Министерство образования и науки Российской Федерации

Новосибирский государственный технический университет

Кафедра прикладной математики

Основы криптографии

Лабораторная работа №3

Факультет: прикладной математики и информатики

Группа: ПМ-63

Студенты: Шепрут И.М.

Кожекин М.В.

Утюганов Д.С.

Преподаватель: Ступаков И.М.

Новосибирск

2018

1. Цель работы

​Изучить вычисление хеш сумм и работу с ECDH и HMAC в языке C#

1. Ход работы

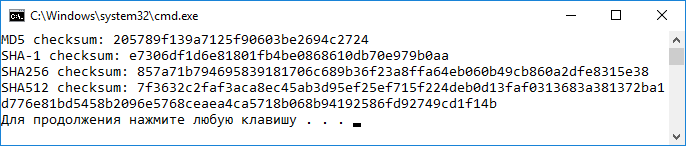
**Задача 1**

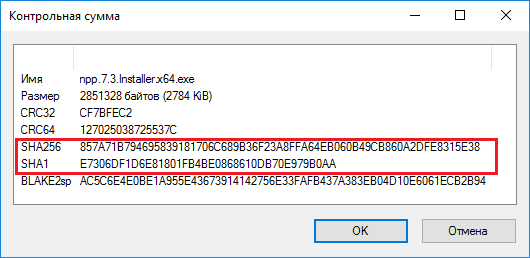
Хеш функции. Взять файл (лучше с известными из надежного источника хешами) и вычислить хешы по алгоритмам MD5 и SHA-256. Использовать класс HashAlgorithm

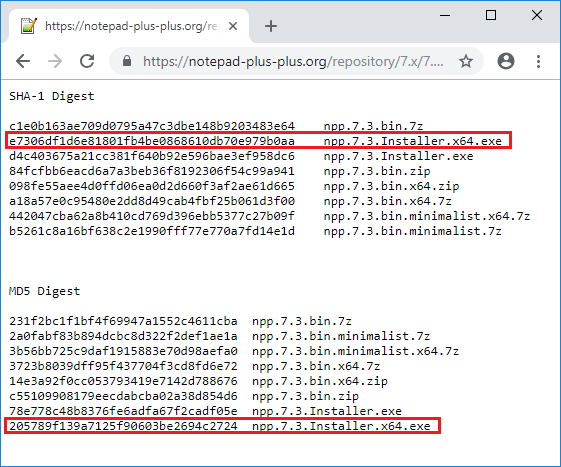
**Checksum.cs**

|  |
| --- |
| using System;  using System.IO;  using System.Security.Cryptography;  class Checksum  {  static string CalculateMD5(string filename)  {  using (var md5 = MD5.Create())  {  using (var stream = File.OpenRead(filename))  {  var hash = md5.ComputeHash(stream);  return BitConverter.ToString(hash).Replace("-", "").ToLowerInvariant();  }  }  }  static string CalculateSHA1(string filename)  {  using (var sha1 = SHA1.Create())  {  using (var stream = File.OpenRead(filename))  {  var hash = sha1.ComputeHash(stream);  return BitConverter.ToString(hash).Replace("-", "").ToLowerInvariant();  }  }  }  static string CalculateSHA256(string filename)  {  using (var sha256 = SHA256.Create())  {  using (var stream = File.OpenRead(filename))  {  var hash = sha256.ComputeHash(stream);  return BitConverter.ToString(hash).Replace("-", "").ToLowerInvariant();  }  }  }  static string CalculateSHA512(string filename)  {  using (var sha512 = SHA512.Create())  {  using (var stream = File.OpenRead(filename))  {  var hash = sha512.ComputeHash(stream);  return BitConverter.ToString(hash).Replace("-", "").ToLowerInvariant();  }  }  }  static void CalculateCheckSums(string filename)  {  Console.WriteLine("MD5 checksum: " + CalculateMD5(filename));  Console.WriteLine("SHA-1 checksum: " + CalculateSHA1(filename));  Console.WriteLine("SHA256 checksum: " + CalculateSHA256(filename));  Console.WriteLine("SHA512 checksum: " + CalculateSHA512(filename));  }  static void Main(string[] args)  {  // Task 1  CalculateCheckSums("npp.7.3.Installer.x64.exe");  }  } |

**Проверим работу программы, скачав установщик Notepad++ 7.3 и сверим его контрольную сумму с сайтом и сторонней программой:**







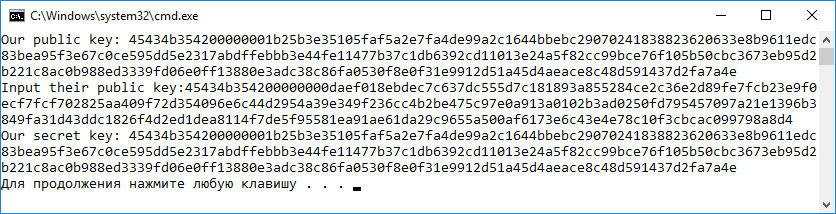
Все контрольные суммы совпали.

**Задача 2**

Используя класс ​ECDiffieHellman​ реализовать обмен ключами.

**Exchange.cs**

|  |
| --- |
| using System;  using System.IO;  using System.Security.Cryptography;  using System.Linq;  class Program  {  static byte[] StringToByteArray(string hash)  {  return Enumerable.Range(0, hash.Length)  .Where(x => x % 2 == 0)  .Select(x => Convert.ToByte(hash.Substring(x, 2), 16))  .ToArray();  }  static void keysExchenge()  {  byte[] inputBuffer = new byte[1024];  Stream inputStream = Console.OpenStandardInput(inputBuffer.Length);  Console.SetIn(new StreamReader(inputStream, Console.InputEncoding, false, inputBuffer.Length));  byte[] OurPublicKey;  byte[] TheirPublicKey;  using (var ecd = new ECDiffieHellmanCng())  {  ecd.KeyDerivationFunction = ECDiffieHellmanKeyDerivationFunction.Hash;  ecd.HashAlgorithm = CngAlgorithm.Sha256;  OurPublicKey = ecd.PublicKey.ToByteArray();  Console.WriteLine("Our public key: {0}", BitConverter.ToString(OurPublicKey).Replace("-", "").ToLowerInvariant());  Console.Write("Input their public key:");  string str = Console.ReadLine();  TheirPublicKey = StringToByteArray(str);  byte[] sendKey = ecd.DeriveKeyMaterial(CngKey.Import(TheirPublicKey, CngKeyBlobFormat.EccPublicBlob));  Console.WriteLine("Our secret key: {0}", BitConverter.ToString(OurPublicKey).Replace("-", "").ToLowerInvariant());  }  }  static void Main(string[] args)  {  // Task 2  keysExchenge();  }  } |



**Задача 3**

Используя класс ​HMAC​ и полученный секретный ключ, реализовать генерацию и валидацию подписи сообщения.

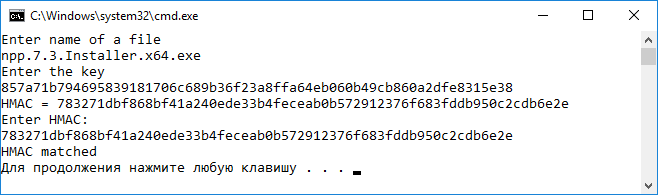
**HMAC.cs**

|  |
| --- |
| using System;  using System.IO;  using System.Security.Cryptography;  using System.Linq;  using System.Collections;  class Program  {  static byte[] StringToByteArray(string hash)  {  return Enumerable.Range(0, hash.Length)  .Where(x => x % 2 == 0)  .Select(x => Convert.ToByte(hash.Substring(x, 2), 16))  .ToArray();  }  static byte[] SignFile(string filename, byte[] key)  {  using (var HMAC = new HMACSHA256(key))  {  using (var file = new FileStream(filename, FileMode.Open))  {  return HMAC.ComputeHash(file);  }  }  }  static bool ValidateFile(string filename, byte[] key, byte[] expectedHash)  {  byte[] realHash;  using (var HMAC = new HMACSHA256(key))  {  using (var file = new FileStream(filename, FileMode.Open))  {  realHash = HMAC.ComputeHash(file);  }  }  return StructuralComparisons.StructuralEqualityComparer.Equals(realHash, expectedHash);  }  static void SignAndValidate()  {  Console.WriteLine("Enter name of a file");  string filename = Console.ReadLine();  Console.WriteLine("Enter the key");  byte[] key = StringToByteArray(Console.ReadLine());  byte[] hmac = SignFile(filename, key);  Console.WriteLine("HMAC = {0}", BitConverter.ToString(hmac).Replace("-", "").ToLowerInvariant());    Console.WriteLine("Enter HMAC:");  hmac = StringToByteArray(Console.ReadLine());  bool isVerified = ValidateFile(filename, key, hmac);  if (isVerified)  Console.WriteLine("HMAC matched");  else  Console.WriteLine("HMAC didn't matched");  }  static void Main(string[] args)  {  // Task 3  SignAndValidate();  }  } |

Файл: npp.7.3.Installer.x64.exe

Ключ (256 бит): 857a71b794695839181706c689b36f23a8ffa64eb060b49cb860a2dfe8315e38 HMAC: 783271db0368bf4la246ede33b4feceabObS72912376f6S3fddb9S0c2cdb6e2e

**Результат:**



**Задача 4 (бонусная)**

Написать собственный HMAC.

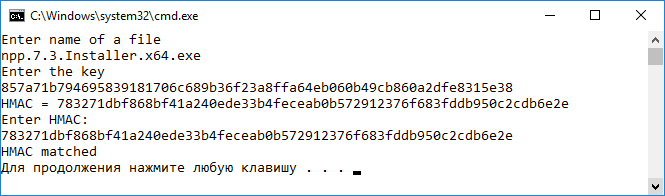
**MyHMAC.cs**

|  |
| --- |
| using System;  using System.IO;  using System.Security.Cryptography;  using System.Linq;  using System.Collections;  public class MyHMAC : IDisposable  {  public MyHMAC(byte[] new\_key)  {  key = new\_key;  ipad = 0x36;  opad = 0x5C;  blockSize = 64;  }  private byte[] StringToByteArray(string str)  {  return Enumerable.Range(0, str.Length)  .Where(x => x % 2 == 0)  .Select(x => Convert.ToByte(str.Substring(x, 2), 16))  .ToArray();  }  private byte[] XOR(byte[] buffer1, byte buffer2)  {  byte[] result = new byte[blockSize];  for (int i = 0; i < blockSize; i++)  result[i] = (byte)(buffer1[i] ^ buffer2);  return result;  }  public byte[] ComputeHash(Stream s)  {  byte[] text = StreamToByteArray(s);  byte[] K = key;  if (K.Length > blockSize)  K = H(K);  if (K.Length < blockSize)  {  byte[] newArray = new byte[blockSize];  K.CopyTo(newArray, 0);  for (int i = K.Length; i < blockSize; ++i)  newArray[i] = 0;  K = newArray;  }  return H(concat(XOR(K, opad), H(concat(XOR(K, ipad), text))));  }  private static byte[] concat(byte[] a, byte[] b)  {  byte[] c = new byte[a.Length + b.Length];  a.CopyTo(c, 0);  b.CopyTo(c, a.Length);  return c;  }  private byte[] H(byte[] arr)  {  using (var sha256 = SHA256.Create())  return sha256.ComputeHash(arr);  }  private byte[] StreamToByteArray(Stream input)  {  using (MemoryStream ms = new MemoryStream())  {  input.CopyTo(ms);  return ms.ToArray();  }  }  public void Dispose()  {  }  private byte ipad;  private byte opad;  private int blockSize;  private byte[] key;  }  class Program  {  static byte[] StringToByteArray(string hash)  {  return Enumerable.Range(0, hash.Length)  .Where(x => x % 2 == 0)  .Select(x => Convert.ToByte(hash.Substring(x, 2), 16))  .ToArray();  }  static byte[] MySignFile(string filename, byte[] key)  {  using (MyHMAC HMAC = new MyHMAC(key))  {  using (var file = new FileStream(filename, FileMode.Open))  {  return HMAC.ComputeHash(file);  }  }  }  static bool MyValidateFile(string filename, byte[] key, byte[] expectedHash)  {  byte[] realHash;  using (var HMAC = new MyHMAC(key))  {  using (var file = new FileStream(filename, FileMode.Open))  {  realHash = HMAC.ComputeHash(file);  }  }  return StructuralComparisons.StructuralEqualityComparer.Equals(realHash, expectedHash);  }  static void MySignAndValidate()  {  Console.WriteLine("Enter name of a file");  string filename = Console.ReadLine();  Console.WriteLine("Enter the key");  string str = Console.ReadLine();  byte[] key = StringToByteArray(str);  byte[] hmac = MySignFile(filename, key);  Console.WriteLine("HMAC = {0}", BitConverter.ToString(hmac).Replace("-", "").ToLowerInvariant());  Console.WriteLine("Enter HMAC:");  hmac = StringToByteArray(Console.ReadLine());  bool isVerified = MyValidateFile(filename, key, hmac);  if (isVerified)  Console.WriteLine("HMAC matched");  else  Console.WriteLine("HMAC didn't matched");  }  static void Main(string[] args)  {  // Task 4  MySignAndValidate();  }  } |

Файл: npp.7.3.Installer.x64.exe

Ключ (256 бит): 857a71b794695839181706c689b36f23a8ffa64eb060b49cb860a2dfe8315e38 HMAC: 783271db0368bf4la246ede33b4feceabObS72912376f6S3fddb9S0c2cdb6e2e

**Результат:**



1. Выводы

В ходе выполнения 3 лабораторной работы было изучена работа классов ECDH и HMAC из библиотеки System.Security.Criptography языка C#. Также был написан собственный HMAC.