Учреждение образования

«Белорусский государственный технологический университет»

**Отчет по лабораторной работе №10**

по дисциплине: «Защита информации и надежность информационных систем»

Исполнитель:

Воликов Д. А., ФИТ 4-7

Руководитель:

Асс. Савельева М. Г.

Минск 2024

# Лабораторная работа №10. Исследование алгоритмов генерации и верификации электронной цифровой подписи

**Цель**: изучение алгоритмов генерации и верификации электронной цифровой подписи и приобретение практических навыков их реализации.

**Задачи**:

1. Закрепить теоретические знания по алгебраическому описанию и алгоритмам реализации операций генерации и верификации электронной цифровой подписи (ЭЦП).
2. Получить навыки практической реализации методов генерации и верификации ЭЦП на основе хеширования подписываемых сообщений и алгоритмов RSA, Эль-Гамаля и Шнорра, а также DSA.
3. Разработать приложение для реализации заданных алгоритмов генерации и верификации ЭЦП.
4. Оценить скорость генерации и верификации ЭЦП.
5. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

## Теоретические сведения

Электронная цифровая подпись (ЭЦП) является важным элементом современных информационных систем, использующих методы и технологии криптографического преобразования информации.

Электронная цифровая подпись – контрольная характеристика сообщения, которая вырабатывается с использованием личного ключа, проверяется с использованием открытого ключа, служит для контроля целостности и подлинности сообщения и обеспечивает невозможность отказа от авторства.

Таким образом, ЭЦП выполняет те же функции, что и собственноручная (поставленная «от руки») подпись:

* аутентифицирование лица, подписавшего сообщение;
* контроль целостности подписанного сообщения;
* защита сообщения от подделок;
* доказательство авторства лица, подписавшего сообщение, если это лицо отрицает свое авторство.

Важнейшие отличительные особенности ЭЦП:

* ЭЦП представляет собой бинарную последовательность (в отличие от графического образа, каковым является подпись от руки);
* указанная бинарная последовательность зависит от содержания подписываемого сообщения.

Классическая технология использования ЭЦП предусматривает подписание не самого сообщения (обозначим его здесь *M*o), а его хэша, *H*(*M*o). Это сокращает время генерации/верификации подписи и снижает вероятность появления случайных ошибок в итоговом документе.

Основу рассматриваемых протоколов составляют методы асимметричной криптографии и эллиптических кривых.

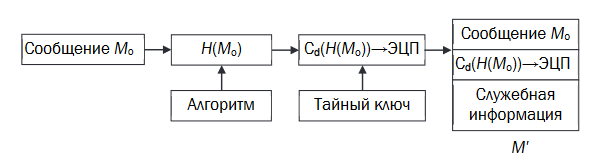


Рисунок 1 – Алгоритм цифровой подписи электронного документа

Важное свойство цифровой подписи заключается в том, что ее может проверить (верифицировать) каждый, кто имеет доступ к открытому ключу ее автора.

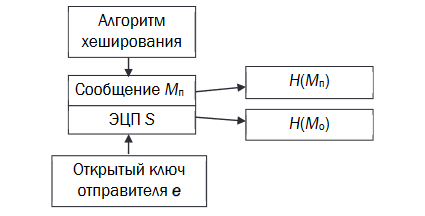


Рисунок 2 – Алгоритм проверки цифровой подписи

Если в результате устанавливается равенство хешей: *Н*(*М*п) = *Н*(*М*о), то принимается решение о подлинности подписи и целостности документа *М*п, т. е. это также означает, что *М*п = *М*о.

ЭЦП на основе алгоритма RSA. Рассмотрим случай, когда сообщение *М*o подписывается и передается в открытом (незашифрованном) виде.

Этот случай соответствует схеме и операциям, представленным на рисунке 1 и рисунке 2. А сама подпись вычисляется по следующей формуле:

S ≡ (*H*(*M*o)) d mod *n*,

где *d* и *n* – элементы тайного ключа отправителя. Передаваемое сообщение *М*' = *М*o || *S*.

Соответственно, операция расшифрования на приемной стороне (получатель анализирует *М*п || *S*) будет производиться в соответствии с формулой с известной модификацией ключей:

*H*(*M*o) ≡ (*S*)е mod *n*,

где *e* и *n* – элементы открытого ключа отправителя.

ЭЦП на основе алгоритма Эль-Гамаля. Ключевая информация отправителя для ЭЦП создается обычным способом, как принято стандартом алгоритма: открытый ключ (*y*, *g* и *р*) и тайный ключ *х*. Чтобы подписать сообщение *М*о, обладатель используемых для ЭЦП ключей должен выбрать случайное число *k*, взаимно простое с (*р* – 1). Затем вычисляется числа *а* и *b*, являющиеся цифровой подписью (*S* = {*a*, *b*}):

*a* ≡ *g*k mod *p*;

для вычисления *b* с помощью расширенного алгоритма Евклида решается уравнение

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

Получателю отправляется сообщение *М*' = *М*о || *S*.

Для верификации подписи вычисляется хеш полученного сообщения *Н*(*М*п) = *h*. Далее нужно убедиться, что выполняется равенство

*y*a ⋅ *a*b ≡ *g*h mod *p*.

Если равенство выполняется, подпись верифицируется.

ЭЦП на основе алгоритма Шнорра. Рассматриваемая схема является основой стандарта ЭЦП в Беларуси. Алгоритм ЭЦП К. Шнорра (К. Schnorr) является вариантом алгоритма ЭЦП Эль-Гамаля.

Одной из особенностей ЭЦП Эль-Гамаля является то, что число *p* должно быть очень большим, чтобы сделать действительно трудной проблему дискретного логарифма. Рекомендуемая длина *p* должна составлять по крайней мере 1024 бита. Чтобы уменьшить размер подписи, Шнорр предложил новую схему, но с уменьшенным размером подписи.

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

*g*q ≡ 1 mod *p*.

Числа *p*, *g*, *q* являются открытыми и могут применяться группой пользователей.

Выбирается число *х* < q (*х* является тайным ключом) и вычисляется последний элемент открытого ключа:

*y* ≡ *g*q–х mod *p*.

Секретный ключ имеет длину не менее 160 битов.

Для подписи сообщения *М*о выбирается случайное число *k* (1 < *k* < *q*) и вычисляет параметр *а*:

*а* ≡ *g*k mod *p*.

Далее вычисляется хеш от конкатенации сообщения *М*о и числа *а*:   
*h* = *H*(*M*o || *a*). Обратим внимание, что хэш-функция непосредственно не применяется к сообщению. Создается хеш-образ подписываемого сообщения, спереди присоединенного к числу *а*. Далее вычисляется значение *b*:

*b* ≡ (*k* + *x* ⋅ *h*) mod *q*.

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

Для проверки подписи получатель вычисляет

*Х* ≡ *g*b ⋅ *y*h (mod *p*).

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

## Практическое задание

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.

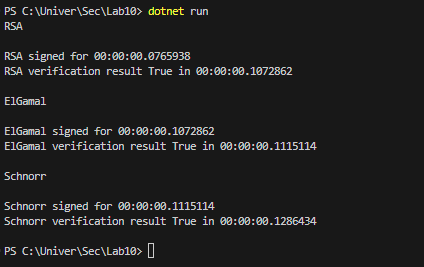


Рисунок 3 – Результат программы цифровой подписи

На рисунке выше можно заметить время подписи сообщения для каждого сообщения, а также время проверки подписи.

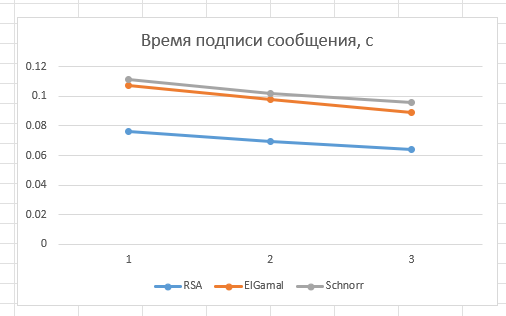


Рисунок 4 – График времени подписи сообщения различными алгоритмами

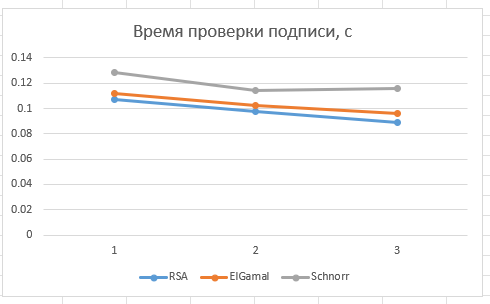


Рисунок 5 – График времени проверки подписи различными алгоритмами

Вывод: в ходе этой лабораторной работы я изучил основы ЭЦП, для чего применяется и какие алгоритмы ЭЦП популярны сегодня. Реализовал программу на языке *C*#, которая подписывает сообщения тремя разными алгоритмами: RSA, ElGamal и Schnorr. Оценил скорость подписи сообщения и скорость проверки подписи каждого из них.

# Листинг «Лабораторная работа №10»

using System.Numerics;

using Lab10.Hash;

using System.Text;

using Lab10.Encoders;

using Lab10.ExtensionMethods;

using Lab10.Generators.PrimeGenerator;

using System.Runtime.Intrinsics.Arm;

namespace Lab10.RSA

{

public class RSA

{

public long N { get; private set; } // module

public long E { get; private set; } // public key

private readonly long D = 0; // private key

public RSA()

{

N = (long)PrimeGenerator.GetRandomPrime() \* PrimeGenerator.GetRandomPrime();

E = N.EilersFunction().GetCoprimeBelow();

D = E.GetReversed(N.EilersFunction());

}

public long[] AsciiEncrypt(string message)

{

return Encoding.ASCII.GetBytes(message)

.Select(b => (long)BigInteger.ModPow(b, E, N))

.ToArray();

}

public string AsciiDecrypt(long[] message)

{

return Encoding.ASCII.GetString(

message.Select(m => (byte)BigInteger.ModPow(m, D, N))

.ToArray()

);

}

public long[] Base64Encrypt(string message)

{

return Encoding.UTF8.GetBytes(Base64.Encode(message))

.Select(b => (long)BigInteger.ModPow(b, E, N))

.ToArray();

}

public string Base64Decrypt(long[] message)

{

return Base64.Decode(

Encoding.UTF8.GetString(

message.Select(m => (byte)BigInteger.ModPow(m, D, N)).ToArray()

)

);

}

public SignedMessage.SignedMessage Sign(string message)

{

return new SignedMessage.SignedMessage()

{

Message = message,

Hash = Base64Encrypt(SHA1.Hash(message))

};

}

public bool Verify(SignedMessage.SignedMessage signedMessage)

{

var messageHash = SHA1.Hash(signedMessage.Message!);

var decryptedHash = Base64Decrypt(signedMessage.Hash);

return messageHash == decryptedHash;

}

}

}

Листинг 1 – Код алгоритма RSA

using System.Numerics;

using System.Text;

using Lab10.Encoders;

using Lab10.ExtensionMethods;

using Lab10.Generators.PrimeGenerator;

using Lab10.Hash;

namespace Lab10.ElGamal

{

public class ElGamal

{

public long P { get; private set; } // part of public key (random prime number)

public long Q { get; private set; } // part of public key (primitive root of P)

public long Y { get; private set; } // part of public key

private readonly long X; // private key

public ElGamal()

{

P = PrimeGenerator.GetRandomPrime();

Q = P.GetPrimitiveRoot();

X = new Random().NextInt64(2, P - 1);

Y = (long)BigInteger.ModPow(Q, X, P);

}

public ElGamalEncryptedByte[] AsciiEncrypt(string message)

{

return Encoding.ASCII.GetBytes(message)

.Select(b => {

long K = PrimeGenerator.GetRandomPrime();

return new ElGamalEncryptedByte

{

A = (long)BigInteger.ModPow(Q, K, P),

B = (long)(BigInteger.ModPow(Y, K, P) \* b % P)

};

})

.ToArray();

}

public string AsciiDecrypt(ElGamalEncryptedByte[] message)

{

return Encoding.ASCII.GetString(

message

.Select(b => (byte)(BigInteger.ModPow(b.A, P - 1 - X, P) \* b.B % P))

.ToArray()

);

}

public ElGamalEncryptedByte[] Base64Encrypt(string message)

{

return Encoding.UTF8.GetBytes(Base64.Encode(message))

.Select(b => {

long K = PrimeGenerator.GetRandomPrime();

return new ElGamalEncryptedByte

{

A = (long)BigInteger.ModPow(Q, K, P),

B = (long)(BigInteger.ModPow(Y, K, P) \* b % P)

};

})

.ToArray();

}

public string Base64Decrypt(ElGamalEncryptedByte[] message)

{

return Base64.Decode(

Encoding.UTF8.GetString(

message

.Select(b => (byte)(BigInteger.ModPow(b.A, P - 1 - X, P) \* b.B % P))

.ToArray()

)

);

}

public SignedMessage.SignedMessage Sign(string message)

{

return new SignedMessage.SignedMessage

{

Message = message,

Hash = Base64Encrypt(SHA1.Hash(message))

};

}

public bool Verify(SignedMessage.SignedMessage signedMessage)

{

var messageHash = SHA1.Hash(signedMessage.Message!);

var decryptedHash = Base64Decrypt(signedMessage.Hash);

return messageHash == decryptedHash;

}

public record ElGamalEncryptedByte

{

public long A { get; set; }

public long B { get; set; }

}

}}

Листинг 2 – Код алгоритма ElGamal

using System.Numerics;

using Lab10.ExtensionMethods;

using Lab10.Generators.PrimeGenerator;

using Lab10.Hash;

namespace Lab10.Schnorr

{

public class Schnorr

{

public long P { get; private set; }

public long Q { get; private set; }

public long G { get; private set; }

public long Y { get; private set; }

private readonly long X;

public Schnorr()

{

P = PrimeGenerator.GetRandomPrime();

Q = (P - 1).PrimeDivider();

G = Q.GetRootOfPow(P);

X = new Random().NextInt64(2, Q - 1);

Y = (long)BigInteger.ModPow(G, Q - X, P);

}

public SignedMessage.SignedMessage Sign(string message)

{

var k = new Random().NextInt64(2, Q - 1);

var a = (long)BigInteger.ModPow(G, k, P);

var hashedMessage = SHA1.Hash(message + a, false);

var b = (long)BigInteger.ModPow(k + X \* hashedMessage, 1, Q);

return new SignedMessage.SignedMessage

{

Message = message,

Hash = new {

h = hashedMessage,

b

}

};

}

public bool Verify(SignedMessage.SignedMessage signedMessage)

{

var gParam = BigInteger.ModPow(G, signedMessage.Hash.b, P);

var yParam = BigInteger.ModPow(Y, signedMessage.Hash.h, P);

var X = (long)(BigInteger.Multiply(gParam, yParam) % P);

return SHA1.Hash(signedMessage.Message + X, false) == signedMessage.Hash.h;

}

}

}

Листинг 3 – Код алгоритма Schnorr

namespace Lab10.SignedMessage

{

public class SignedMessage

{

public string? Message { get; set; }

public required dynamic Hash { get; set; }

}

}

Листинг 4 – Структура класса подписанного сообщения