Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Дисциплина «Основы защиты информации»

Отчёт по практическому занятию №8

Студент: Шатерник Г. И.

ФИТ 2 курс 3 группа

Преподаватель: Ржеутская Н.В.

Минск 2023 г.

**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.

**Ход выполнения работы:**

1. Ознакомиться с созданием криптографического приложения
2. Ответить на теоретические вопросы
3. Ответы оформить в отчет

**Контрольные вопросы:**

1. Какие симметричные алгоритмы шифрования Вы знаете?

Простая перестановка, одиночная перестановка по ключу, двойная перестановка по ключу, магический квадрат Дюрера, шифр Цезаря, ширф Трисемуса, шифр Плейфера, шифр Виженера, двойной квадрат Уинстона.

1. Какие ассиметричные алгоритмы шифрования Вы знаете?

· **RSA** (аббревиатура от Rivest, Shamir и Adelman, фамилий создателей алгоритма) — алгоритм, в основе которого лежит вычислительная сложность факторизации (разложения на множители) больших чисел. Применяется в защищенных протоколах SSL и TLS, стандартах шифрования, например в PGP и S/MIME, и так далее. Используется и для шифрования данных, и для создания цифровых подписей.

· **DSA** (Digital Signature Algorithm, «алгоритм цифровой подписи») — алгоритм, основанный на сложности вычисления дискретных логарифмов. Используется для генерации цифровых подписей. Является частью стандарта DSS (Digital Signature Standard, «стандарт цифровой подписи»).

· **Схема Эль-Гамаля** — алгоритм, основанный на сложности вычисления дискретных логарифмов. Лежит в основе DSA и устаревшего российского стандарта ГОСТ 34.10–94. Применяется как для шифрования, так и для создания цифровых подписей.

1. Основное назначение библиотеки System.Security.Cryptography?

Классы в пространстве имён *System.Security.Cryptography* управляют множеством сведений о криптографии. Некоторые из них являются оболочками для реализации операционных систем, а другие — исключительно управляемыми реализациями. Позволяют создать открытые и закрытые ключи, электронные подписи, ассиметрично шифровать файлы и сообщения через безопасные потоки, используя последние разработки в этой сфере, а также переопределять абстрактные классы для создания собственных реализаций алгоритмов.

1. Влияет ли размер ключа на криптостойкость алгоритма?

Совершенно просто оценивается криптостойкость симметричных ключей. Если, например, длина симметричного ключа составляет 40 бит (такое шифрование называют слабым), то для его реконструкции надо перебрать 240 чисел.

Если, например, длина ключа составляет 64 бита, то необходима сеть из нескольких десятков специализированных компьютеров, и задача решается в течение нескольких недель. Это крайне дорогое мероприятие, но технически оно возможно при современном уровне развития техники.

Сильным называют шифрование с длиной симметричного ключа 128 бит. На любом современном оборудовании реконструкция такого ключа занимает времени в миллионы раз больше, чем возраст Вселенной.

Для ключей несимметричного шифрования получить столь простую формулу, как для симметричных ключей, как правило, не удается. Алгоритмы несимметричного шифрования еще не до конца изучены (в этом нет ничего удивительного, поскольку по сей день не изучены даже свойства таких “простых” математических объектов, как простые числа). Поэтому при использовании несимметричного шифрования говорят об относительной криптостойкости ключей. Понятно, что, как и для симметричных ключей, их криптостойкость зависит от длины, но выразить это соотношение простой формулой для большинства алгоритмов пока не удалось. Обычно относительную криптостойкость оценивают по эмпирическим данным, полученным опытным путем.

1. Назовите основные классы библиотеки System.Security.Cryptography?

Иерархия классов:

- класс типа алгоритма, например SymmetricAlgorithm или HashAlgorithm. Это абстрактный уровень;

- класс алгоритма, является производным от класса типа алгоритма, например RC2 или SHA1. Это абстрактный уровень;

- реализация класса алгоритма, который является производным от класса алгоритма, например RC2CryptoServiceProvider или SHA1Managed. Это уровень реализации алгоритма.

Основные классы:

* CryptoStream – производный от класса Stream поток, который связывает потоки данных с криптографическими преобразованиями;
* CSPParameters – параметры, передаваемые поставщику служб шифрования (CSP), который выполняет криптографические вычисления;
* RSACryptoServiceProvder – шифрование и дешифрование с помощью алгоритма RSA;
* RSAParameters - стандартные параметры для алгоритма RSA (значения d, e, n, p, q и т.д.).
* Пространство имен Cryptography содержит базовый класс HashAlgorithm и производные классы, поддерживающие алгоритмы MD5, SHA1, SHA256, SHA384 и SHA512.

4) Выполнить шифрование, дешифрование и хеширование своей фамилии по указанным алгоритмам. Используя функции работы с файлами сохранить ключи шифрования, результаты шифрования и хеширования.

|  |  |  |
| --- | --- | --- |
| 21 | AES (256bit) | MD5 |

Код

using System.Security.Cryptography;

using System.Text;

class Program

{

static void Main()

{

BinaryWriter binaryWriter = new BinaryWriter(File.Open("E:\\Key.bin", FileMode.Create));

BinaryWriter binaryWriter2 = new BinaryWriter(File.Open("E:\\encr.bin", FileMode.Create));

BinaryWriter binaryWrite3 = new BinaryWriter(File.Open("E:\\hash.bin", FileMode.Create));

string message = "Шатерник Глеб Игоревич";

string secretKey = "test";

// Генерация ключа для AES

byte[] aesKey;

byte[] aesIV;

using (Aes aes = Aes.Create())

{

aesKey = aes.Key;

aesIV = aes.IV;

}

// Шифрование сообщения AES

byte[] encryptedMessage = EncryptStringAES(message, aesKey, aesIV);

binaryWriter2.Write(encryptedMessage);

binaryWriter.Write(aesKey);

// Хеширование ключа AES и секретного ключа

string aesKeyHash = GetMD5Hash(Encoding.UTF8.GetString(aesKey));

string secretKeyHash = GetMD5Hash(secretKey);

binaryWrite3.Write(aesKeyHash);

// ЭЦП = хешированный ключ AES + хешированный ключ секретный

string digitalSignature = $"{aesKeyHash}:{secretKeyHash}";

binaryWriter2.Write(aesKeyHash);

binaryWriter2.Write(secretKeyHash);

Console.WriteLine($"Сообщение: {message}");

Console.WriteLine($"ЭЦП: {digitalSignature}");

Console.WriteLine("Проверка ЭЦП:");

// Расшифровка сообщения AES

byte[] decryptedMessage = DecryptStringAES(encryptedMessage, aesKey, aesIV);

// Сравнение хешированных ключей

if (GetMD5Hash(secretKey) == secretKeyHash)

{

Console.WriteLine("ЭЦП действительна.");

}

else

{

Console.WriteLine("ЭЦП недействительна.");

}

binaryWriter.Close();

binaryWriter2.Close();

binaryWrite3.Close();

}

static byte[] EncryptStringAES(string plainText, byte[] Key, byte[] IV)

{

byte[] encrypted;

using (Aes aesAlg = Aes.Create())

{

aesAlg.Key = Key;

aesAlg.IV = IV;

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

using (MemoryStream msEncrypt = new MemoryStream())

{

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

{

using (StreamWriter swEncrypt = new StreamWriter(csEncrypt))

{

swEncrypt.Write(plainText);

}

encrypted = msEncrypt.ToArray();

}

}

}

return encrypted;

}

static byte[] DecryptStringAES(byte[] cipherText, byte[] Key, byte[] IV)

{

byte[] decrypted;

using (Aes aesAlg = Aes.Create())

{

aesAlg.Key = Key;

aesAlg.IV = IV;

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))

{

decrypted = Encoding.UTF8.GetBytes(srDecrypt.ReadToEnd());

}

}

}

}

return decrypted;

}

static string GetMD5Hash(string input)

{

using (MD5 md5Hash = MD5.Create())

{

byte[] data = md5Hash.ComputeHash(Encoding.UTF8.GetBytes(input));

StringBuilder builder = new StringBuilder();

for (int i = 0; i < data.Length; i++)

{

builder.Append(data[i].ToString("x2"));

}

return builder.ToString();

}

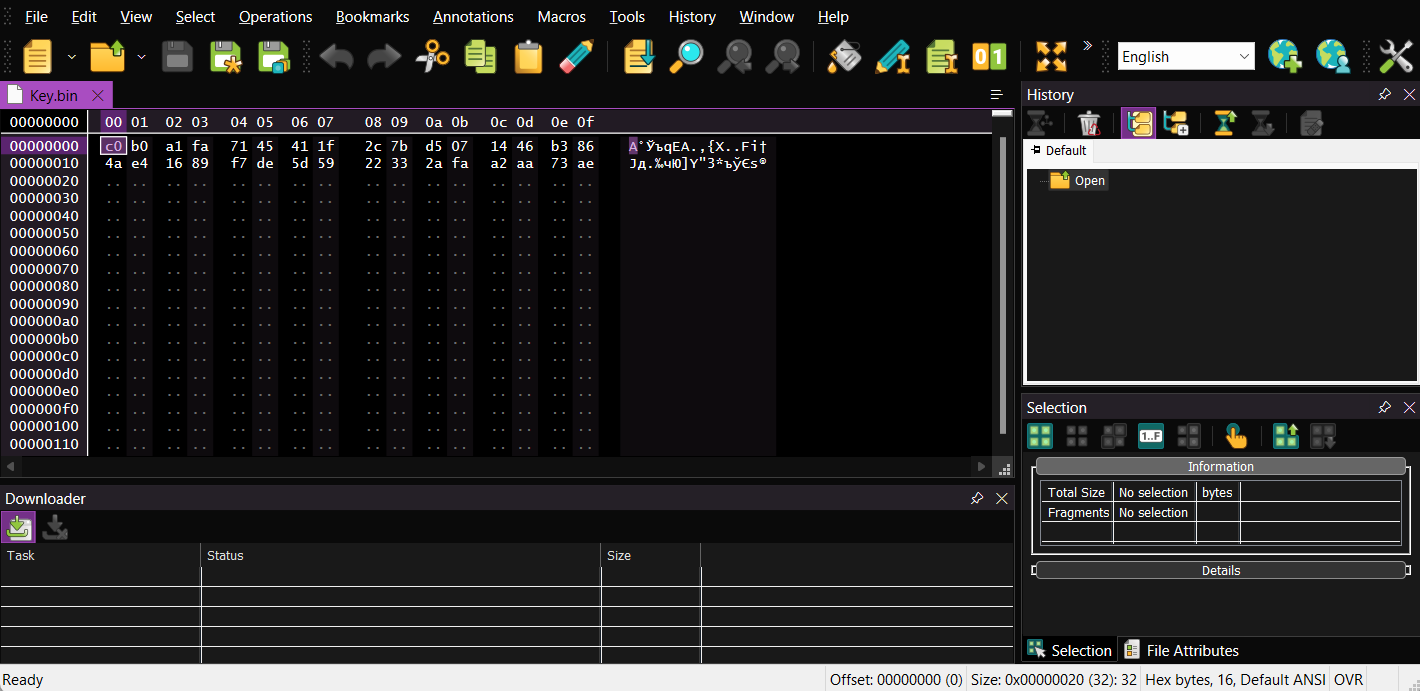
}

}

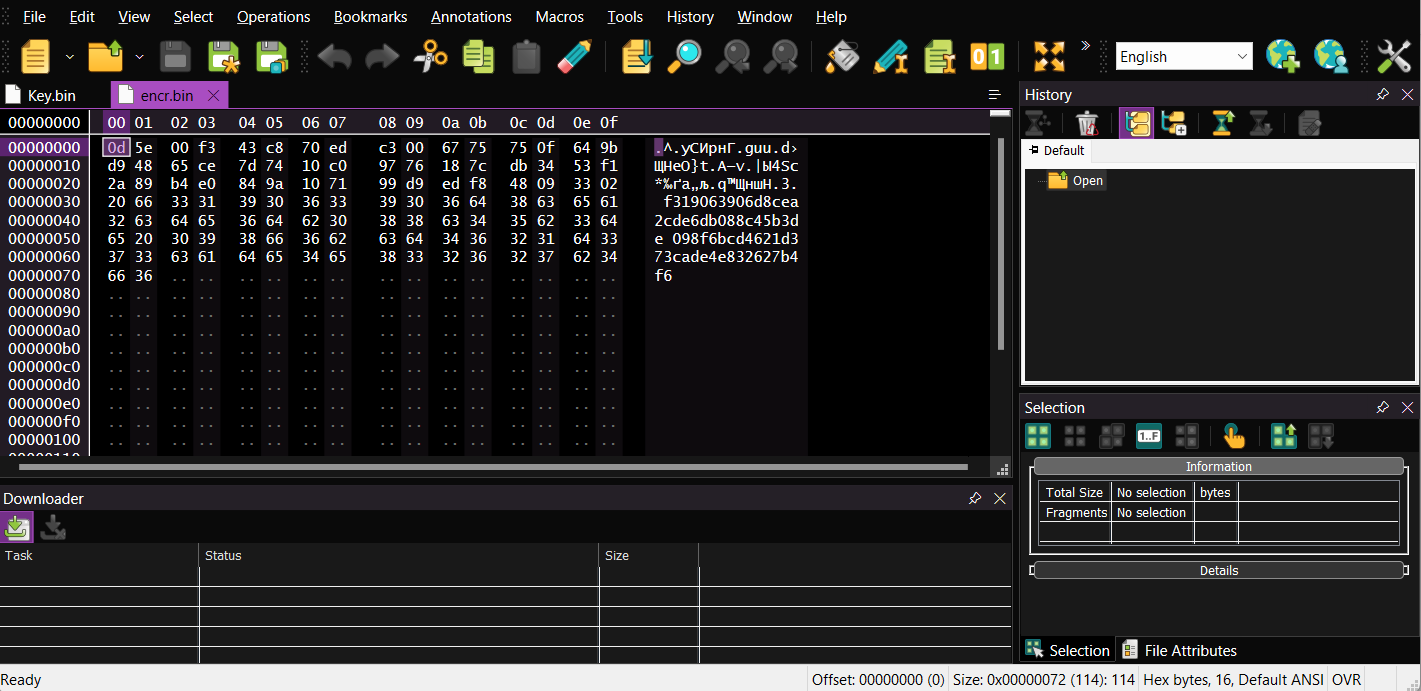
Результат



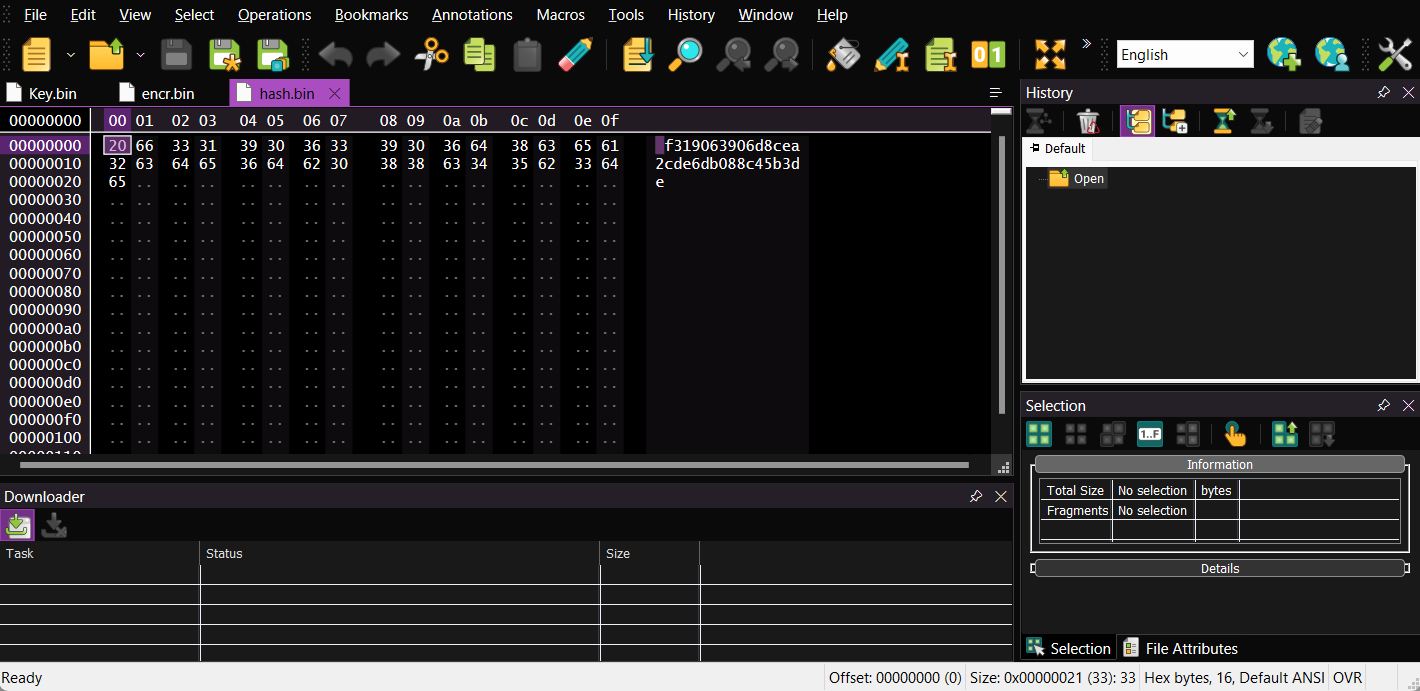
5). Для выше указанных алгоритмов используя Hex-редактор продемонстрировать ключи шифрования, зашифрованные и захешированные данные.

****

**Ключи**

****

**Сообщение**

****

**Хеш**

**Вывод:** в ходе работы были изучены модели криптографии .NET Framework, основные классы и структуры данных.