Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Исследование алгоритмов генерации**

**и верификации электронной цифровой**

**подписи**

Студент: Рубашек А. А.

ФИТ 3 курс 5 группа

Преподаватель:

Савельева Маргарита Геннадьевна

1. **Цель работы**

Изучение алгоритмов генерации и верификации электронной цифровой подписи и приобретение практических навыков их реализации.

1. **Задание**
2. Разработать авторское оконное приложение в соответствии с целью лабораторной работы. При этом можно воспользоваться результатами выполнения предыдущих лабораторных работ, а также доступными библиотеками либо программными кодами. Приложение должно реализовывать генерацию и верификацию ЭЦП на основе алгоритмов RSA, Эль-Гамаля и Шнорра, оценку времени выполнения указанных процедур при реальных (требуемых) ключевых параметрах. Для вычисления хешей можно также воспользоваться доступными online-средствами, например katvin.
3. Для выполнения необходимых операций передачи (по сети)/верификации информации обменяться открытой ключевой информацией с получателем подписанного сообщения для каждого исследуемого алгоритма (по согласованию с преподавателем).
4. **Ход работы**
   1. **ЭЦП на основе RSA**

Цифровая подпись S вычисляется на основе известного соотношения:

, *dо* и *no* – элементы тайного ключа отправителя. Передаваемое сообщение *М' = Мo || S*.

Соответственно, операция расшифрования на приемной стороне (получатель анализирует *Мп* || *S*) будет производиться в соответствии с формулой с известной модификацией ключей: . Далее вычисляется *Н*(*Mп*). Если *H(Mo)* = *H(Mп)*, подпись верифицирована.

Код для создания и верификации ЭЦП на основе RSA представлен в классе RSA в методах CreateDigitalSignature и VerifyDigitalSignature, которые представлены в Приложении 1. Результат работы этих функций представлен на рисунке 3.1.

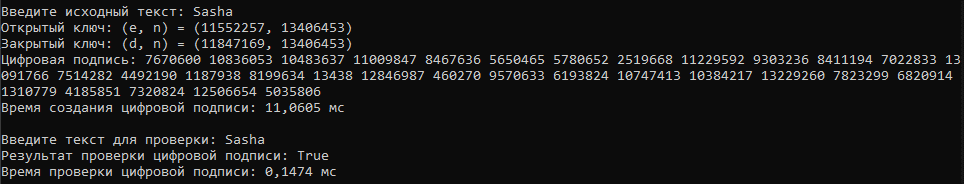


Рисунок 3.1 – Результат работы класса RSA

* 1. **ЭЦП Эль-Гамаля**

Ключевая информация отправителя: открытый ключ: *y*, *g* и *р*; тайный ключ: *х*. Чтобы подписать сообщение *Мо*, обладатель используемых для ЭЦП ключей должен выбрать, как и в предыдущей схеме, случайное число *k*, взаимно простое с (*р* – 1). Затем вычисляется числа *а* и *b*, являющиеся цифровой подписью (*S* = {*a*, *b*}):

*a ≡ gk*mod *p*.

Для вычисления *b* с помощью расширенного алгоритма Евклида решается уравнение:

*Н*(*Mо*) ≡ (*x ⋅ a* + *k ⋅ b*) mod (*p* – 1).

Получателю отправляется сообщение *М*' = *Мо* || *S*. Для верификации подписи вычисляется хеш полученного сообщения *Н*(*Мп*) = *h*. Далее нужно убедиться, что выполняется равенство:

*yaab* ≡ *gh* mod *p*.

Если равенство выполняется, подпись верифицируется.

Код для создания и верификации ЭЦП Эль-Гамаля представлен в классе ElGamal в методах CreateDigitalSignature и VerifyDigitalSignature, которые представлены в Приложении 2. Результат работы этих функций представлен на рисунке 3.2.

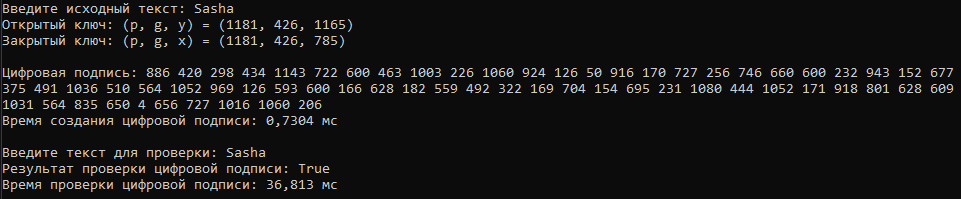


Рисунок 3.2 – Результат работы класса ElGamal

* 1. **ЭЦП Шнорра**

Ключевая информация: *p* – простое число в диапазоне от 512 до 1024 битов; *q* –160-битное простое число, делитель (p – 1); любое число g (g ≠ 1) такое, что *gq* ≡ 1 mod *p*.

Выбирается число *х* < *q* (*х* является тайным ключом) и вычисляется последний элемент открытого ключа: *y* ≡ *g–х* mod *p*.

Для подписи сообщения *Мо*выбирается случайное число *k* (1 < k < q) и вычисляет параметр а: *а* ≡ *gk* mod *p*.

Далее вычисляется хеш от конкатенации сообщения *Мо* и числа *а*: *h* = *H*(*Mo*||*a*). Создается хеш-образ подписываемого сообщения, спереди присоединенного к числу *а*. Далее вычисляется значение *b*: *b* ≡ (*k* + *x ⋅ h*) mod *q*.

Получателю отправляются *М'* = *Мо || S*; *S* = {*h*, *b*}.

Для проверки подписи получатель вычисляет *Х* ≡ *gbyh*(mod *p*).

Затем он проверяет выполнение равенства: *h* = *Н*(*Mп*||*Х*). Подпись достоверна, если равенство выполняется.

Код для создания и верификации ЭЦП Шнорра представлен в классе Schnorr в методах GenerateDigitalSignature и VerifyDigitalSignature, которые представлены в Приложении 3. Результат работы этих функций представлен на рисунке 3.3.

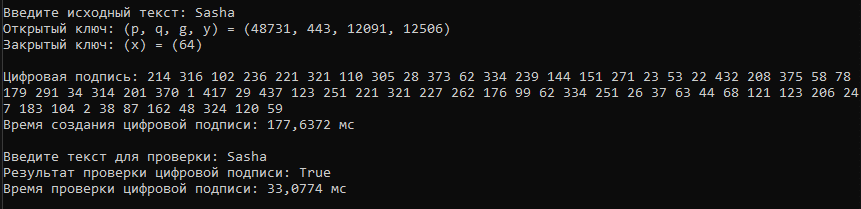


Рисунок 3.3 – Результат работы класса Schnorr

* 1. **Оценка времени**

Так же нужно было оценить скорость генерации и верификации ЭЦП на основе этих алгоритмов. Результаты представлены на рисунках 3.5 и 3.6.

Рисунок 3.5 – Скорость генерации ЭЦП

Рисунок 3.6 – Скорость верификации ЭЦП

Из графиков видно, что RSA имеет более быструю скорость создания и верификации ЭЦП по сравнению с ElGamal и Schnorr. ElGamal и Schnorr обладают более высокой вычислительной сложностью и требуют больше времени для создания и верификации ЭЦП. Время создания и верификации ЭЦП во всех трех алгоритмах увеличивается с увеличением количества символов.

Такие результаты обусловлены тем, что в RSA процесс создания ЭЦП включает операцию возведения в степень с большими числами и вычисление модуля. Однако, RSA позволяет выбрать маленькую экспоненту для ускорения операций и получения более быстрой скорости создания ЭЦП, в то время как в ElGamal используется операция возведения в степень с большими числами, а также вычисление модуля и умножение, а Schnorr имеет более сложный процесс создания ЭЦП, включающий операции с большими числами, такие как умножение, сложение и вычисление хеш-функций. Эти операции требуют значительно больше вычислительных ресурсов и занимают больше времени, особенно при увеличении количества символов.

**Вывод**

В ходе лабораторной работы были изучены алгоритмы генерации и верификации электронной цифровой подписи и приобретены практические навыки их реализации. Кроме того, был сделан вывод о скорости генерации и верификации ЭЦП с помощью различных алгоритмов.

Также было разработано авторское приложение в соответствии с целью лабораторной работы.

**Приложение 1**

Методы CreateDigitalSignature и VerifyDigitalSignature для создания и верификации ЭЦП на основе RSA

public BigInteger[] CreateDigitalSignature(string text)

{

DateTime startTimeRSA = DateTime.Now;

using (SHA256 sha256 = SHA256.Create())

{

hash = sha256.ComputeHash(Encoding.UTF8.GetBytes(text));

}

BigInteger[] digitalSign = new BigInteger[hash.Length];

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

{

digitalSign[i] = BigInteger.ModPow(hash[i], d, n);

}

DateTime endTimeRSA = DateTime.Now;

Console.WriteLine($"Цифровая подпись: {string.Join(" ", digitalSign)}");

Console.WriteLine($"Время создания цифровой подписи: {(endTimeRSA - startTimeRSA).TotalMilliseconds} мс");

return digitalSign;

}

public bool VerifyDigitalSignature(string text, BigInteger[] digitalSign)

{

DateTime startVerifyTimeRSA = DateTime.Now;

byte[] signBytes = new byte[digitalSign.Length];

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

{

signBytes[i] = (byte)BigInteger.ModPow(digitalSign[i], e, n);

}

using (SHA256 sha256 = SHA256.Create())

{

byte[] receivedHash = sha256.ComputeHash(Encoding.UTF8.GetBytes(text));

bool result = VerifyByteArrays(receivedHash, signBytes);

DateTime endVerifyTimeRSA = DateTime.Now;

Console.WriteLine($"Результат проверки цифровой подписи: {result}");

Console.WriteLine($"Время проверки цифровой подписи: {(endVerifyTimeRSA - startVerifyTimeRSA).TotalMilliseconds} мс\n");

return result;

}

**Приложение 2**

Методы CreateDigitalSignature и VerifyDigitalSignature для создания и верификации ЭЦП на основе Эль-Гамаля

public BigInteger[,] CreateDigitalSignature(string message)

{

DateTime startTimeElGamal = DateTime.Now;

using (SHA256 sha256 = SHA256.Create())

{

hash = sha256.ComputeHash(Encoding.UTF8.GetBytes(message));

}

BigInteger[,] digitalSignature = new BigInteger[hash.Length, 2];

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

{

do

{

BigInteger k = random.Next(2, (int)primeP - 2);

while (Helper.GetGCD(k, primeP - 1) != 1)

{

k = random.Next(2, (int)primeP - 2);

}

digitalSignature[i, 0] = BigInteger.ModPow(generatorG, k, primeP);

BigInteger temp = BigInteger.Multiply(BigInteger.Subtract(hash[i], BigInteger.Multiply(privateKeyX, digitalSignature[i, 0])), Helper.ModInverse(k, primeP - 1));

digitalSignature[i, 1] = temp < 0 ? (primeP - 1) - BigInteger.ModPow(BigInteger.Negate(temp), 1, primeP - 1) : BigInteger.ModPow(temp, 1, primeP - 1);

} while (digitalSignature[i, 1] == 0);

}

DateTime endTimeElGamal = DateTime.Now;

Console.WriteLine($"Цифровая подпись: {string.Join(" ", digitalSignature.Cast<BigInteger>())}");

Console.WriteLine($"Время создания цифровой подписи: {(endTimeElGamal - startTimeElGamal).TotalMilliseconds} мс");

return digitalSignature;

}

public bool VerifyDigitalSignature(string message, BigInteger[,] digitalSignature)

{

DateTime startVerifyTimeElGamal = DateTime.Now;

using (SHA256 sha256 = SHA256.Create())

{

hash = sha256.ComputeHash(Encoding.UTF8.GetBytes(message));

}

bool result = true;

for (int i = 0; i < digitalSignature.GetUpperBound(0) + 1; i++)

{

BigInteger leftPart = BigInteger.ModPow(generatorG, hash[i], primeP);

BigInteger rightPart = BigInteger.ModPow(BigInteger.Multiply(BigInteger.Pow(publicKeyY, (int)digitalSignature[i, 0]), BigInteger.Pow(digitalSignature[i, 0], (int)digitalSignature[i, 1])), 1, primeP);

bool compareResult = leftPart == rightPart;

result = result && compareResult;

}

DateTime endVerifyTimeElGamal = DateTime.Now;

Console.WriteLine($"Результат проверки цифровой подписи: {result}");

Console.WriteLine($"Время проверки цифровой подписи: {(endVerifyTimeElGamal - startVerifyTimeElGamal).TotalMilliseconds} мс\n");

return result;

}

**Приложение 3**

Методы CreateDigitalSignature и VerifyDigitalSignature для создания и верификации ЭЦП на основе Шнора

public BigInteger[,] GenerateDigitalSignature(string message)

{

DateTime startTimeSchnorr = DateTime.Now;

BigInteger randomK;

do

{

randomK = random.Next();

} while (!(randomK > 1 && randomK < primeQ));

BigInteger r = BigInteger.ModPow(generatorG, randomK, primeP);

message += r;

using (SHA256 sha256 = SHA256.Create())

{

hash = sha256.ComputeHash(Encoding.UTF8.GetBytes(message));

}

BigInteger[,] digitalSignature = new BigInteger[hash.Length, 2];

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

{

digitalSignature[i, 0] = hash[i];

digitalSignature[i, 1] = BigInteger.Add(randomK, BigInteger.Multiply(secretKeyX, hash[i])) % primeQ;

}

DateTime endTimeSchnorr = DateTime.Now;

Console.WriteLine($"Цифровая подпись: {string.Join(" ", digitalSignature.Cast<BigInteger>())}");

Console.WriteLine($"Время создания цифровой подписи: {(endTimeSchnorr - startTimeSchnorr).TotalMilliseconds} мс");

return digitalSignature;

}

public bool VerifyDigitalSignature(string message, BigInteger[,] digitalSignature)

{

DateTime startVerifyTimeSchnorr = DateTime.Now;

BigInteger x = BigInteger.Multiply(BigInteger.ModPow(generatorG, (int)digitalSignature[0, 1], primeP), BigInteger.ModPow(publicKeyY, (int)digitalSignature[0, 0], primeP)) % primeP;

message += x;

using (SHA256 sha256 = SHA256.Create())

{

hash = sha256.ComputeHash(Encoding.UTF8.GetBytes(message));

}

bool result = hash.SequenceEqual(Enumerable.Range(0, digitalSignature.GetLength(0)).Select(i => digitalSignature[i, 0].ToByteArray()[0]).ToArray());

DateTime endVerifyTimeSchnorr = DateTime.Now;

Console.WriteLine($"Результат проверки цифровой подписи: {result}");

Console.WriteLine($"Время проверки цифровой подписи: {(endVerifyTimeSchnorr - startVerifyTimeSchnorr).TotalMilliseconds} мс\n");

return result;

}