Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Дисциплина «Основы защиты информации»

Отчёт по практическому занятию №8

Студент: Турчинович Н. А.

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.

**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.

**Ответить на следующие вопросы:**

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.

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.

**2. 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.

Он поддерживает работу с асимметричными и симметричными ключами, то есть позволяет шифровать и расшифровывать данные, а также работать с электронными сертификатами1. Набор поддерживаемых криптографических алгоритмов зависит от конкретного криптопровайдера.

Криптопровайдер (Cryptography Service Provider, CSP) — это независимый модуль, позволяющий осуществлять криптографические операции в операционных системах Microsoft, управление которым происходит с помощью функций CryptoAPI.

**3. 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. 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:**

|  |  |  |
| --- | --- | --- |
| 29 | AES (192bit) | SHA384 |

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.

Код:

using System.Net.Security;

using System.Security.Cryptography;

using System.Text;

namespace lab08\_1 {

internal class Program {

static void Main(string[] args) {

string filePath = @"D:\Study\2c1s\ОИБ\lab08\lab08\lab08.1\Files";

string surname = "Турчинович";

Console.WriteLine($"[Инициализация] Исходный текст: {surname}");

Console.WriteLine();

// ШИФРОВАНИЕ

// Создание AES провайдера

using (var aes192 = Aes.Create()) {

// Установка параметров

aes192.KeySize = 192;

aes192.GenerateKey();

aes192.GenerateIV();

// Получение ключей

var aes192Key = Convert.ToBase64String(aes192.Key);

var aes192IV = Convert.ToBase64String(aes192.IV);

// Шифрование

var bytesToEncrypt = Encoding.UTF8.GetBytes(surname);

byte[] encryptedBytes;

using (var aes192Encryptor = aes192.CreateEncryptor(aes192.Key, aes192.IV)) {

encryptedBytes = aes192Encryptor.TransformFinalBlock(bytesToEncrypt, 0, bytesToEncrypt.Length);

}

var encryptedData = Convert.ToBase64String(encryptedBytes);

// Запись данных в файл

File.WriteAllText(Path.Combine(filePath, "aes192Key.txt"), aes192Key);

File.WriteAllText(Path.Combine(filePath, "aes192IV.txt"), aes192IV);

File.WriteAllText(Path.Combine(filePath, "encryptedData.txt"), encryptedData);

// Вывод данных в консоль

Console.WriteLine($"[Шифрование] Ключ: {aes192Key}");

Console.WriteLine($"[Шифрование] Вектор: {aes192IV}");

Console.WriteLine($"[Шифрование] Зашифрованный текст: {encryptedData}");

Console.WriteLine();

}

// ДЕШИФРОВАНИЕ

// Создание AES провайдера

using (var aes192 = Aes.Create()) {

// Чтение данных

var aes192Key = File.ReadAllText(Path.Combine(filePath, "aes192Key.txt"));

var aes192IV = File.ReadAllText(Path.Combine(filePath, "aes192IV.txt"));

var encryptedData = File.ReadAllText(Path.Combine(filePath, "encryptedData.txt"));

// Установка параметров

aes192.KeySize = 192;

aes192.Key = Convert.FromBase64String(aes192Key);

aes192.IV = Convert.FromBase64String(aes192IV);

// Дешифрование

var encryptedBytes = Convert.FromBase64String(encryptedData);

byte[] decryptedBytes;

using (var aes192Decryptor = aes192.CreateDecryptor(aes192.Key, aes192.IV)) {

decryptedBytes = aes192Decryptor.TransformFinalBlock(encryptedBytes, 0, encryptedBytes.Length);

}

var decryptedData = Encoding.UTF8.GetString(decryptedBytes);

// Вывод данных в консоль

Console.WriteLine($"[Дешифрование] Ключ: {aes192Key}");

Console.WriteLine($"[Дешифрование] Вектор: {aes192IV}");

Console.WriteLine($"[Дешифрование] Зашифрованный текст: {encryptedData}");

Console.WriteLine($"[Дешифрование] Расшифрованный текст: {decryptedData}");

Console.WriteLine();

}

// ХЕШИРОВАНИЕ

// Создание SHA провайдера

using (var sha384 = SHA384.Create()) {

// Хеширование

var bytes = Encoding.UTF8.GetBytes(surname);

var hashBytes = sha384.ComputeHash(bytes);

var hash = Convert.ToBase64String(hashBytes);

// Запись данных в файл

File.WriteAllText(Path.Combine(filePath, "hash.txt"), hash);

// Вывод в консоль

Console.WriteLine($"[Хеширование] Хеш: {hash}");

Console.WriteLine();

}

using (var aes192 = Aes.Create()) {

var aes192Key = File.ReadAllText(Path.Combine(filePath, "aes192Key.txt"));

var aes192IV = File.ReadAllText(Path.Combine(filePath, "aes192IV.txt"));

var encryptedData = File.ReadAllText(Path.Combine(filePath, "encryptedData.txt"));

// Установка параметров

aes192.KeySize = 192;

aes192.Key = Convert.FromBase64String(aes192Key);

aes192.IV = Convert.FromBase64String(aes192IV);

}

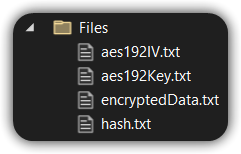
}

}

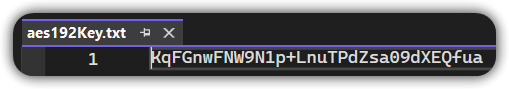
}

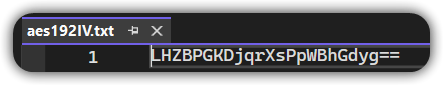
Результат:

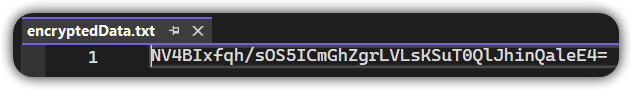
Запись в файлы



Содержимое:

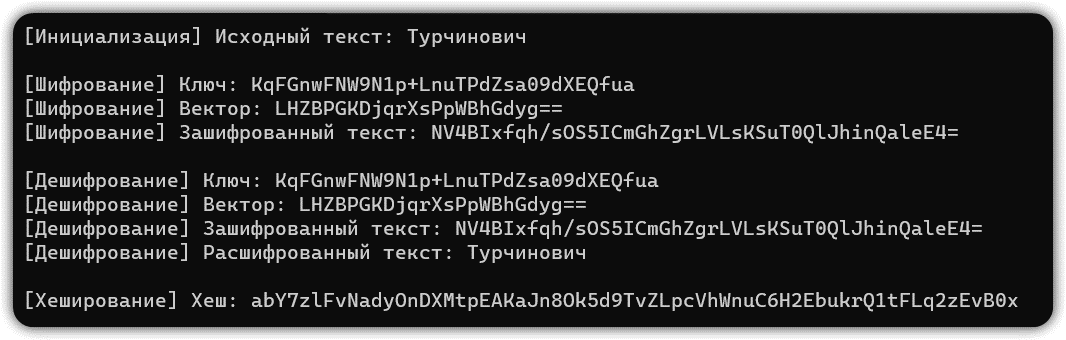






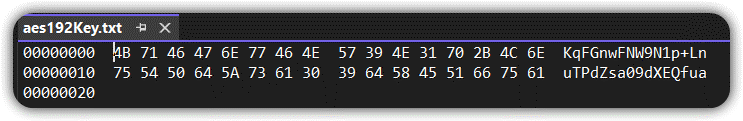


Консоль:

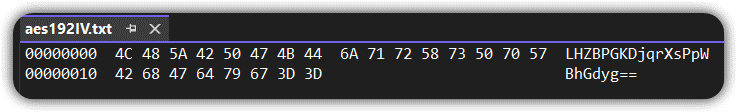


1. Для выше указанных алгоритмов используя Hex-редактор продемонстрировать ключи шифрования, зашифрованные и захешированные данные.

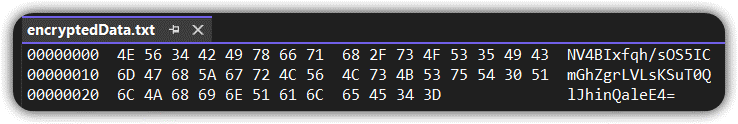
Ключ шифрования:



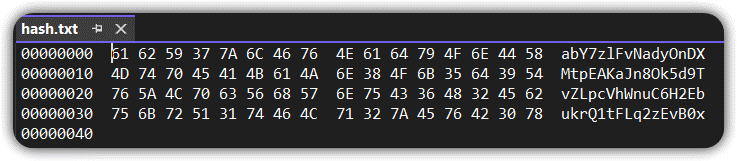
Вектор:



Зашифрованные данные:



Захешированные данные:



\*Реализовать проверку сообщения (фамилии) и хеша по примеру ЭЦП. Также продемонстрировать что будет если будет изменен хеш или сообщение.

Этот код выполняет шифрование и дешифрование фамилии с использованием RSA, создает хеш SHA384 фамилии, создает цифровую подпись на основе хеша и проверяет ее. Он также демонстрирует, что произойдет, если хеш или подпись будут изменены.

using System;

using System.IO;

using System.Security.Cryptography;

using System.Text;

using System.Linq;

using System.Security.Cryptography.X509Certificates;

public class Program {

public static void Main() {

string labPath = @"D:\Study\2c1s\ОИБ\lab08\lab08\lab08.2\Files";

// Ваша фамилия

string surname = "Турчинович";

// Создание RSA с 2048 битами

using (var rsa = new RSACryptoServiceProvider(2048)) {

// Получение ключей

var publicKey = rsa.ExportParameters(false);

var privateKey = rsa.ExportParameters(true);

// Сохранение ключей в файлы

File.WriteAllText(Path.Combine(labPath, "publicKey.txt"), Convert.ToBase64String(publicKey.Modulus));

File.WriteAllText(Path.Combine(labPath, "privateKey.txt"), Convert.ToBase64String(privateKey.Modulus));

// Шифрование

var bytesToEncrypt = Encoding.UTF8.GetBytes(surname);

var encryptedBytes = rsa.Encrypt(bytesToEncrypt, false);

var encryptedData = Convert.ToBase64String(encryptedBytes);

// Сохранение зашифрованных данных в файл

File.WriteAllText(Path.Combine(labPath, "encryptedData.txt"), encryptedData);

// Дешифрование

var bytesToDecrypt = Convert.FromBase64String(encryptedData);

var decryptedBytes = rsa.Decrypt(bytesToDecrypt, false);

var decryptedData = Encoding.UTF8.GetString(decryptedBytes);

// Проверка, что дешифрованные данные совпадают с исходными

Console.WriteLine("Decryption successful: " + (surname == decryptedData));

Console.WriteLine("Encrypted Data: " + encryptedData);

// Создание SHA провайдера

using (var sha384 = SHA384.Create()) {

var bytes = Encoding.UTF8.GetBytes(surname);

var hashedBytes = sha384.ComputeHash(bytes);

var hashStr = BitConverter.ToString(hashedBytes).Replace("-", "").ToLower();

// Сохранение хеша в файл

File.WriteAllText(Path.Combine(labPath, "hash.txt"), hashStr);

Console.WriteLine("Hash: " + hashStr);

}

// Создание цифровой подписи

// Импорт закрытого ключа

rsa.ImportParameters(privateKey);

// Чтение хеша из файла

var hash = File.ReadAllText(Path.Combine(labPath, "hash.txt"));

// Преобразование хеша в байты

var hashBytes = Enumerable.Range(0, hash.Length / 2)

.Select(x => Convert.ToByte(hash.Substring(x \* 2, 2), 16))

.ToArray();

// Создание подписи на основе хеша

var signature = rsa.SignHash(hashBytes, CryptoConfig.MapNameToOID("SHA384"));

// Сохранение подписи в файл

File.WriteAllText(Path.Combine(labPath, "signature.txt"), Convert.ToBase64String(signature));

// Вывод подписи в консоль

Console.WriteLine("Signature: " + Convert.ToBase64String(signature));

// Проверка цифровой подписи

// Импорт открытого ключа

rsa.ImportParameters(publicKey);

// Проверка подписи на основе хеша

var isValid = rsa.VerifyHash(hashBytes, CryptoConfig.MapNameToOID("SHA384"), signature);

// Вывод результата проверки в консоль

Console.WriteLine("Signature is valid: " + isValid);

// Демонстрация изменения хеша или сообщения

// Импорт открытого ключа

rsa.ImportParameters(publicKey);

// Чтение хеша из файла

// Изменение одного байта в хеше

hashBytes[0] ^= 1;

// Вывод результата проверки в консоль

Console.WriteLine("Signature is valid after hash change: " + isValid);

// Восстановление исходного хеша

hashBytes[0] ^= 1;

// Изменение одного байта в подписи

signature[0] ^= 1;

// Проверка подписи на основе измененной подписи

isValid = rsa.VerifyHash(hashBytes, CryptoConfig.MapNameToOID("SHA384"), signature);

// Вывод результата проверки в консоль

Console.WriteLine("Signature is valid after signature change: " + isValid);

}

// Хеширование SHA512

using (var sha512 = SHA512.Create()) {

var bytes = Encoding.UTF8.GetBytes(surname);

var hashBytes = sha512.ComputeHash(bytes);

var hash = BitConverter.ToString(hashBytes).Replace("-", "").ToLower();

// Сохранение хеша в файл

File.WriteAllText(Path.Combine(labPath, "hash.txt"), hash);

Console.WriteLine("Hash: " + hash);

}

}

}

Консоль:



**Вывод**

В ходе работы были изучены модели криптографии .NET Framework, основные классы и структуры данных.