Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

Практическое задание 4\_2

По дисциплине «Основы информационной безопасности»

На тему «Криптографическая защита информации»

Выполнила:

Студентка 2 курса 2 группы

Глухова Д.В.

Вариант 6

Преподаватель: ст. пр. Ржеутская Н. В.

2023, Минск

**Практическое задание №4.2**

**Тема «Криптографическая защита информации»**

Цель: Овладение основными криптографическими алгоритмами асимметричного шифрования .

**Задание №1. Рассказать процесс работы алгоритма RSA.**

**RSA (аббревиатура от фамилий Rivest, Shamir и Adleman)** — криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших целых чисел. Вот процесс работы алгоритма RSA:

1. Генерация ключей:
   * Генерируется два простых числа p и q.
   * Вычисляется их произведение n = p \* q, которое является модулем для операций шифрования и расшифрования.
   * Вычисляется значение функции Эйлера от числа n: φ(n) = (p - 1) \* (q - 1).
   * Выбирается целое число e, которое является относительно простым с φ(n) и находится в диапазоне 1 < e < φ(n).
   * Вычисляется число d, обратное к e по модулю φ(n), то есть (d \* e) mod φ(n) = 1. Число d является приватным ключом.
2. Шифрование:
   * Сообщение, которое нужно зашифровать, представляется в виде числа m, где 0 <= m < n.
   * Вычисляется зашифрованное значение c = (m^e) mod n, где e - публичный ключ.
3. Расшифрование:
   * Зашифрованное сообщение c получает получатель.
   * С помощью приватного ключа d, получатель вычисляет исходное сообщение m = (c^d) mod n.

**Задание №2. Рассказать процесс работы алгоритма Диффи-Хеллмана.**

**Алгоритм Диффи-Хеллмана** - это алгоритм для безопасного обмена ключами в открытой сети. Вот процесс работы алгоритма Диффи-Хеллмана:

1. Согласование параметров:
   * Две стороны, назовем их Alice и Bob, согласовывают общие параметры: простое число p и генератор g. Эти параметры должны быть известны обеим сторонам.
2. Генерация ключей:
   * Каждая сторона выбирает секретное число: a для Alice и b для Bob. Эти числа должны быть случайными и оставаться в секрете.
   * Alice вычисляет свой публичный ключ A = (g^a) mod p и отправляет его Bob.
   * Bob вычисляет свой публичный ключ B = (g^b) mod p и отправляет его Alice.
3. Вычисление общего секрета:
   * Alice вычисляет общий секретный ключ S = (B^a) mod p.
   * Bob вычисляет общий секретный ключ S = (A^b) mod p.
   * Значение общего секретного ключа S будет одинаковым у обеих сторон.

**Задание №3. Рассказать процесс работы алгоритма Эль-Гамаля.**

**Алгоритм Эль-Гамаля** является асимметричным алгоритмом шифрования и подписи сообщений. Вот процесс его работы:

1. Генерация ключей:
   * Генерируется большое простое число p.
   * Выбирается первообразный корень g по модулю p. Первообразный корень - это число, которое является генератором всех возможных остатков по модулю p.
   * Выбирается случайное число x, 1 <= x <= p-2, которое будет приватным ключом.
   * Вычисляется значение y = (g^x) mod p, которое будет публичным ключом.
2. Шифрование:
   * Предположим, что отправитель хочет зашифровать сообщение M для получателя.
   * Получатель выбирает случайное число k, 1 <= k <= p-2.
   * Вычисляется первая часть шифрованного текста: a = (g^k) mod p.
   * Вычисляется вторая часть шифрованного текста: b = (y^k \* M) mod p.
   * Зашифрованное сообщение состоит из пары (a, b) и отправляется получателю.
3. Расшифрование:
   * Получатель получает зашифрованное сообщение (a, b).
   * Получатель вычисляет общий секретный ключ: s = (a^x) mod p.
   * Получатель вычисляет обратное значение общего секретного ключа: s\_inv = s^(-1) mod p, где s^(-1) - обратный элемент s по модулю p.
   * Получатель вычисляет исходное сообщение: M = (b \* s\_inv) mod p.

**Задание №4\*. Используя существующие криптографические библиотеки, создать приложение и проанализировать работу вышеперечисленных алгоритмов.**

**RSA:**

|  |
| --- |
|  |
| using System;  using System.Security.Cryptography;  using System.Text;  using Org.BouncyCastle.Crypto;  using Org.BouncyCastle.Crypto.Parameters;  using Org.BouncyCastle.Security;  public class RSA  {  public static void Main()  {  // Пример RSA  Console.WriteLine("RSA Example:");  RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();  string rsaPlainText = "Hello, RSA!";  byte[] rsaPlaintextBytes = Encoding.UTF8.GetBytes(rsaPlainText);  // Шифрование  byte[] rsaCiphertextBytes = rsa.Encrypt(rsaPlaintextBytes, true);  string rsaCiphertextText = Convert.ToBase64String(rsaCiphertextBytes);  // Расшифрование  byte[] rsaDecryptedBytes = rsa.Decrypt(rsaCiphertextBytes, true);  string rsaDecryptedText = Encoding.UTF8.GetString(rsaDecryptedBytes);  Console.WriteLine("Original: " + rsaPlainText);  Console.WriteLine("Ciphertext: " + rsaCiphertextText);  Console.WriteLine("Decrypted: " + rsaDecryptedText);  }  } |

**Эль Гамаль:**

|  |
| --- |
|  |
| using System;  using System.Security.Cryptography;  using System.Text;  using Org.BouncyCastle.Crypto;  using Org.BouncyCastle.Crypto.Generators;  using Org.BouncyCastle.Crypto.Parameters;  using Org.BouncyCastle.Security;  public class ElGamal  {  public static void Main()  {  // Пример Эль-Гамаля  Console.WriteLine("ElGamal Example:");  // Генерация параметров  ElGamalParametersGenerator generator = new ElGamalParametersGenerator();  generator.Init(256, 20, new SecureRandom());  ElGamalParameters parameters = generator.GenerateParameters();  // Генерация ключей  ElGamalKeyGenerationParameters keyGenParams = new ElGamalKeyGenerationParameters(new SecureRandom(), parameters);  ElGamalKeyPairGenerator keyPairGenerator = new ElGamalKeyPairGenerator();  keyPairGenerator.Init(keyGenParams);  AsymmetricCipherKeyPair keyPair = keyPairGenerator.GenerateKeyPair();  ElGamalPublicKeyParameters publicKey = (ElGamalPublicKeyParameters)keyPair.Public;  ElGamalPrivateKeyParameters privateKey = (ElGamalPrivateKeyParameters)keyPair.Private;  // Шифрование  byte[] plaintextBytes = Encoding.UTF8.GetBytes("Hello, ElGamal!");  byte[] encryptedBytes = ElGamalEncrypt(plaintextBytes, publicKey);  // Расшифрование  byte[] decryptedBytes = ElGamalDecrypt(encryptedBytes, privateKey);  string decryptedText = Encoding.UTF8.GetString(decryptedBytes);  Console.WriteLine("Original: " + Encoding.UTF8.GetString(plaintextBytes));  Console.WriteLine("Encrypted: " + Convert.ToBase64String(encryptedBytes));  Console.WriteLine("Decrypted: " + decryptedText);  }  private static byte[] ElGamalEncrypt(byte[] plaintext, ElGamalPublicKeyParameters publicKey)  {  IBufferedCipher cipher = CipherUtilities.GetCipher("ElGamal/None/NoPadding");  cipher.Init(true, publicKey);  return cipher.DoFinal(plaintext);  }  private static byte[] ElGamalDecrypt(byte[] ciphertext, ElGamalPrivateKeyParameters privateKey)  {  IBufferedCipher cipher = CipherUtilities.GetCipher("ElGamal/None/NoPadding");  cipher.Init(false, privateKey);  return cipher.DoFinal(ciphertext);  }  } |

**Дифии-Хеллман:**

|  |
| --- |
|  |
| using System.Security.Cryptography;  class Program  {  static void Main()  {  // Создание объектов ECDiffieHellman для Алисы и Боба  using (ECDiffieHellman alice = ECDiffieHellman.Create())  using (ECDiffieHellman bob = ECDiffieHellman.Create())  {  // Генерация ключей для Алисы и Боба  byte[] alicePublicKeyBytes = alice.PublicKey.ToByteArray();  byte[] bobPublicKeyBytes = bob.PublicKey.ToByteArray();  // Обмен публичными ключами и вычисление общего секрета  using (ECDiffieHellmanPublicKey alicePublicKey = ECDiffieHellmanCngPublicKey.FromByteArray(alicePublicKeyBytes, CngKeyBlobFormat.EccPublicBlob))  using (ECDiffieHellmanPublicKey bobPublicKey = ECDiffieHellmanCngPublicKey.FromByteArray(bobPublicKeyBytes, CngKeyBlobFormat.EccPublicBlob))  {  byte[] aliceSharedSecret = alice.DeriveKeyMaterial(bobPublicKey);  byte[] bobSharedSecret = bob.DeriveKeyMaterial(alicePublicKey);  // Проверка совпадения общих секретов  bool sharedSecretsMatch = CompareByteArrays(aliceSharedSecret, bobSharedSecret);  Console.WriteLine("Alice's public key: " + Convert.ToBase64String(alicePublicKeyBytes));  Console.WriteLine("Bob's public key: " + Convert.ToBase64String(bobPublicKeyBytes));  Console.WriteLine("Shared secrets match: " + sharedSecretsMatch);  }  }  }  private static bool CompareByteArrays(byte[] array1, byte[] array2)  {  if (array1.Length != array2.Length)  {  return false;  }  for (int i = 0; i < array1.Length; i++)  {  if (array1[i] != array2[i])  {  return false;  }  }  return true;  }  } |