Лабораторная работа

Тема: Реализация алгоритма передачи информации с использованием современных асимметричных криптосистем

Теоретическая часть

Криптографическая система с открытым ключом (разновидность асимметричного шифрования, асимметричного шифра) — система шифрования и/или электронной подписи (ЭП), при которой открытый ключ передаётся по открытому (то есть незащищённому, доступному для наблюдения) каналу и используется для проверки ЭП и для шифрования сообщения. Для генерации ЭП и для расшифровки сообщения используется закрытый ключ[1]. Криптографические системы с открытым ключом в настоящее время широко применяются в различных сетевых протоколах, в частности, в протоколах TLS и его предшественнике SSL (лежащих в основе HTTPS), в SSH. Также используется в PGP, S/MIME.

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

Практическая часть

Реализация алгоритма шифрования RSA

Для использования алгоритма RSA при разработке приложений на языке программирования C# необходимо использовать пространства имен System.Security.Cryptography.

```
using System.Security.Cryptography;
```

Из данного пространства имен потребуются классы UnicodeEncoding и RSACryptoServiceProvider.

UnicodeEncoding - Представляет кодировку символов Юникода в формате UTF-16. Кодирование - это процесс преобразования набора символов Юникода в последовательность байтов. Декодирование — это процесс преобразования последовательности закодированных байтов в набор символов Юникода. Ссылка на документацию от Microsoft: https://learn.microsoft.com/ru-ru/dotnet/api/system.text.unicodeencoding?view=net-6.0

RSACryptoServiceProvider - Выполняет асимметричное шифрование и расшифровку с помощью реализации алгоритма RSA, предоставляемого поставщиком служб шифрования (CSP). Этот класс не наследуется. Поддерживает размеры ключей от 384 до 16384 бит приращения в 8 бит, если установлен расширенный поставщик шифрования Майкрософт. Ссылка

на документацию от Microsoft: https://learn.microsoft.com/ru-ru/dotnet/api/system.security.cryptography.rsacryptoserviceprovider?view=net-6.0

Для объявления экземпляров классов в классе SocketWorker можно использовать следующий код:

```
//Создаём UnicodeEncoder для перевода информации между массивом байт и строкой.
private static UnicodeEncoding ByteConverter = new UnicodeEncoding();
// Создаем объект класса RSACryptoServiceProvider для работы с библиотекой шифрования
// Инициализирует новый экземпляр класса RSACryptoServiceProvider с созданной случайным образом парой ключей указанного размера.

private static RSACryptoServiceProvider RSA = new RSACryptoServiceProvider();
```

Работа с ключами

В рамках класса RSACryptoServiceProvider существует 2 варианта выгрузки/загрузки ключей.

1. Выгрузка/загрузка в формате строки XML (тип данных string). Представляет более наглядное представление ключа. Пример ключа: "<RSAKeyValue><Modulus>mcsKi7t4sT3/B1AcfkIHdn2lJ5Ah/jitAPhBrsL1ciz2jHLYrLQwS/cWihGG5x5LvUeab4PlkM8lAnvgPLYPN2tWEwAc4EY6ZyjlFBjNAv/6Z1J3PJF3+iB1xMArbJPLlsdgXbxQQU2L7maAq6qVzr11YwWX87vGORper3YUJWE=</Modulus><Exponent>AQAB</Exponent></RSAKeyValue>"

```
// Функция ToXmlString выгружает ключ текущего объекта
// false, чтобы экспортировать информацию об открытом ключе или передать
// true для экспорта информации об открытом и закрытом ключах.
string publicKeyXml = RSA.ToXmlString(false);
string privateKeyXml = RSA.ToXmlString(true);

// Функция FromXmlString загружает ключ в текущий объект
RSA.FromXmlString(publicKeyXml);
RSA.FromXmlString(privateKeyXml);
```

2. Выгрузка/загрузка в виде массива байт. Позволяет передавать ключ по сети без дополнительных переводов в строковый вид и обратно.

```
// Выгрузка ключей
byte[] publicKeyBytes = RSA.ExportCspBlob(false);
byte[] privateKeyBytes = RSA.ExportCspBlob(true);

// Загрузка ключей
RSA.ImportCspBlob(publicKeyBytes);
RSA.ImportCspBlob(privateKeyBytes);
```

Работа алгоритма

Для шифрования сообщений используется метод Encrypt, в который необходимо передать данные в виде массива байт (byte[]) и флаг fOAEP (Boolean): true для выполнения прямой расшифровки RSA с помощью заполнения OAEP (доступно только на компьютере под управлением Windows XP или более поздних версий). В противном случае — false для использования заполнения PKCS #1 v1.5. По умолчанию используется флаг false.

```
//Шифрование
byte[] encryptedData = new byte[128];
encryptedData = RSA.Encrypt(Encoding.Unicode.GetBytes(Data), false);
```

Для расшифровки данных используется метод Decrypt, куда также передается массив байт с зашифрованным сообщением и флаг false. Расшифровка происходит на стороне получателя, а так как он является генератором ключей, то закрытый ключ уже находится в его распоряжении.

```
byte[] dataToDecrypt = new byte[128];
dataToDecrypt = Encoding.Unicode.GetBytes(GetInformation);
byte[] decryptedData = RSA.Decrypt(dataToDecrypt, false);
GetInformation = Encoding.Unicode.GetString(decryptedData);
```

Типовой порядок работы

В рамках лабораторной работы предлагается следующий порядок работы:

1. После соединения клиента и сервера посредствам сокетов (программа разработанная ранее) клиент отправляет запрос на открытый ключ сервера.
2. Сервер, получив запрос (например, сообщение "OpenKeyRequest"), выгружает открытый ключ и отправляет его клиенту.
3. Клиент, получая очередное сообщение, должен проверить является ли оно ключом. Об этом можно судить по наличию открывающего тега <RSAKeyValue>.
4. Клиент должен проверить целостность ключа. Это можно определить по закрывающему тегу </RSAKeyValue>.
5. После получения открытого ключа клиент должен его сохранить и внести в параметры крипто провайдера, например, с использованием метода RSA.FromXmlString.
6. Все дальнейшие сообщение должны шифроваться на стороне клиента и передаваться на сервер в шифрованном виде.

Схему работы разрабатываемого клиент-серверного приложения можно представить в виде рисунка 1.

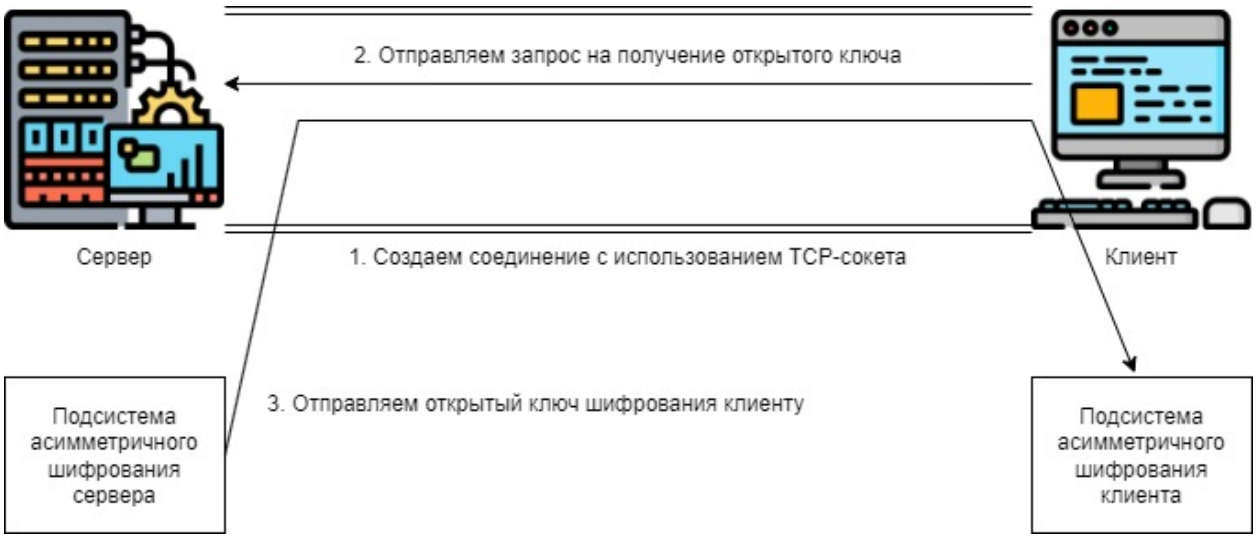

Рис. 1. Схема работы клиент-серверного приложения

Для визуализации примера работы алгоритма шифрования RSA разработаем тестовое приложение. Код программы:

```
 // Создаём экземпляр класса для работы с криптографией
RSACryptoServiceProvider RSA = new RSACryptoServiceProvider(1024);

// Выгуржаем ключи шифрования
string publickey = RSA.ToXmlString(false); //получим открытый ключ
string privatekey = RSA.ToXmlString(true); //получим закрытый ключ

// выводим ключи на форму
richTextBox1.Text = publickey;
richTextBox2.Text = privatekey;

RSACryptoServiceProvider Rsa = new RSACryptoServiceProvider(1024);
Rsa.FromXmlString(publickey);
byte[] EncryptedData;
byte[] data = new byte[1024];
data = Encoding.Unicode.GetBytes(textBox1.Text);
EncryptedData = Rsa.Encrypt(data, false);

textBox2.Text = Encoding.Unicode.GetString(EncryptedData);

byte[] DecryptedData;


DecryptedData = RSA.Decrypt(EncryptedData, false);

textBox3.Text = Encoding.Unicode.GetString(DecryptedData);
```

Внешний вид тестового приложения приведен на рисунке 2.

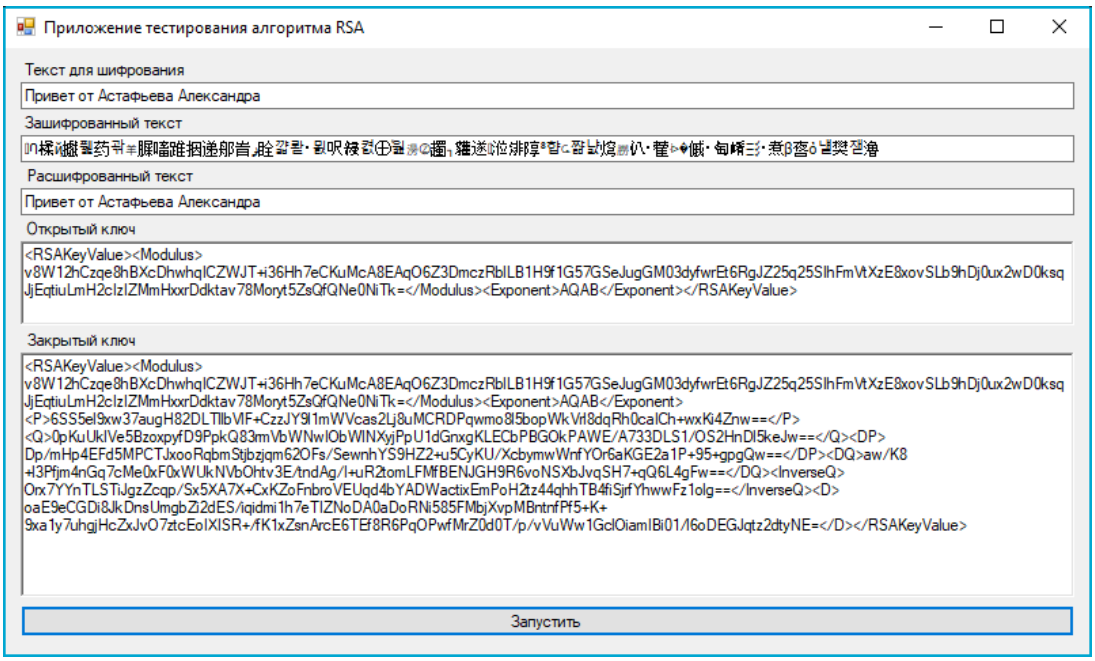

Рис. 2. Внешний вид тестового приложения

Пример работы клиент-серверного приложения можно представить в виде рисунка 3.

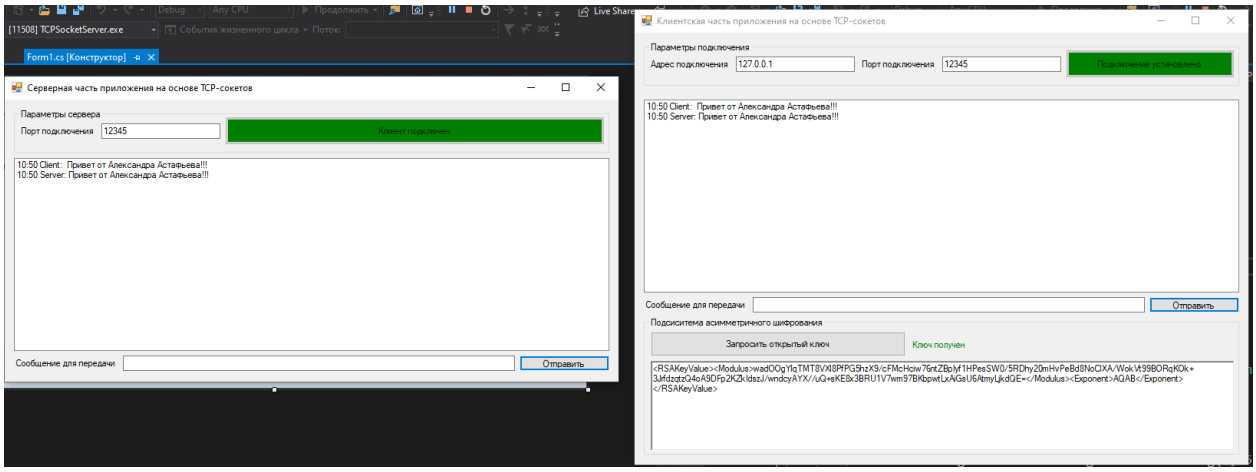

Рис.3. Пример клиент-серверного приложения с использованием асимметричного шифрования

Задание на лабораторную работу

Используя в качестве механизма сетевого взаимодействия приложение, реализованное в прошлой работе, произвести реализацию возможности передачи информации по открытому каналу связи с использованием криптографического алгоритма RSA:

1. Организовать генерацию открытого и закрытого ключей на стороне сервера.
2. Реализовать механизм запроса открытого ключа со стороны клиента.
3. Реализовать механизм получения открытого ключа клиентом, его валидацию (при возможности) и инициализацию криптопровайдера.

4. Реализовать передачу информации от клиента к серверу в зашифрованном виде.
5. Реализовать механизм расшифровки полученных сообщений на стороне сервера.
6. Произвести захват сетевого трафика с использованием программы WireShark по локальному интерфейсу «Adapter for loopback traffic capture».
7. Создать дисплейный фильтр для фильтрации сообщений, передаваемых приложениями на базе сокетов.
8. Визуализировать содержание пакетов с открытым ключом и зашифрованной информацией.
9. Оформить отчёт по проделанной работе с фиксацией основных моментов работы.