2

Актуальность и важность проблемы обеспечения информационной безопасности обусловлена следующими факторами:

• Современные уровни и темпы развития средств информационной безопасности значительно отстают от уровней и темпов развития информационных технологий.

• Высокие темпы роста парка персональных компьютеров, применяемых в разнообразных сферах человеческой деятельности.

Разработка система защиты информации позволяет передавать конфиденциальную информацию. Однако впоследствии можно шифровать информацию с целью её хранения в ненадёжных источниках.

3

Целью курсовой работы является разработка безопасного клиент-серверного приложение для шифрованного обмена информацией внутри предприятия.

4

1. Рассмотрение принципов функционирования алгоритмов шифрования
2. Выделение особенностей написания клиент-серверных приложений на основе шифрования;
3. Изучение программных средств языка C# для шифрования и дешифрования текстовых данных по алгоритмам AES и RSA, а также отправки и получения TCP пакетов;
4. Разработка приложения для шифрованного обмена данными внутри предприятия с использованием интуитивно понятного интерфейса программа должна иметь простой интерфейс.

5

Об актуальности темы говорит тот факт, что совсем недвано самые популярные мессенджеры WhatsApp и Viber обзавелись дополнительными мерами для обеспечения безопасности данных (сквозным шифрованием), которые в теории должны предотвратить возможность доступа к передаваемой информации владельцами данных компаний. Большинство интернет-провайдеров предлагают передачу данных через свои серверы. В целом, во времена, когда частная жизнь всё больше становится виртуальной, нет ничего важнее, чем позаботится о безопасности этой жизни. Всё большую часть информации люди хранят и передают с помощью сети интернет, поэтому необходимо предоставить средства для обеспечения безопасности этих данных. В этой работе эти алгоритмы соединены вместе в гибридную схему, которая обеспечивает безопасность передаваемых данных.

Эту проблему можно решить, организовав соединение типа точка-точка (peer-to-peer) между абонентами-пользователями. Такое соединение позволяет передавать данные напрямую между абонентами, что значительно снижает вероятность доступа к этому трафику третьими лицами. Вопрос лишь в том, как обменяться данными для подключения друг к другу, ведь когда у приложения клиент-серверная архитектура, клиенты просто подключаются к серверу, который всегда расположен по одному и тому же адресу. Есть разные способы обмена контактной информацией между клиентами: можно как передать данные в реальном мире, так и отправить электронным письмом. Стоит также помнить о том, что при передаче информации, пользователь может менять своё местоположение. При этом его ноутбук будет отключаться от одного маршрутизатора и подключатся к другому, что в свою очередь повлечет за собой смену и внешнего, и внутреннего IP-адресов пользователя. Это в свою очередь повлечет за собой разрыв прямого соединения.

6.

С учетом всего этого предлагается клиент-серверная архитектура приложения. Клиент с помощью сервера контролирует свои IP-адреса, отправляет их на сервер приложения и обновляет при смене. Таким образом возможно будет решить проблему с разорванным соединением, а также будет отображаться статус собеседника (в сети, не в сети). Сервер в свою очередь хранит идентификаторы пользователей и их корректные IP-адреса. При создании соединения двух пользователей, сервер должен отправить им адреса друг друга, чтобы они смогли соединиться напрямую, а также должен обновлять адреса для поддержания соединения. Дополнительным плюсом такой архитектуры является отсутствие необходимости передавать все данные сервером. То есть, затраты на сервер будут существенно меньше, при этом приложение проще масштабировать.

Практическая часть курсовой работы представлена в виде приложение Windows Forms созданного в среде Microsoft Visual Studio Community 2019 v16.7.6 и написанного на языке C # (.NET Framework v4.7.2). Приложение создано с использованием классов TcpListener и TcpClient, CspParameters и RSACryptoServiceProvider. А также функции асинхронного чтения и записи. Приложения-клиенты реализуют выполнение следующих функций: открытие сокета, установка связи с сервером, подготовка и передача данных в соответствии с заданием. Сервер использует многопоточность для приема клиентов (т.е. сервер может принимать несколько клиентов). Это приложение обеспечивает шифрование и дешифрование пользовательских данных, а так же обмен зашифрованными письмами внутри сети предприятия на основе гибридного алгоритма.

--------

После вызова метода Start() запускается бесконечный цикл, в котором объект TcpListener будет ожидать подключения клиента и как только оно произойдёт при помощи метода AcceptTcpClient будет создан объект User, который позволяет производить обмен данными с подключившимся клиентом.

Для получения и отправки данных с помощью потоков в .NET используется класс потоков NetworkStream из пространства имен System.Net.Sockets. Он наследуется от базового класса Stream. В то же время он отличается от других классов потоков тем, что он не является буферизованным и не поддерживает перемещение в произвольную позицию с помощью метода Seek.

Поскольку клиент может послать данные гораздо большего размера, чем размер буфера потока, то для считывания всех данных, я использовал цикл do..while, который проверяем наличие данных с помощью свойства stream.DataAvailable

История переписки не сохраняется во внешней памяти клиента. Она существует только пока работает приложение. После выхода, переписка удаляется. Информация о том, кто и когда подключился и отключился и с кем вел переписку сохраняется на сервера в Log файле. Также стоит обратить внимание на то, что в случае возникновения исключения, работа сервера останавливается, чтобы в случае критической ошибки программа не «зависла».

К сожалению, в данном приложении не был реализован механизм постоянной поддержки связи за счет сервера при смене IP-адреса.

Клиентская часть использует только класс TcpClient. Создав экземпляр класса TcpClient, следующим шагом идет установка соединения с удаленным сервером. Для соединения клиента с сервером TCP используется метод Connect(), где передается объект IPEndPoint, представляющий удаленную конечную точку, с которой надо соединиться. Если соединение будет неудачным или возникнут другие проблемы, порождается исключение SocketException.

Для обработки на уровне потока, как канал между двумя соединенными приложениями, используется класс NetworkStream. Прежде чем отправлять и получать любые данные, нужно определить базовый поток. Класс TcpClient предоставляет метод GetStream() исключительно для этих целей. С помощью базового сокета он создает экземпляр класса NetworkStream и возвращает его вызывающей программе. Получив поток, используются методы Read() и Write() класса NetworkStream для чтения из приложения хоста и записи к нему. Метод Write() принимает три параметра: массив байтов, содержащий данные, которые надо отправить хосту, позицию в потоке, с которой хотим начать запись, и длину данных.

7⃣

Для простоты использования на предприятии необходимо было создать удобный интерфейс для пользователя.

Программа предназначена для приятного и полезного использования в рабочее время. Поэтому в меню использованы максимально лаконичные команды и ответы на них. Это во многом облегчает работу пользователя.

Открытый/ закрытый у всех

Сторона A использует открытый ключ Стороны B для шифрования своего собственного симметричного ключа:

После того как отправлено стороне B, сторона B использует свой собственный закрытый ключ для расшифровки симметричного ключа шифрования A:

Сами ключи при этом для алгоритма не различаются никак, нет привязки, что открытый - шифрует, закрытый - расшифровывает.  
  
Нужно понимать эту фразу так - RSA не важно, первую или вторую экспоненту вы обозвали открытым ключом. Вы всегда можете что-то зашифровать используя из пары первую экспоненту и расшифровать, используя затем вторую экспоненту из этой же пары.

Об актуальности данной темы говорит тот факт, что за время написания данной работы самые популярные мессенджеры WhatsApp и Viber обзавелись дополнительными мерами для обеспечения безопасности данных (сквозным шифрованием), которые в теории должны предотвратить возможность доступа к передаваемой информации владельцами данных компаний. В целом, во времена, когда частная жизнь всё больше становится виртуальной, нет ничего важнее, чем позаботится о безопасности этой жизни. Всё большую часть информации люди хранят и передают с помощью сети интернет, поэтому необходимо предоставить средства для обеспечения безопасности этих данных. В этой работе эти алгоритмы соединены вместе в гибридную схему, которая обеспечивает безопасность передаваемых данных.

2---