифровая подпись S – достоверна.

2.2.3. Пример работы алгоритма RSA

Хешируемое сообщение M представим как последовательность целых чисел 312. В соответствии с приведённым выше алгоритмом формирования ЭЦП RSA выбираем два взаимно простых числа $p = 3, q = 11$, вычисляем значение $n = p*q = 3*11 = 33$, выбираем значение секретного ключа $d = 7$ и вычисляем значение

открытого ключа $e = 3$. Вектор инициализации H_0

выбираем равным 6 (выбирается случайным образом). Хэш-код сообщения $M = 312$ формируется следующим образом:

$$H_1 = (M_1 + H_0)^2 \pmod n = (3 + 6)^2 \pmod{33} = 81 \pmod{33} = 15;\ H_2 = (M_2 + H_1)^2 \pmod n = (1 + 15)^2 \pmod{33} = 256 \pmod{33} = 25;$$

$$H_3 = (M_3 + H_2)^2 \pmod n = (2 + 25)^2 \pmod{33} = 729 \pmod{33} = 3;\ m = 3$$

1. Для получения ЭЦП шифруем хэш-значение m с помощью секретного ключа d и отправляем получателю цифровую подпись

$S = m^d \pmod n$ и открытый текст сообщения M

$$S = 3^7 \pmod{33} = 2187 \pmod{33} = 9$$

2. Проверка подлинности ЭЦП

Расшифровка S (т. е. вычисление её хэш-значения m') производится с

помощью открытого ключа e.

$$m' = S^e \pmod n = 9^3 \pmod{33} = 729 \pmod{33} = 3$$

Если сравнение хэш-значений m' и m показывает их равенство, т.е. $m = m'$, то подпись достоверна.

3.Описание программной реализации.

- Приложение написано на языке программирования C#.
- Программная платформа .NET Framework 4.0
- Среда разработки Microsoft Visual Studio Express 2013
- ОС Microsoft Windows 7

Проект *RSA* содержит 3 модуля:

- *RSA_BigInt.cs* – класс «длинной арифметики».
- *RSA_Sign.cs* – класс реализует создание ЭЦП методом RSA.
- *RSA_MAIN.cs* – класс осуществляет проверку работоспособности программы.

3.1 Модуль *RSA_BigInt.cs*

Алгоритм RSA использует для создания ключей большие целые числа, превышающие машинное слово (128бит, 256бит и т.д) поэтому нам необходимо реализовать программно возможность работы с такими числами.
Для этого нам понадобится класс «длинной арифметики».

Длинная арифметика — выполняемые с помощью вычислительной машины арифметические операции (сложение, вычитание, умножение, деление, возведение в степень, логические операции и др) над числами, разрядность которых превышает длину машинного слова данной вычислительной машины.

Класс длинной арифметики `public class BigInt32` содержит:

Поля:

```
/// Максимальная (const) длина BigInt32 в массиве uint[] data (4 байта)
private const int maxLength = 50;

/// Массив типа uint с помощью которого представляется число BigInt32
public uint[] data = null;

/// Число элементов в BigInt32
public int dataLength;
```

Конструкторы класса:

```
/// Конструктор без параметра
/// </summary>
public BigInt32()

/// Конструктор с параметром типа long
/// </summary>
public BigInt32(long value)

/// Конструктор копирования
/// </summary>
public BigInt32(BigInt32 bi)

/// Представляет длинное число из строки
/// </summary>
```

```
/// <param name="value">Исходная строка</param>
/// <param name="radix">Основание</param>
public BigInt32(string value)

/// Конструктор BigInt32 из массива байтов
/// </summary>
public BigInt32(byte[] inData)

///  Конструктор BigInt32 из массива целых чисел
/// </summary>
public BigInt32(uint[] inData)
```

Оператор неявного преобразование для long, ulong, int, uint

```
public static implicit operator BigInt32(long value)

public static implicit operator BigInt32(ulong

public static implicit operator BigInt32(int value)

public static implicit operator BigInt32(uint value)
```

Перегразука операций

```
/// Перегрузка оператора сложения
/// </summary>
public static BigInt32 operator +(BigInt32 bi1, BigInt32 bi2)

/// <summary>
/// Перегрузка оператора инкримента
/// </summary>
public static BigInt32 operator ++(BigInt32 bi1)

/// <summary>
/// Перегрузка оператора вычитания
/// </summary>
public static BigInt32 operator -(BigInt32 bi1, BigInt32 bi2)

/// <summary>
/// Перегрузка оператора декримента
/// </summary>
public static BigInt32 operator --(BigInt32 bi1)
```

```
/// <summary>
/// Перегрузка оператора умножения
/// </summary>
public static BigInt32 operator *(BigInt32 bi1, BigInt32 bi2)

/// <summary>
/// Перегрузка оператора сдвига влево <<
/// </summary>
/// <param name="bi1"> Папаметр значения </param>
/// <param name="shiftVal"> Величина сдвига</param>
public static BigInt32 operator <<(BigInt32 bi1, int shiftVal)

/// <summary>
/// Перегрузка оператора сдвига вправо >>
/// </summary>
/// <param name="bi1"></param>
/// <param name="shiftVal"></param>
public static BigInt32 operator >>(BigInt32 bi1, int shiftVal)

/// <summary>
/// Перегрузка оператора отрицания
/// </summary>
public static BigInt32 operator -(BigInt32 bi1)

/// <summary>
/// Определяет считаются ли равными объекты класса BigInt32
/// </summary>
public override bool Equals(object o)

/// <summary>
/// Перегрузка операторов равенства/неравенства
/// </summary>
public static bool operator ==(BigInt32 bi1, BigInt32 bi2)

/// Необходимо перегрузить (Т.к метод Equals перегружен в данном
классе)
public override int GetHashCode()

/// <summary>
/// Перегрузка операторов строгово/ нестрогово сравнения
/// </summary>
public static bool operator >(BigInt32 bi1, BigInt32 bi2)

public static bool operator >=(BigInt32 bi1, BigInt32 bi2)
```

```
public static bool operator <=(BigInt32 bi1, BigInt32 bi2)

/// <summary>
/// Перегрузка оператора деления
/// </summary>
public static BigInt32 operator /(BigInt32 bi1, BigInt32 bi2)

/// <summary>
/// Перегрузка оператора присваивания остатка
/// </summary>
public static BigInt32 operator %(BigInt32 bi1, BigInt32 bi2)
```

Методы

```
/// <summary>
/// Метод сдвига влево
/// </summary>
private static int shiftLeft(uint[] buffer, int shiftVal)

/// <summary>
/// Метод сдвига вправо
/// </summary>
private static int shiftRight(uint[] buffer, int shiftVal)

/// <summary>
/// Алгоритм деления двух больших чисел.
/// Делитель содержит больше 1 цифры
/// </summary>
/// <param name="bi1">Делимое</param>
/// <param name="bi2">Делитель</param>
/// <param name="Quotient">Частное</param>
/// <param name="Remainder">Остаток</param>
private static void AlgorithmOfDivideMultiByte(BigInt32 bi1,
BigInt32 bi2, BigInt32 Quotient, BigInt32 Remainder)

/// <summary>
/// Алгоритм деления числа BigInt32 на однозначное число
/// </summary>
private static void AlgorithmOfDivideSingleByte(BigInt32 bi1,
BigInt32 bi2, BigInt32 Quotient, BigInt32 Remainder)
```

```
/// <summary>
/// Возвращает минимальный BigInt32
/// </summary>
public BigInt32 min(BigInt32 bi)

/// <summary>
/// Возвращает строку из BigInt32
/// </summary>
public override string ToString()

/// <summary>
/// Выполняет модульное деление числа, возведенного в степень
exp.
/// </summary>
/// <param name="exp">экспонента</param>
/// <param name="n">модуль</param>
/// <returns></returns>
public BigInt32 modPow(BigInt32 exp, BigInt32 n)

/// <summary>
/// Вычисляет return = x mod n.
/// Использованная литература.
/// Анализ арифметических операций современной криптографии и
способы их аппаратной реализации Расулов О.Х
/// </summary>
/// <param name="x"> Делимое </param>
/// <param name="n"> модуль </param>
/// <param name="constant"> константа "мю" </param>
private BigInt32 AlgorithmOfBarrettReduction(BigInt32 x,
BigInt32 n, BigInt32 constant)

/// <summary>
/// Генератор случайных бит
/// </summary>
public void GetRandomBits(int bits, Random rand)

/// <summary>
/// Возвращает позицию старшего значимого  бита в BigInt32.
/// </summary>
public int bitCount()

/// <summary>
/// Вероятностный тест на простоту основанный на алгоритме
Милера-Рабина
```

```
/// Для любого p > 0, при p - 1 = 2^s * t
/// p вероятно простое для любого  a < p,
/// 1) a^t mod p = 1 или
/// 2) a^((2^j)*t) mod p = p-1 для некоторого 0 <= j <= s-1
/// </summary>
public bool RabinMillerTest(int confidence)

/// <summary>
/// Алгоритм Евклида
/// </summary>
/// <returns> НОД </returns>
public BigInt32 gcd(BigInt32 bi)

/// <summary>
/// Возвращает младшие 4 байта в BigInt32 как int.
/// </summary>
public int IntValue()

/// <summary>
/// Генерирует положительный  BigInt32. С большой вероятностью
простое.+
/// </summary>
public static BigInt32 GetPseudoPrime(int bits, int confidence,
Random rand)

/// <summary>
/// Статический НОД чисел a и b
/// </summary>
public static BigInt32 NOD(BigInt32 a, BigInt32 b)

/// <summary>
/// Расширенный алгоритм Евклида (для нахождения обратного
элемента по некоторому модулю)
/// </summary>
private static void Evklid(BigInt32 a, BigInt32 b, ref BigInt32
x, ref BigInt32 y, ref BigInt32 d)

/// <summary>
/// Ищет обратный элемент к e по модулю n
/// </summary>
public static BigInt32 Inverse(BigInt32 e, BigInt32 n)
```

3.2 Модуль *RSA_Sign.cs*

Модуль *RSA_Sign.cs* содержит класс *public RSA,* который отвечает за формирование двух ключей (открытый/закрытый), хэширование файла хэш-функцией MD5, создание подписи файла, проверка подписи файла на подлинность.

Поля класса (доступны только для чтения):

```
/// <summary>
/// Модуль n
/// </summary>
public readonly BigInt32 n;

/// <summary>
/// Открытый ключ
/// </summary>
public readonly BigInt32 e;

/// <summary>
/// Закрытый ключ
/// </summary>
public readonly BigInt32 d;
```

Конструктор класса:

```
/// <summary>
/// Конструктор. Генерирует и проверяет ключи.
/// </summary>
public RSA()
{
Random uy = new Random();

const int KeyOFLenght = 128; // Длина p и q

BigInt32 p = BigInt32.GetPseudoPrime(KeyOFLenght, 10, uy);
BigInt32 q = BigInt32.GetPseudoPrime(KeyOFLenght - 1, 10, uy);

n = p * q; // Модуль как произведение q и p

p--;
q--;
```

```
BigInt32 w = p * q;                          // функция Эйлера

e = new BigInt32();                         // открытый ключ

do
{
e.GetRandomBits(KeyOFLenght * 2, uy);
}
while (BigInt32.NOD(e, w) != 1 && (e > n)); // Генерируем
случайный e до тех пор пока e не будет больше n и взаимно просто
с w

d = BigInt32.Inverse(e, w); // d - закрытый ключ
}                           // d - существует <=> НОД(e,w) = 1
```

Методы класса:

```
/// <summary>
/// Вычисляет хэш исходных данных используя алгоритм хэширования
MD5
/// </summary>
/// <param name="data">Исходные данные</param>
/// <returns>128-битный хэш</returns>
private static byte[] MD5hash(byte[] data)

/// <summary>
/// Вырабатывает ЭЦП сообщения-Text
/// </summary>
/// <param name="Text">Сообщение</param>
/// <returns>Подпись</returns>
public static BigInt32 CreateSignature(byte[]Text, BigInt32 d,
BigInt32 n)

/// <summary>
/// Проверка цифровой подписи Signature текста Text
/// </summary>
/// <param name="Text">Текст</param>
/// <param name="Signature">Подпись</param>
/// <param name="e">Открытый ключ</param>
/// <param name="n">Модуль</param>
/// <returns>true если подпись верна, false в обратном
случае</returns>
```

```
public static bool VerifySignature(byte[] Text, BigInt32
Signature, BigInt32 e, BigInt32 n)
```

3.3 Модуль RSA_MAIN.cs

Отвечает за взаимодействие с пользователем.
После запуска проекта мы увидим следующее окнорис2:

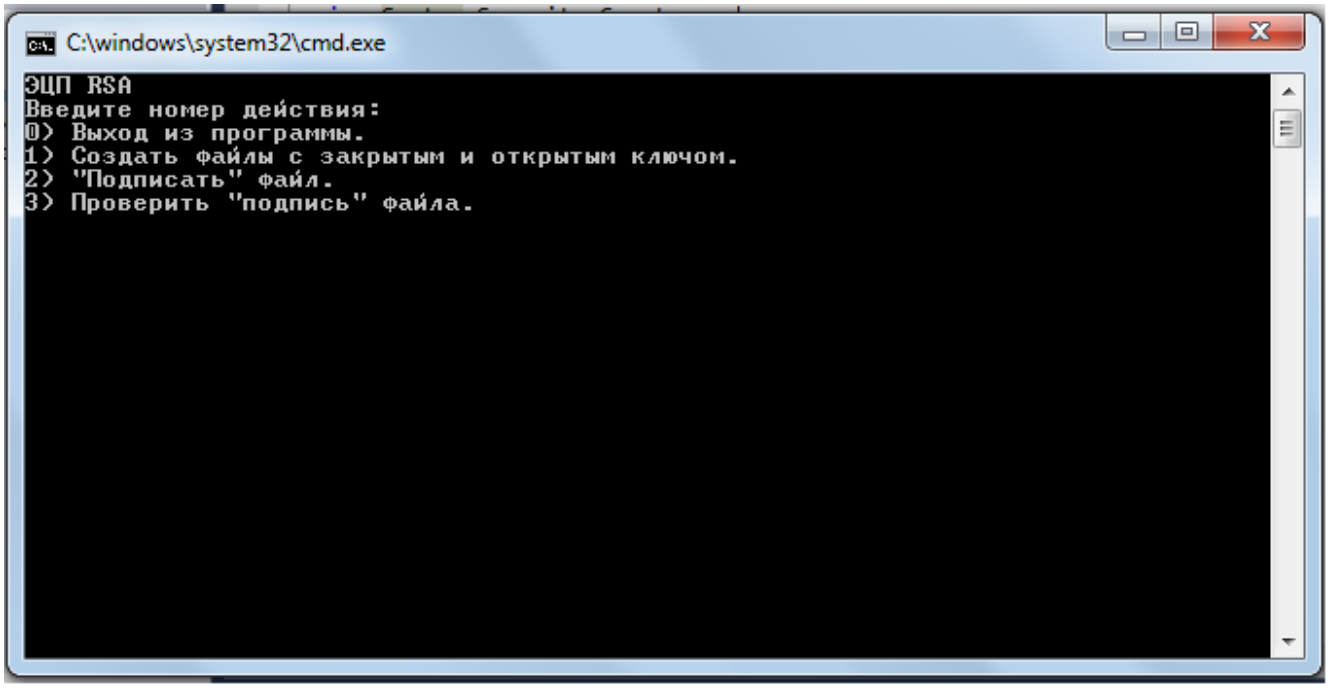

Рис2.

Выбрав первый пункт (клавиша «1» и enter) вызывается статический метод *CreateFileOFKeys().*

```
/// <summary>
/// Создает файлы "OpenKey.txt" с открытым ключом и "Secret.txt"
с закрытым
/// </summary>
static void CreateFilesOfKeys()
```

В каталоге приложения появятся два файла «OpenKey.txt» и «SecretKey.txt» с открытым и закрытым файлом соответственно рис3.

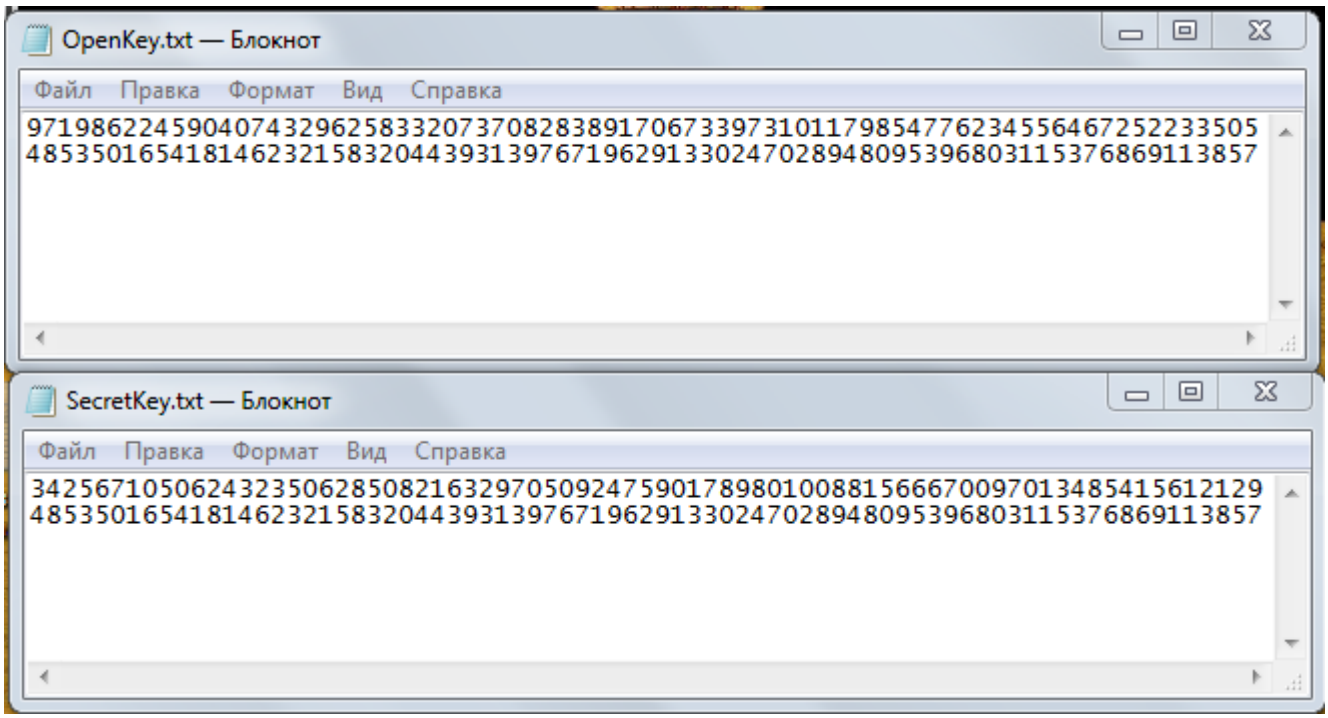

Рис3.

Далее, выбрав второй пункт (клавиша «2» и enter) вызывается статический метод `CreateFilesOfSign`.

```
/// Создание ЭЦП файла
/// </summary>
/// <param name="filename">Имя файла, который мы хотим подписать</param>
static void CreateFilesOfSign(string filename)
```

Например, мы хотим подписать файл “Test.txt” рис4.

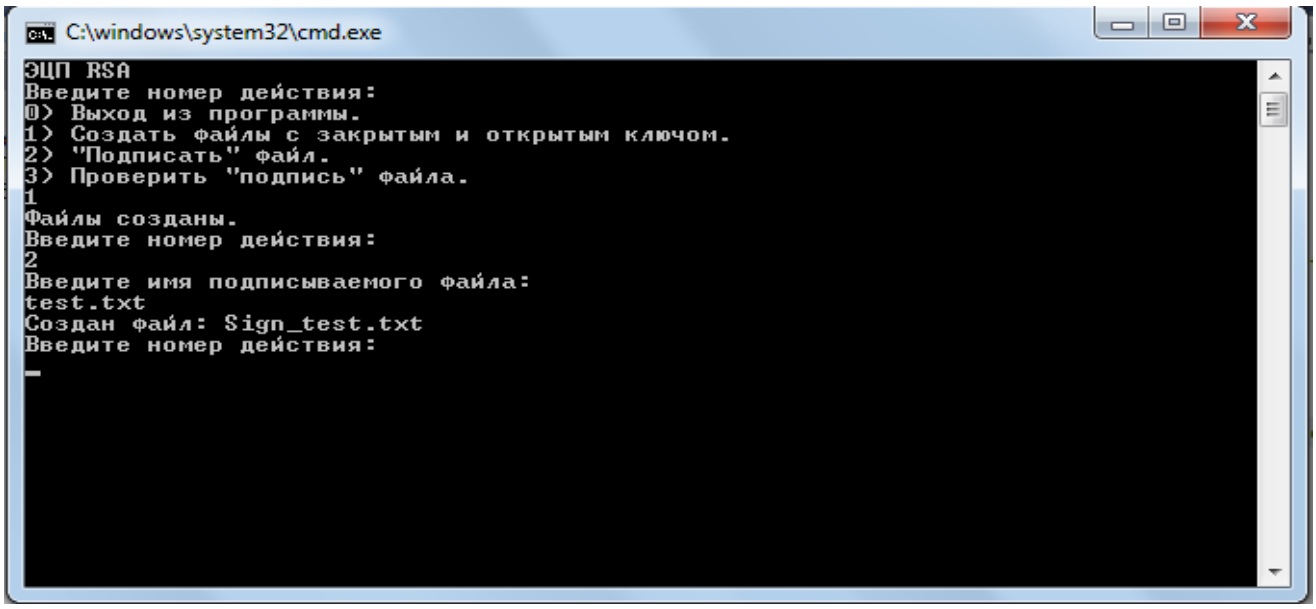

Рис4.

В каталоге программы появится файл «Sign_test.txt» расширенный строкой с ЭЦП и строкой с именем подписываемого файла рис5.

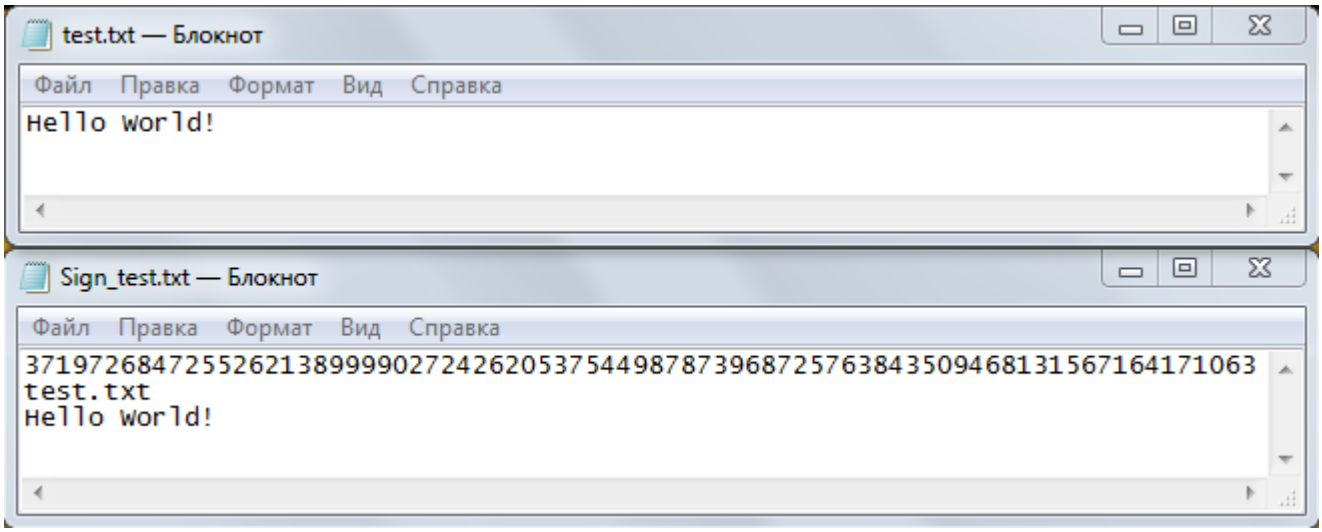

Рис5.

Если мы выберем пункт 3 нашего приложения и введем имя подписанного файла, то вызовется статический метод `VerificationSignature`

```
/// Проверка цифровой подписи файла на подлинность
/// </summary>
/// <param name="filename"> Имя подписанного файла</param>
static void VerificationSignature(string filename)
```

Метод `VerificationSignature` выдает строку на экран «подпись верна» в случае если целостность файла и ЭЦП не были нарушены и «подпись не верна в противном случае рис6.

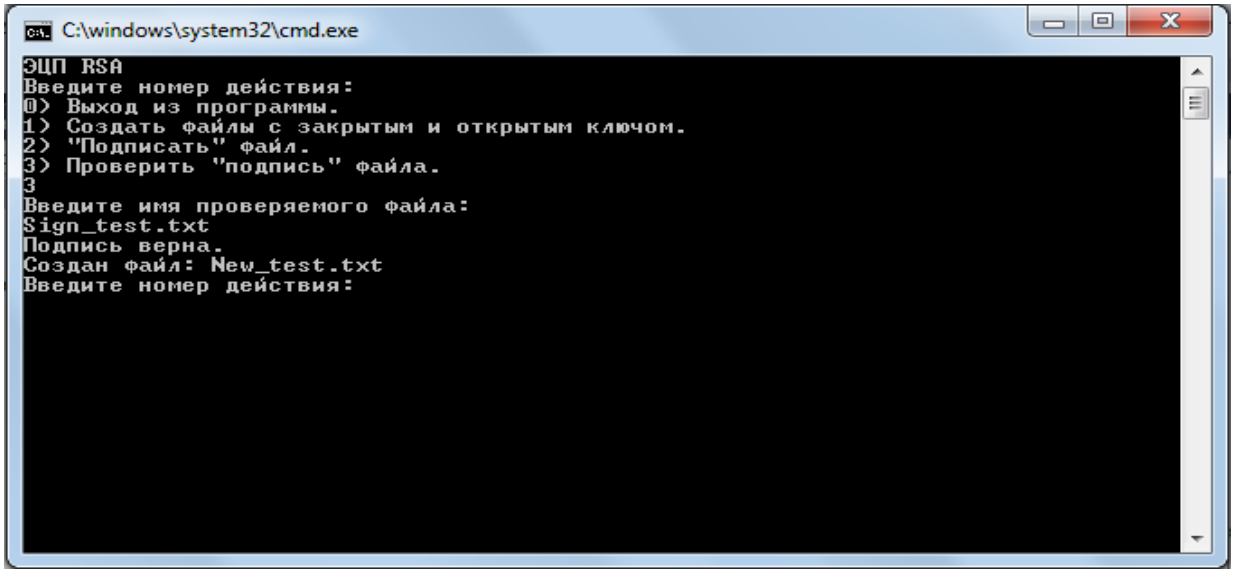

Рис6.

Если файл прошел проверку подлинности ЭЦП, то приложение создат файл «New_test.txt» содержащий исходные данные файла «test.txt»

Для выхода из программы нажмите 0.

4. Заключение.

В данной курсовой работе продемонстрирована работа приложения, позволяющего создать, ЭЦП методом RSA для любых типов файлов и любой длинны подписи. А также произвести проверку на соответствие файла его цифровой подписи.

При создание приложения не использовалось платформозависимых решений.

5. Список использованной литературы и сайтов.

1. http://orenstudent.ru/RSA256.htm
2. https://ru.wikipedia.org/wiki/Электронная_подпись
3. https://ru.wikipedia.org/wiki/rsa
4. Длинная Арифметика. В.Гольдштейн
5. Анализ арифметических операций современной криптографии и способы их аппаратной реализации. Расулов О.Х
6. Кормен, Лейзерсон, Ривест, Штайн. Алгоритмы. Построение и анализ 2-е издание.
7. Алферов А.П, Зубов А.Ю, Кузьмин А.С, Черемушкин А.В. Основы криптографии. 2002г