МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Отчет по лабораторной работе №12

Исследование алгоритмов генерации и верификации электронной цифровой подписи

Выполнил студентка: Авсюкевич П.В.

ФИТ 3 курса, 5 группа

Проверила: Савельева М. Г.

Минск 2024

Задание 1. Разработать приложение, реализующее генерацию и верификацию ЭЦП на основе алгоритмов RSA, Эль-Гамаля и Шнорра.

Рассмотрим процесс генерации для алгоритма RSA.

Шаги для алгоритма генерации подписи: выбор простых чисел p, q; выбор случайного числа e, взаимно простого с функцией Эйлера *ф*(*n*)=(*p*-1)(*q*-1); нахождение числа d, такого что *ed* =1 *mod* (*p*-1)(*q*-1); вычисление хеш-образа сообщения *h=H*(*M*); вычисление ЭЦП: *S* = *hd* mod *n*.

Далее полученный открытый ключ *{*e, n*}* и письмо с ЭЦП *{m, sign}* будут отправлены получателю. Реализация представлена на листинге 1.1.

|  |
| --- |
| public List<string> Encode(string hash, int publicKey, int modulus)  {  var encodedResult = new List<string>();  BigInteger encodedCharacter;  foreach (char character in hash)  {  int index = Array.IndexOf(ValidCharacters, character);  encodedCharacter = new BigInteger(index);  encodedCharacter = BigInteger.Pow(encodedCharacter, publicKey);  BigInteger modulusBigInt = new BigInteger(modulus);  encodedCharacter = encodedCharacter % modulusBigInt;  encodedResult.Add(encodedCharacter.ToString());  }  return encodedResult; } |

Листиниг 1.1 – Генерация ЭЦП

Для операции верификации на стороне получателя требуется выполнить действия: вычислить *h* = *Se* mod n; сравнить значение выше с полученным *h*=*H*(*M*)*.* Если значения совпали, то подпись верифицирована. Реализация представлен на листинге 1.2

|  |
| --- |
| public string Decode(List<string> input, int privateKey, int modulus){ string decodedResult = "";  foreach (string item in input){  decodedCharacter = new BigInteger(Convert.ToDouble(item));  decodedCharacter = BigInteger.Pow(decodedCharacter, privateKey);  BigInteger modulusBigInt = new BigInteger(modulus);  decodedCharacter = decodedCharacter % modulusBigInt;  int index = Convert.ToInt32(decodedCharacter.ToString());  decodedResult += ValidCharacters[index].ToString();  }  return decodedResult;} |

Листиниг 1.2 – Генерация хеша получателя

Результат работы программы представлен на рисунке 1.1.

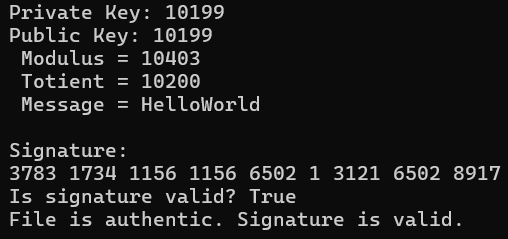


Рисунок 1.1 – Результат работы программы

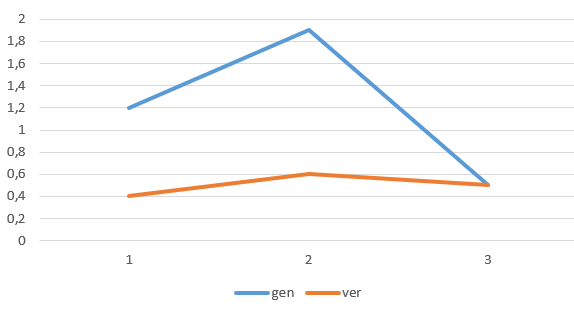


Рисунок 1.2 – График времени генерации/верификации подписи

Рассмотрим процесс генерации для алгоритма Эль-Гамаля.

Для генерации подписи используется программный код, представленный на листинге 1.3.

|  |
| --- |
| public static Tuple<int, int> GenerateSignature(int p, int g, int x, int k, int message)  {  int y = (int)BigInteger.ModPow(g, x, p);  int a = (int)BigInteger.ModPow(g, k, p);  int m = p - 1;  int b = (int)BigInteger.ModPow((kInv \* (message - (x \* a) % m) % m) % m, 1, m);  return Tuple.Create(a, b);  } |

Листинг 1.3 – Генерация подписи

Для верификации подлинности полученного сообщения необходимо проверить равенство и если оно выполняется, подпись верифицированна и подлинна. Реализация представлена на листинге 1.4.

|  |
| --- |
| public static bool VerifySignature(int p, int g, int y, int a, int b, int message){  BigInteger ya = BigInteger.ModPow(y, a, p);  BigInteger ab = BigInteger.ModPow(a, b, p);  BigInteger pr1 = BigInteger.ModPow(ya \* ab, 1, p);  BigInteger pr2 = BigInteger.ModPow(g, message, p);  return pr1 == pr2;} |

Листинг 1.4– Верификация полученного сообщения

Результат работы программы представлен на рисунке 1.3.

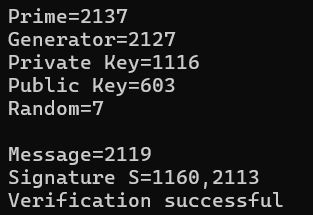


Рисунок 1.3 – Результат работы программы

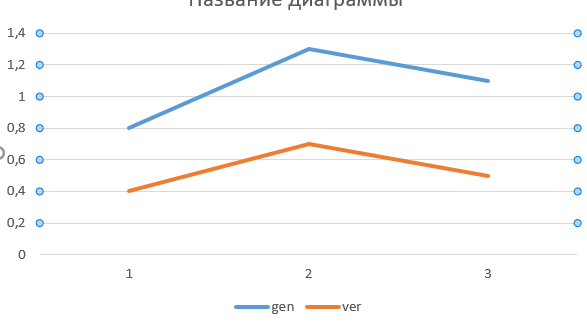


Рисунок 1.4 – График времени генерации/верификации подписи

Рассмотрим процесс генерации для алгоритма Шнорра.

Алгоритм генерации ключевой информации представлен на листинге 1.5.

|  |
| --- |
| public static BigInteger CalculateMd5Hash(string input)  {  var md5 = MD5.Create();  var inputBytes = Encoding.ASCII.GetBytes(input);  var hash = md5.ComputeHash(inputBytes);  return new BigInteger(hash.Concat(new byte[] { 0 }).ToArray());  } |

Листинг 1.5 – Генерация ключевой информации

Для собственной подписи сообщения требовалось реализовать функцию, представленную на листинге 1.6.

|  |
| --- |
| public static bool VerifySignature(string message, BigInteger publicKey, BigInteger generator, BigInteger prime, BigInteger signature)  {  BigInteger hash = ElGamal.CalculateMd5Hash(message + publicKey.ToString());  BigInteger leftPart = BigInteger.ModPow(generator, signature, prime);  BigInteger rightPart = (publicKey \* BigInteger.ModPow(generator, hash, prime)) % prime;  BigInteger calculatedHash = ElGamal.CalculateMd5Hash(message + leftPart.ToString());  return hash == calculatedHash;  } |

Листинг 1.6 – Реализация подписи сообщения

Результат работы программы представлен на рисунке 1.5.

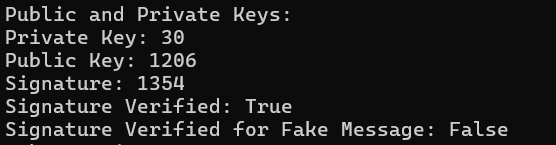


Рисунок 1.5 – Результат работы программы

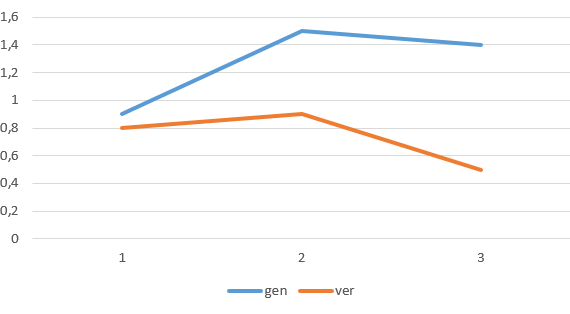


Рисунок 1.6 – График времени генерации/верификации подписи

В процессе выполнения лабораторной работы были углубленно изучены алгоритмы генерации и верификации электронной цифровой подписи (ЭЦП), а также получены практические навыки их реализации. Основными алгоритмами, рассмотренными в рамках работы, были алгоритмы Эль-Гамаля и Шнорра.