МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждения образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Практическая работа №8**

по дисциплине «Основы информационной безопасности»

на тему: Изучение стандартных средств для реализации приложений, использующих симметричное и ассиметричное шифрование с использованием библиотеки System.Security.Cryptography

Выполнил студент 2 курса 7 группы специальность ПОИБМС Володькин Никифор Дмитриевич

(Ф.И.О.)

Преподаватель Ржеутская Надежда Викентьевна

(Ф.И.О.)

**Цель: Изучить модель криптографии .NET Framework, основные классы и структуры данных, разработать приложение для шифрования файлов использующих симметричные и ассиметричные алгоритмы шифрования.**

**Теоретическое введение**

В .Net Framework присутствует пространство имён для выполнения криптографических операций под названием System.Security.Cryptography. Данное пространство имён предоставляет криптографические службы, включающие безопасное кодирование и декодирование данных, а также другие операции, такие как хеширование сообщений, генерация случайных чисел и проверка подлинности сообщений. Данная библиотека предоставляет доступ для использования различных реализаций алгоритмов в основном это программные интерфейсы CryptoApi (CAPI) и Cryptography Next Generation API (CNG API) помимо этого для некоторых алгоритмов возможно использование реализаций на основе OpenSsl.

CryptoAPI — интерфейс программирования приложений, который обеспечивает разработчиков Windows-приложений стандартным набором функций для работы с криптопровайдером. Входит в состав операционных систем Microsoft. Большинство функций CryptoAPI поддерживается, начиная с Windows 2000.

Cryptography Next Generation стала долгосрочной заменой CAPI. Данный набор интерфейсов поддерживает все алгоритмы предлагаемые CAPI а также другие алгоритмы перечисленные в своде правил Suite B Агентства национальной безопасности США [1]. Данный интерфейс поддерживает следующие длины ключей или размерность хеша.

• RSA 512 бит to 16384 бит, с шагом 64 бит

• DH 512 бит to 16384 бит, с шагом 64 бит

• DSA 512 бит to 1024 бит, с шагом 64 бит

• ECDSA P-256, P-384, P-521 (NIST Curves)

• ECDH P-256, P-384, P-521 (NIST Curves)

• MD2 128 бит

• MD4 128 бит

• MD5 128 бит

• SHA-1 160 бит

• SHA-256 256 бит

• SHA-384 384 бит

• SHA-512 512 бит

Рассматривая структуру наследования для симметричных алгоритмов в .Net стоит упомянуть что SymmetricAlgorithm является абстрактным классом, от который наследуют абстрактные классы для реализаций каждого из алгоритмов. В свою очередь каждая из реализации алгоритма является производной от абстрактного класса алгоритма. Ниже представлена структура наследования.

SymmetricAlgorithm

Aes

AesCng

AesManeged

AesCryptoServiceProvider

Des

DesCng

DesManeged

DesCryptoServiceProvider

TripleDes

TripleDesCng

TripleDesManeged

TripleDesCryptoServiceProvider

Данная структура наследования повторяется для каждого из трех типов поддерживаемых криптографических операций: SymmetricAlgorithm, AsymmetricAlgorithm, HashAlgorithm.

**Выполнение работы**

Было написано следующее приложение на языке C#, реализующее алгоритм TripleDES и SHA384:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Security.Cryptography;

namespace Lab\_8

{

class TrippleDESCryptography

{

public static byte[] Encrypt(string Data, byte[] Key, byte[] IV)

{

try

{

MemoryStream mStream = new MemoryStream();

CryptoStream cStream = new CryptoStream(mStream,

new TripleDESCryptoServiceProvider().CreateEncryptor(Key, IV),

CryptoStreamMode.Write);

byte[] toEncrypt = new ASCIIEncoding().GetBytes(Data);

cStream.Write(toEncrypt, 0, toEncrypt.Length);

cStream.FlushFinalBlock();

byte[] ret = mStream.ToArray();

cStream.Close();

mStream.Close();

return ret;

}

catch (CryptographicException e)

{

Console.WriteLine("A Cryptographic error occurred: {0}", e.Message);

return null;

}

}

public static string Decrypt(byte[] Data, byte[] Key, byte[] IV)

{

try

{

MemoryStream msDecrypt = new MemoryStream(Data);

CryptoStream csDecrypt = new CryptoStream(msDecrypt,

new TripleDESCryptoServiceProvider().CreateDecryptor(Key, IV),

CryptoStreamMode.Read);

byte[] fromEncrypt = new byte[Data.Length];

csDecrypt.Read(fromEncrypt, 0, fromEncrypt.Length);

return new ASCIIEncoding().GetString(fromEncrypt);

}

catch (CryptographicException e)

{

Console.WriteLine("A Cryptographic error occurred: {0}", e.Message);

return null;

}

}

}

}

using System.Security.Cryptography;

using System.Text;

using Lab\_8;

try

{

TripleDESCryptoServiceProvider tripleDES = new TripleDESCryptoServiceProvider();

tripleDES.KeySize = 192;

string message = "Wolodkin";

byte[] data = TrippleDESCryptography.Encrypt(message, tripleDES.Key, tripleDES.IV);

File.WriteAllBytes(@"../../../encryptedMessage.txt", data);

File.WriteAllBytes(@"../../../key.txt", tripleDES.Key);

string decryptedMessage = TrippleDESCryptography.Decrypt(data, tripleDES.Key, tripleDES.IV);

Console.WriteLine(decryptedMessage);

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

try

{

string message = "Wolodkin";

byte[] hash = SHA384.HashData(new ASCIIEncoding().GetBytes(message));

File.WriteAllBytes(@"../../../hash.txt", hash);

}

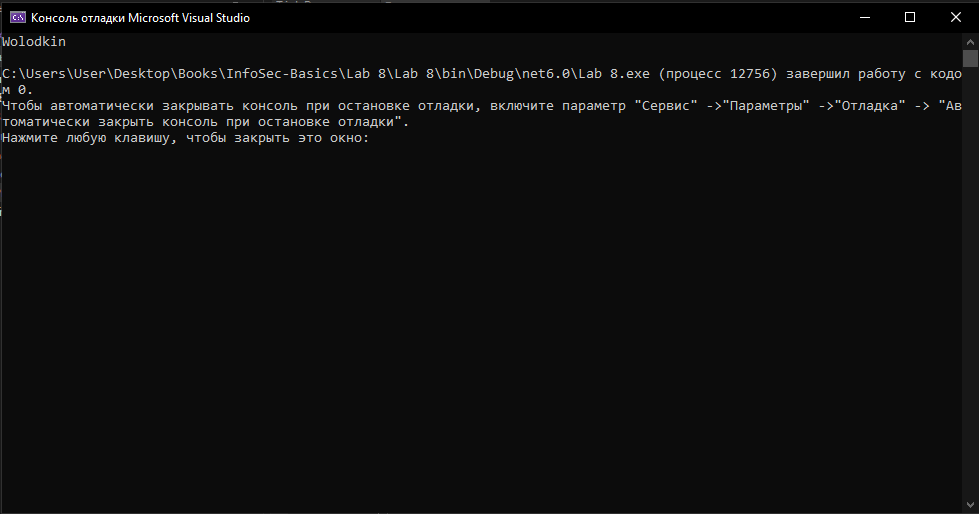
catch (Exception ex)

{

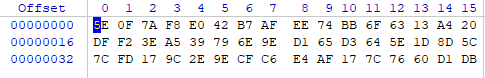
Console.WriteLine(ex.Message);

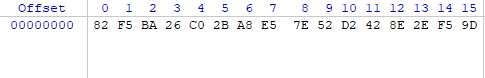
}

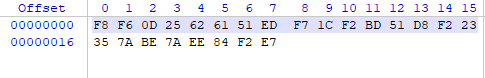
Результат выполнения программы:



С помощью HEX-редактора были изучены байтовые значения хеша, зашифрованных данных и ключа:







Было написано следующее приложение на языке C#, реализующее алгоритм ЭЦП:

using System;

using System.Security.Cryptography;

using System.Text;

class RSASign

{

static void Main()

{

try

{

UTF8Encoding ByteConverter = new UTF8Encoding();

Console.Write("Введите сообщение: ");

string dataString = Console.ReadLine();

byte[] originalData = ByteConverter.GetBytes(dataString);

byte[] signedData;

RSACryptoServiceProvider RSAalg = new RSACryptoServiceProvider();

RSAParameters Key = RSAalg.ExportParameters(true);

signedData = HashAndSignBytes(originalData, Key);

if (VerifySignedHash(originalData, signedData, Key))

{

Console.WriteLine("ЭЦП принадлежит этому сообщению");

}

else

{

Console.WriteLine("ЭЦП не принадлежит этому сообщению");

}

if (VerifySignedHash(ByteConverter.GetBytes(dataString + "1"), signedData, Key))

{

Console.WriteLine("ЭЦП принадлежит этому сообщению");

}

else

{

Console.WriteLine("ЭЦП не принадлежит этому сообщению");

}

signedData[0] = 0;

if (VerifySignedHash(originalData, signedData, Key))

{

Console.WriteLine("ЭЦП принадлежит этому сообщению");

}

else

{

Console.WriteLine("ЭЦП не принадлежит этому сообщению");

}

}

catch (ArgumentNullException)

{

Console.WriteLine("ЭЦП не проверено");

}

}

public static byte[] HashAndSignBytes(byte[] DataToSign, RSAParameters Key)

{

try

{

RSACryptoServiceProvider RSAalg = new RSACryptoServiceProvider();

RSAalg.ImportParameters(Key);

return RSAalg.SignData(DataToSign, SHA256.Create());

}

catch (CryptographicException e)

{

Console.WriteLine(e.Message);

return null;

}

}

public static bool VerifySignedHash(byte[] DataToVerify, byte[] SignedData, RSAParameters Key)

{

try

{

RSACryptoServiceProvider RSAalg = new RSACryptoServiceProvider();

RSAalg.ImportParameters(Key);

return RSAalg.VerifyData(DataToVerify, SHA256.Create(), SignedData);

}

catch (CryptographicException e)

{

Console.WriteLine(e.Message);

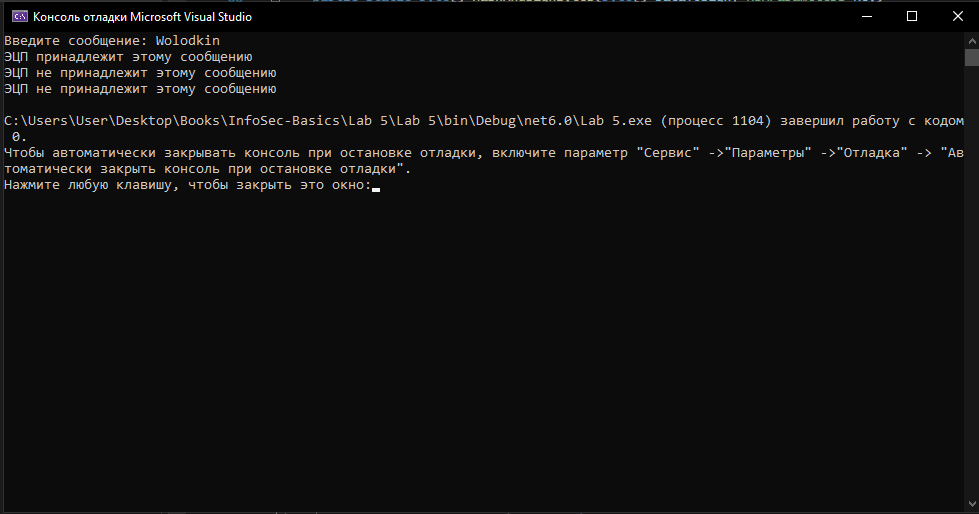
return false;

}

}

}

Результат выполнения программы:



**Вывод: изучены и закреплены умения реализации ЭЦП.**