**Практическая работа №8**

**Тема «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

Контрольные вопросы:

1. Основное назначение библиотеки System.Security.Cryptography?

Основное назначение библиотеки System.Security.Cryptography состоит в предоставлении программистам возможностей для работы с криптографическими операциями в рамках платформы .NET. Эта библиотека предлагает различные классы и методы для реализации алгоритмов шифрования, хэширования, генерации случайных чисел, цифровой подписи и других криптографических операций. Она обеспечивает безопасность данных, конфиденциальность и целостность информации.

1. Что из себя представляет интерфейс CryptoApi?

Интерфейс CryptoApi (Cryptographic Application Programming Interface) представляет собой набор функций и структур, предоставляемых операционной системой Windows для выполнения криптографических операций. CryptoApi обеспечивает доступ к криптографическим службам, таким как шифрование, хэширование, цифровая подпись и управление сертификатами. Он предоставляет разработчикам низкоуровневый интерфейс для работы с криптографическими алгоритмами, ключами, контейнерами и другими сущностями, связанными с безопасностью.

1. Что такое Cryptography Next Generation?

Cryptography Next Generation (CNG) - это новое поколение криптографического программного интерфейса (API) в операционной системе Windows. Он был введен в Windows Vista и предлагает расширенные возможности для выполнения криптографических операций. CNG предоставляет более гибкую архитектуру и новые алгоритмы шифрования, хэширования, генерации ключей и других криптографических функций. Он также поддерживает работу с аппаратными ускорителями для повышения производительности. CNG является более современным и рекомендуемым вариантом для разработки криптографических приложений в операционной системе Windows.

|  |
| --- |
| **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.   \*Реализовать проверку сообщения (фамилии) и хеша по примеру ЭЦП. Также продемонстрировать что будет если будет изменен хеш или сообщение. Пример того что нужно реализовать представлен ниже. |

**Варианты индивидуальных заданий**

|  |  |  |
| --- | --- | --- |
| № | Алгоритм шифрования(размер ключа) | Алгоритм хеширования |
| 6 | DES | SHA256 |

|  |
| --- |
| Результат работы: |
|  |
| Код программы: |
| using System;  using System.IO;  using System.Security.Cryptography;  using System.Text;  class Program  {  // Функция для шифрования DES  static byte[] EncryptDES(byte[] key, string plaintext)  {  using (DESCryptoServiceProvider des = new DESCryptoServiceProvider())  {  des.Key = key;  des.Mode = CipherMode.ECB;  des.Padding = PaddingMode.PKCS7;  using (ICryptoTransform encryptor = des.CreateEncryptor())  {  byte[] plainBytes = Encoding.UTF8.GetBytes(plaintext);  return encryptor.TransformFinalBlock(plainBytes, 0, plainBytes.Length);  }  }  }  // Функция для дешифрования DES  static string DecryptDES(byte[] key, byte[] ciphertext)  {  using (DESCryptoServiceProvider des = new DESCryptoServiceProvider())  {  des.Key = key;  des.Mode = CipherMode.ECB;  des.Padding = PaddingMode.PKCS7;  using (ICryptoTransform decryptor = des.CreateDecryptor())  {  byte[] decryptedBytes = decryptor.TransformFinalBlock(ciphertext, 0, ciphertext.Length);  return Encoding.UTF8.GetString(decryptedBytes);  }  }  }  // Функция для хеширования SHA256  static string HashSHA256(string data)  {  using (SHA256 sha256 = SHA256.Create())  {  byte[] hashBytes = sha256.ComputeHash(Encoding.UTF8.GetBytes(data));  StringBuilder builder = new StringBuilder();  foreach (byte b in hashBytes)  {  builder.Append(b.ToString("x2"));  }  return builder.ToString();  }  }  static void Main(string[] args)  {  // Фамилия для шифрования, дешифрования и хеширования  string surname = "Gluhova";  // Генерация ключа для DES (64 бита)  byte[] key = new byte[8];  using (RandomNumberGenerator rng = RandomNumberGenerator.Create())  {  rng.GetBytes(key);  }  // Шифрование DES  byte[] encryptedData = EncryptDES(key, surname);  Console.WriteLine("Шифрование: " + BitConverter.ToString(encryptedData).Replace("-", ""));  // Дешифрование DES  string decryptedData = DecryptDES(key, encryptedData);  Console.WriteLine("дешифрование: " + decryptedData);  // Хеширование SHA256  string hashedData = HashSHA256(surname);  // Сохранение ключа шифрования  File.WriteAllBytes("des\_key.txt", key);  byte[] bt = File.ReadAllBytes("des\_key.txt");  Console.WriteLine("Ключ шифрования: " + BitConverter.ToString(bt).Replace("-", ""));  // Сохранение зашифрованных данных  File.WriteAllBytes("encrypted\_data.txt", encryptedData);  bt = File.ReadAllBytes("encrypted\_data.txt");  Console.WriteLine("зашифрованные данные: " + BitConverter.ToString(bt).Replace("-", ""));  // Сохранение расшифрованных данных  File.WriteAllText("decrypted\_data.txt", decryptedData);  bt = File.ReadAllBytes("decrypted\_data.txt");  Console.WriteLine("расшифрованные данные: " + BitConverter.ToString(bt).Replace("-", ""));  // Сохранение хешированных данных  File.WriteAllText("hashed\_data.txt", hashedData);  bt = File.ReadAllBytes("hashed\_data.txt");  Console.WriteLine("хешированные данные: " + BitConverter.ToString(bt).Replace("-", ""));  }  } |

Работа с Hex-редактором HxD:







