Лабораторная работа №3. Протоколы распределения ключей

Протокол распределения ключей – криптографический протокол, целью которого является создание защищенного канала связи при помощи генерации и обмена сеансовыми ключами.

1. Протоколы распределения ключей

1.1. Протокол Нидхема-Шрёдера (Needham-Schroeder protocol) на симметричных ключах

Данный протокол лежит в основе большого количества протоколов распространения ключей, использующих доверенные центры. Существует два вида данного протокола:

- Протокол Нидхема-Шрёдера на симметричных ключах;
- Протокол Нидхема-Шрёдера на ассиметричных ключах.

Протокол на симметричных ключах работает следующим образом:

Предварительный этап:

1. Необходимо наличие долговременных ключей E_A и E_B для общения с доверенного центра T с A и B соответственно;
2. A и B выбирают случайные одноразовые числа N_A и N_B соответственно.

Рабочий этап:

1. Алиса отправляет Тренту сообщение, состоящее из идентификаторов Алисы, Боба и случайного числа N_A: $M_0 = A, B, N_A$;
2. На основании полученных данных Трент формирует сообщение, имеющее следующий вид: $M_1 = E_A\big(N_A, B, K, E_B(K, A)\big)$, где K – новый сеансовый ключ;
3. Алиса расшифровывает сообщение и проверяет его подлинность, осуществляя поиск числа N_A. После этого она отправляет часть сообщения, зашифрованную на ключе E_B Бобу: $M_2 = E_B(K, A)$;
4. Боб получает и расшифровывает сообщение, достает из него ключ K и формирует сообщение для Алисы, состоящее из его случайного числа N_B, зашифрованного на ключе K: $M_3 = E_K(N_B)$;
5. Алиса получает сообщение, получает из него N_B, меняет его и отправляет его обратно Бобу: $M_4 = E_K(N_B - 1)$;
6. Алиса и Боб владеют общим секретным ключом K.

1.2. Протокол Нидхема-Шрёдера (Needham-Schroeder protocol) на асимметричных ключах

Протокол с применением ассиметричных ключей работает так:

Рабочий этап:

1. Алиса выбирает свою часть ключа, k_A, и формирует сообщение, в которое кладет свой идентификатор и k_A. Сообщение шифруется с помощью открытого ключа Боба: $M_0 = P_B(A, k_A)$;
2. Боб расшифровывает сообщение, выбирает свою часть ключа k_B и отправляет сообщение Алисе, состоящее из его части ключа и части ключа Алисы, полученной из

расшифрованного сообщение. Сообщение Боба зашифровано открытым ключом Алисы: $M_1 = P_A(k_A, k_B)$;

3. Алиса расшифровывает сообщение и забирает оттуда k_A, убедившись в том, что она осуществляла связь именно с Бобом. После этого она отправляет Бобу его часть ключа, зашифрованную на открытом ключе Боба: $M_2 = P_B(k_B)$;

4. Боб, получив сообщение и расшифровав его, убеждается в том, что он осуществлял связь именно с Алисой. При этом Алиса и Боб осуществили не только получение общего ключа, состоящего из k_A, k_B, но и взаимную аутентификацию.

1.3. Протокол Kerberos

Протокол Kerberos является одной из реализаций протокола аутентификации Нидама-Шрёдера с использованием третьей стороны, призванной уменьшить количество сообщений, которыми обмениваются стороны.

Рабочий этап:

1. Алиса отправляет Тренту сообщение, состоящее из идентификаторов Алисы и Боба: $M_0 = A, B$;

2. На основании полученных данных Трент формирует сообщение, состоящее из двух частей и отправляет его Алисе: $M_1 = \{E_A(T_T, L, B, K), E_B(T_T, L, A, K)\}$, где K – новый сеансовый ключ;

3. Алиса создает сообщение из собственного идентификатора и метки времени T_T, шифрует его на сеансовом ключе и посылает Бобу вместе со второй половиной сообщения, полученного от Трента: $M_2 = \{E_K(A, T_T), E_B(T_T, L, A, K)\}$;

4. Боб получает и расшифровывает сообщение и проверяет метку времени. Затем, в целях собственной аутентификации, Боб шифрует модифицированную метку времени $T_T + 1$ на общем сеансовом ключе: $M_3 = E_K(T_T + 1)$;

1.4. Протокол Отвея-Рииса (Otway-Rees Protocol)

Перед началом протокола имеется:

- Доверенный центр Трент;
- Пользователи Алиса и Боб, ключи для общения с Трентом E_A и E_B соответственно;
- Алиса выбирает числа N и N_A, Боб выбирает N_B;

Рабочий этап:

1. Алиса формирует сообщения для Боба, в котором передаются незашифрованные N, A, B, а также N, A, B и N_A, зашифрованные на облщем ключе Алисы и Трента: $M_0 = N, A, B, E_A(N_A, N, A, B)$;

2. Боб получает сообщение и добавляет к нему еще одну строку, которую он шифрует на общем ключе Трента и Боба: $M_1 = N, A, B, E_A(N_A, N, A, B), E_B(N_B, N, A, B)$;

3. Трент, зная оба ключа, расшифровывает сообщения Алисы и Боба. Далее Трент подтверждает тот факт, что Трент – это он и формирует общий ключ K: $M_2 = E_A(N_A, K), E_B(N_B, K)$.

4. Боб получает и расшифровывает сообщени, затем убеждается в том, что сообщение пришло от Трента. Боб принимает сгенерированный ключ K и отправляет Алисе первую часть сообщения Трента: $M_4 = E_A(N_A, K)$;

5. Алиса принимает сообщение, удостоверяется в том, что оно от Трента и принимает сгенерированный ключ K.

2. Задания

2.1. Реализация симметричного протокола Нидхема-Шрёдера

Целью данного задания является реализация протокола распределения ключей Нидхема-Шрёдера, основанного на симметричных ключах. В интерфейсе приложения должны быть наглядно представлены:

- Исходные данные протокола (модули, ключи, секретные данные и т.п.);
- Данные, передаваемые по сети каждой из сторон;
- Проверки, выполняемые каждым из участников.

Процесс взаимодействия между сторонами протокола может быть реализован как с применением сетевых технологий, так и при помощи буферных переменных. Также необходимо выделить каждый из этапов протоколов для того, чтобы его можно было отделить от остальных.

2.2. Реализация протокола Kerberos

Целью данного задания является реализация протокола распределения ключей Kerberos, являющегося модификацией симметричного протокола Нидхема-Шрёдера. В интерфейсе приложения должны быть наглядно представлены:

- Исходные данные протокола (модули, ключи, секретные данные и т.п.);
- Данные, передаваемые по сети каждой из сторон;
- Проверки, выполняемые каждым из участников.

Процесс взаимодействия между сторонами протокола может быть реализован как с применением сетевых технологий, так и при помощи буферных переменных. Также необходимо выделить каждый из этапов протоколов для того, чтобы его можно было отделить от остальных.

2.3. Реализация протокола Отвея-Рииса

Целью данного задания является реализация протокола распределения ключей Нидхема-Шрёдера, основанного на асимметричных ключах. В интерфейсе приложения должны быть наглядно представлены:

- Исходные данные протокола (модули, ключи, секретные данные и т.п.);
- Данные, передаваемые по сети каждой из сторон;
- Проверки, выполняемые каждым из участников.

Процесс взаимодействия между сторонами протокола может быть реализован как с применением сетевых технологий, так и при помощи буферных переменных. Также необходимо выделить каждый из этапов протоколов для того, чтобы его можно было отделить от остальных.

2.4. Реализация протокола Нидхема-Шрёдера на ассиметричных ключах

Целью данного задания является реализация протокола распределения ключей Отвея-Рииса. В интерфейсе приложения должны быть наглядно представлены:

- Исходные данные протокола (модули, ключи, секретные данные и т.п.);
- Данные, передаваемые по сети каждой из сторон;
- Проверки, выполняемые каждым из участников.

Процесс взаимодействия между сторонами протокола может быть реализован как с применением сетевых технологий, так и при помощи буферных переменных. Также необходимо выделить каждый из этапов протоколов для того, чтобы его можно было отделить от остальных.

3. Примечания

3.1. AesCryptoServiceProvider

Для реализации симметричных криптографических преобразований в языке C# можно использовать класс AesCryptoServiceProvider:

Шифрование:

```
byte[] Encrypt(string plainText, byte[] Key, byte[] IV){
{
using (var aesAlg = new AesCryptoServiceProvider())
{
          aesAlg.Key = Key; // задание ключа преобразования
          aesAlg.IV = IV; // задание инициализирующего вектора

          // Создание объекта-преобразования
          var encryptor = aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV);

          // Создание потока для осуществления преобразования
          using (var msEncrypt = new MemoryStream())
          {
                    // Создание криптографического потока, применяющего преобразование
                    using (var csEncrypt = new
                              CryptoStream(msEncrypt, encryptor, CryptoStreamMode.Write))
                    {
                          using (var swEncrypt = new StreamWriter(csEncrypt))
                          {

                              //Запись шифртекста в поток
                              swEncrypt.Write(plainText);
                          }
                          var encrypted = msEncrypt.ToArray();
                      }
          }
}
return encrypted;
}
```

Расшифрование:

```
string Decrypt(byte[] cipherText, byte[] Key, byte[] IV)
{
using (var aesAlg = new AesCryptoServiceProvider())
{
          aesAlg.Key = Key; // задание ключа преобразования
          aesAlg.IV = IV; // задание инициализирующего вектора

          // Создание объекта-преобразования
          var decryptor = aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV);

          // Создание потока для осуществления преобразования
          using (var msDecrypt = new MemoryStream())
          {
                    // Создание криптографического потока, применяющего преобразование
                    using (var csDecrypt = new
                              CryptoStream(msDecrypt, decryptor, CryptoStreamMode.Read))
                    {
                          using (var srDecrypt = new StreamReader(csDecrypt))
                          {

                              //Чтение байтов шифртекста и их расшифрование
                              plaintext = srDecrypt.ReadToEnd();
                          }
                    }
          }
}
return plaintext;
}
```

При создании нового объекта криптопровайдера новый ключ и инициализирующий вектор создаются автоматически. Однако, в случае расшифрования нам необходимо задать в криптопровайдере ключ и вектор, которые использовались при шифровании текста.

3.2. RSACryptoServiceProvider

Для реализации ассиметричных алгоритмов можно использовать класс RSACryptoServiceProvider:

Шифрование:

```
byte[] RSAEncrypt(byte[] DataToEncrypt, RSAParameters RSAKeyInfo, bool DoOAEPPadding)
{
          using (var RSA = new RSACryptoServiceProvider())
          {
                    byte[] encryptedData;
                    // Импорт ключей
                    RSA.ImportParameters(RSAKeyInfo)

                    // Шифрование данных
                    encryptedData = RSA.Encrypt(DataToEncrypt, DoOAEPPadding);
                    return encryptedData;
          }
}
```

Расшифрование:

```
byte[] RSADecrypt(byte[] DataToDecrypt, RSAParameters RSAKeyInfo, bool DoOAEPPadding)
{
          using (var RSA = new RSACryptoServiceProvider())
          {
                    byte[] decryptedData;
                    // Импорт ключей
                    RSA.ImportParameters(RSAKeyInfo)

                    // Шифрование данных
                    decryptedData = RSA.Decrypt(DataToEncrypt, DoOAEPPadding);
                    return decryptedData;
          }
}
```

Информация о ключах может быть передана из провайдера в провайдер при помощи структуры RSAParameters. Объект RSAParameters может быть получен из криптопровайдера при помощи функции RSACryptoServiceProvider.ExportParameters().

3.3. Преобразование строк в байты и обратно

Для того, чтобы осуществить конвертацию строк в байты и наоборот, можно воспользоваться пространством имен System.Text:

- Для перевода строки в массив байт можно воспользоваться функцией GetBytes() класса Encoding: var bytes = Encoding.GetBytes(“sampleString”);
- Для перевода строки в массив байт можно воспользоваться функцией GetString() класса Encoding: var string = Encoding.GetString(bytes);