Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Дисциплина «Основы информационной безопасности»

Отчёт по практическому занятию №8

**Изучение стандартных средств для реализации приложений, использующих симметричное и ассиметричное шифрование с использованием библиотеки** [**System.Security.Cryptography**](https://docs.microsoft.com/ru-ru/dotnet/api/system.security.cryptography)

Студент: Жук С.С.

ФИТ 2 курс 2 группа

Преподаватель: Ржеутская Н.В.

**Практическое занятие №8**

**«Изучение стандартных средств для реализации приложений, использующих симметричное и ассиметричное шифрование с использованием библиотеки** [**System.Security.Cryptography**](https://docs.microsoft.com/ru-ru/dotnet/api/system.security.cryptography)**»**

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

|  |
| --- |
| **Теоретические сведения** |

**Криптография в .Net**

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.

Рассмотрим часть кода для шифрования сообщения используя алгоритм Aes. Как представлено ниже мы создаём объект шифратор на основе созданного экземпляра криптографического объекта aesAlg. Далее мы создаём потоки для шифрования. Стоит отметить CryptoStream который определяет поток, связывающий потоки данный с криптографическим преобразованием.

ICryptoTransform encryptor = aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV);

// создание потоков используемого для шифрования

using (MemoryStream msEncrypt = new MemoryStream())

{

using (CryptoStream csEncrypt = new CryptoStream(msEncrypt, encryptor, CryptoStreamMode.Write))

//1-используемый поток, 2 - криптографические преобразование, 3 тип доступа

{

using (StreamWriter swEncrypt = new StreamWriter(csEncrypt))

{

swEncrypt.Write(plainText);

}

encrypted = msEncrypt.ToArray();

}

}

}

Дешифрование выполняется аналогично за исключением изменения типа доступа CryptoStream и изменения потока с записи на чтение.

ICryptoTransform decryptor = aesAlg.CreateDecryptor(aesAlg.Key, aesAlg.IV);

using (MemoryStream msDecrypt = new MemoryStream(cipherText))

{

using (CryptoStream csDecrypt = new CryptoStream(msDecrypt, decryptor, CryptoStreamMode.Read))

{

using (StreamReader srDecrypt = new StreamReader(csDecrypt))

{

plaintext = srDecrypt.ReadToEnd();

}

}

}

Также стоит упомянуть что сложность большинства алгоритмов шифрование имеет сильную зависимость от длины ключа. Изменение длины ключа возможно в большинстве алгоритмов как симметричного, так и ассиметричного шифрования. Для того что бы изменить размер ключа после инициализации алгоритма необходимо изменить свойство KeySize, что представлено в листинге ниже.

using (TripleDES myDes = TripleDES.Create())

{

myDes.KeySize = 128;

EncryptStringToFile\_DES(route, "2 keys usedE.txt", myDes.Key, myDes.IV);

DecryptStringToFile\_DES("2 keys usedE.txt", "2 key deceypted.txt", myDes.Key, myDes.IV);

}

Что касается ассиметричных алгоритмов то они представлены в данной структуре наследования.

AsymmetricAlgorithm

Rsa

RsaCng

RsaOpenSsl

RsaCryptoServiceProvider

Dsa

DsaCng

DsaOpenSsl

DsaCryptoServiceProvider

ECDiffieHellman

ECDiffieHellmanCng

ECDiffieHellmanOpenSsl

Шифрование и дешифрование для асимметричных алгоритмов выполняется проще из за встроенных функций Encrypt и Decrypt. Так же стоит помнить, что Rsa является блочным алгоритмов и если длина данных не совпадает с длиной блока, то данные нужно дополнить до длины блока. Для этого в данном пространстве имён можно использовать разные режимы заполнения.

using (RSA myRsa = RSA.Create())

{

string publickey = myRsa.ToXmlString(false); //получим открытый ключ

string privatekey = myRsa.ToXmlString(true);//получим закрытый ключ

byte[] encrypted = myRsa.Encrypt(data, RSAEncryptionPadding.Pkcs1);

byte[] decrypted = myRsa.Decrypt(encrypted, RSAEncryptionPadding.Pkcs1);

Console.WriteLine("До шифрования: {0}", original);

Console.WriteLine("Зашифрованное: {0}", System.Text.Encoding.UTF8.GetString(encrypted));

Console.WriteLine("После дешифровки: {0}", System.Text.Encoding.UTF8.GetString(decrypted));

}

Как представлено в коде выше после создания экземпляра Rsa возможна сразу же зашифровать и дешифровать данные использую автоматически сгенерированные ключи. Также код представленный выше содержит пример для получения открытого и закрытого ключа в виде строки в формате Xml.

<RSAKeyValue>

<Modulus>6yEjtrItcUq1hoA01xc63EW5/P99kstIobXsxPCUfUODRn2dazzcyhJ5Quhw1oHodlOMvtDN3xJdOTWDbH3xdQ==

</Modulus>

<Exponent>AQAB

</Exponent>

</RSAKeyValue>

Пример сохранённого в файл открытого ключа

<RSAKeyValue>

<Modulus>6yEjtrItcUq1hoA01xc63EW5/P99kstIobXsxPCUfUODRn2dazzcyhJ5Quhw1oHodlOMvtDN3xJdOTWDbH3xdQ==

</Modulus>

<Exponent>AQAB

</Exponent>

<P>9TsbWvgvA2OqPZxUZ96PomUG8rJk2T0SiH6chz65zkc=

</P>

<Q>9XR41YP8/CUoajovRPKWQZou3J23n3usp1acC3v9dGM=

</Q>

<DP>lnjHJ0GD72t3KUjETdu1fKrK4Z5u3RFpTtGJkD7/2b8=

</DP>

<DQ>9Qv2ppNCuigOlR7JmjflslDPgAklDN9XmyhoWT7L5qk=

</DQ>

<InverseQ>gSJ6G275fGFrEMqwSdgJYvmUQhnpCTcX0T3imIVQwOE=

</InverseQ>

<D>z0gMwu+6zehNtP/rFT9eXXd+qgHWAwYAxsapr0hjrzSXv1qS9QJJ+062YbdHc24WZagrKqABfOLQ3hfLXP3JdQ==

</D></RSAKeyValue>

Пример сохранённого в файл открытого ключа

Как видно на представленных выше файлах открытого и закрытого ключа объём ключей отличается, но также совпадают modulus (n) и Exponent (e). Остальные параметры в закрытом ключе совпадают с общепринятыми обозначениями кроме DP=d mod (p-1), DQ=d mode (q-1) и D обозначающего r или закрытый показатель степени.

Xеширования в .Net выполняется проще остальных операций из за отсутствия необходимости в обратном преобразовании. В данном пространстве имён поддерживаются следующие алгоритмы:

* MD5
* SHA256
* SHA384
* SHA512

Пример кода представлен ниже.

byte[] hashValue = new byte[256];

using (SHA256 mysha256 = SHA256.Create())

{

hashValue = mysha256.ComputeHash(File.ReadAllBytes("sha.txt"));

File.WriteAllBytes("hash.txt", hashValue);

}

Как видно из представленного кода после инициализации экземпляра для шифрования нужно вызвать только один метод ComputeHash который выполнит хеширование данных.

Пример захешированных данных представлен ниже

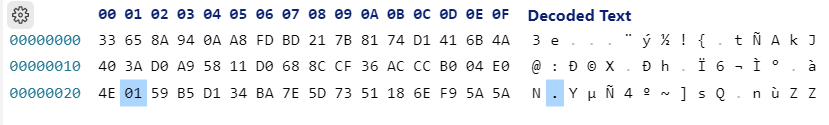


Рисунок 1

Список литературы:

[1] https://apps.nsa.gov/iaarchive/programs/iad-initiatives/cnsa-suite.cfm

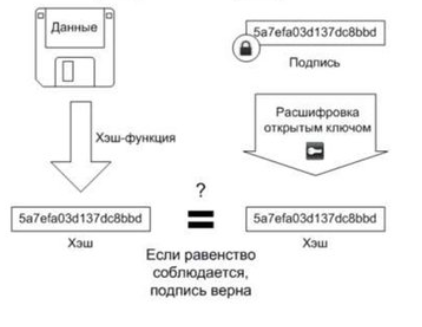


Рисунок 2

**Контрольные вопросы**

1. Основное назначение библиотеки System.Security.Cryptography?

System.Security.Cryptography - пространство имён для выполнения криптографических операций. 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.

1. Что из себя представляет интерфейс CryptoApi?

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.

1. Что такое Cryptography Next Generation?

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

**Задание для выполнения**

1. Ознакомиться с созданием криптографического приложения;
2. Выполнить шифрование, дешифрование и хеширование своей фамилии по указанным алгоритмам. Используя функции работы с файлами сохранить ключи шифрования, результаты шифрования и хеширования.
3. Для выше указанных алгоритмов используя Hex-редактор продемонстрировать ключи шифрования, зашифрованные и захешированные данные.

|  |  |  |
| --- | --- | --- |
| № | Алгоритм шифрования(размер ключа) | Алгоритм хеширования |
| 10 | DES | SHA384 |

**Исполнительская часть**

**Задание 1.**

Выполнить шифрование, дешифрование и хеширование (отсутствия необходимости в обратном преобразовании) своей фамилии по указанным алгоритмам. Используя функции работы с файлами сохранить ключи шифрования, результаты шифрования и хеширования.

using System.Text;

using System.Security.Cryptography;

static class Encryptor

{

public static string Encrypt(string plainText, out string keyBase64, out string vectorBase64)

{

using (DES DesAlgorithm = DES.Create())

{

Console.WriteLine($"Des режим шифрования : {DesAlgorithm.Mode}");

Console.WriteLine($"Des режим заполнения: {DesAlgorithm.Padding}");

Console.WriteLine($"Des размер ключа : {DesAlgorithm.KeySize}");

Console.WriteLine($"Des размер блока : {DesAlgorithm.BlockSize}");

keyBase64 = Convert.ToBase64String(DesAlgorithm.Key); // Key - ключ шифрования

vectorBase64 = Convert.ToBase64String(DesAlgorithm.IV); // IV - вектор инициализации нужен для того чтобы шифровать и расшифровывать данные

File.WriteAllText("encryptKey.txt", keyBase64);

File.WriteAllText("Ivector.txt", vectorBase64);

ICryptoTransform encryptor = DesAlgorithm.CreateEncryptor();

byte[] encryptedData;

// создание потоков используемого для шифрования

using (MemoryStream ms = new MemoryStream())

{

using (CryptoStream cs = new CryptoStream(ms, encryptor, CryptoStreamMode.Write))

{

//1-используемый поток, 2 - криптографические преобразование, 3 тип доступа

using (StreamWriter sw = new StreamWriter(cs))

{

sw.Write(plainText);

}

encryptedData = ms.ToArray();

}

}

File.WriteAllText("encryptedData.txt", Convert.ToBase64String(encryptedData));

return Convert.ToBase64String(encryptedData);

}

}

public static string Decrypte(string cipherText, string keyBase64, string vectorBase64)

{

using (DES DesAlgorithm = DES.Create())

{

DesAlgorithm.Key = Convert.FromBase64String(keyBase64);

DesAlgorithm.IV = Convert.FromBase64String(vectorBase64);

ICryptoTransform decryptor = DesAlgorithm.CreateDecryptor();

byte[] cipher = Convert.FromBase64String(cipherText);

using (MemoryStream ms = new MemoryStream(cipher))

{

using (CryptoStream cs = new CryptoStream(ms, decryptor, CryptoStreamMode.Read)) // режим для чтения данных

{

using (StreamReader sr = new StreamReader(cs))

{

return sr.ReadToEnd();

}

}

}

}

}

}

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Введите текст который вы хотите зашифровать:");

string message = Console.ReadLine();

string cipherText = Encryptor.Encrypt(message, out string keyBase64, out string vectorBase64);

Console.WriteLine("Зашифрованное сообщение: " + cipherText);

Console.WriteLine("Ключ шифрование в виде строки: " + keyBase64);

Console.WriteLine("Вектор инициализации (IV) в виде строки: " + vectorBase64);

Console.WriteLine("Расшифрованное сообщение: " + Encryptor.Decrypte(cipherText, keyBase64, vectorBase64));

Console.WriteLine("Вычесленый хэш из файла hash.txt: " + GetHash());

}

static string GetHash(string? input = null)

{

input = input ?? File.ReadAllText("hash.txt");

SHA384 sha = new SHA384Managed();

var hash = sha.ComputeHash(Encoding.UTF8.GetBytes(input));

File.WriteAllBytes("hashedInBytes.txt", hash);

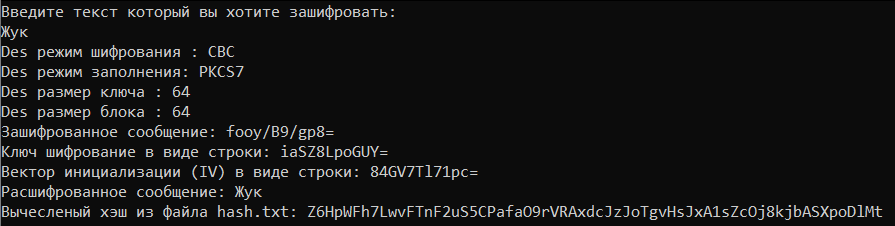
File.WriteAllText("hashedInString.txt", Convert.ToBase64String(hash));

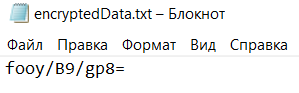
return Convert.ToBase64String(hash);

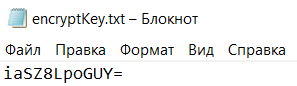
}

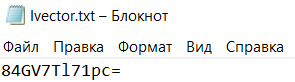
}

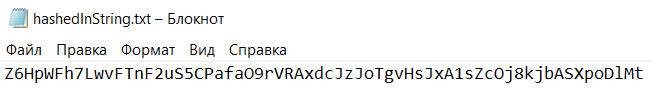
Вывод программы:





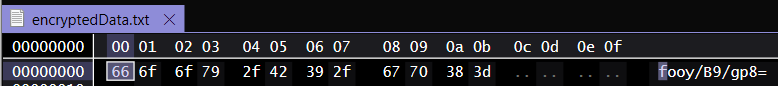


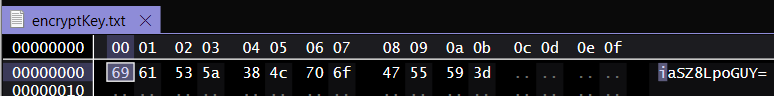


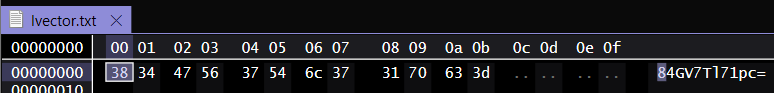


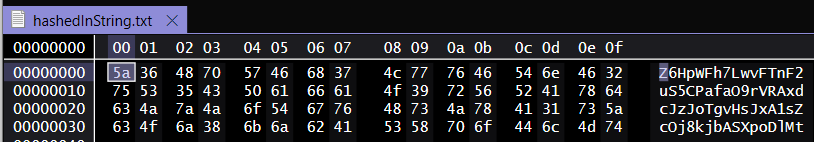
**Задание 2.**

Для выше указанных алгоритмов используя Hex-редактор продемонстрировать ключи шифрования, зашифрованные и захешированные данные.









**Вывод:** в ходе работы были изучены модели криптографии .NET Framework, основные классы и структуры данных, разобрались с разработкой приложения для шифрования файлов, использующих симметричные и ассиметричные алгоритмы шифрования.