Московский физико-технический институт

**(Государственный университет)**

Факультет радиотехники и кибернетики   
Кафедра защиты информации

КУРСОВАЯ РАБОТА

**Тема**: «Программная реализация криптографических протоколов»

**Вариант 7**: «Программирование трехступенчатого протокола аутентификации с использованием алгоритмов RSA и SHA-256»

Автор работы:

Шарапов Роман Андреевич,

МФТИ(ГУ), 419 группа.

Долгопрудный 2018 год

1. Схемы и описание работы программируемых криптографических протоколов

Программируется протокол трехшаговой аутентификации. Уникальность передаваемых сообщений контро­лируется с помощью случайно сгенерированных чисел, так называемых nonce.

1. Первый шаг
   1. Клиент шлет сообщение со случайным числом RB и некоторым текстовым сообщением (опциональ­но) серверу.
2. Второй шаг
   1. Сервер шлет ответ со своим сертификатом и сообщением, которое состоит из:
      1. случайного числа сервера RА;
      2. лучайного числа клиента RВ;
      3. имени клиента;
      4. некоторого текста;
      5. всей вышеописанной информации, но подписанной (текстовая часть неподписанного сообщения отличается от текстовой части подписанного).
   2. Клиент принимает пересланное ему сообщение и проверяет:
      1. подпись;
      2. соответствует ли его случайное число RВ, отосланное ранее, числу, содержащемуся в подпи­санной части;
      3. соответствует ли его имя имени, содержащемуся в подписанной части;
   3. Если соответствует, то наступает третий шаг, если нет то соединения разрывается.
3. Третий шаг
   1. Клиент шлет ответ со своим сертификатом и сообщением, которое состоит из:
      1. случайного числа сервера RА;
      2. случайного числа клиента RВ;
      3. имени сервера;
      4. некоторого текста;
      5. всей вышеописанной информации, но подписанной (текстовая часть непописанного сообщения отличается от текстовой части подписанного).
   2. Сервер принимает пересланное ему сообщение и проверяет:
      1. подпись;
      2. соответствует ли его случайное число RA, отосланное ранее, числу, содержащемуся в подпи­санной части;
4. Используемые средства, описание интерфейсов функций

В процессе реализации протокола на языке С++ использовалась библиотека OpenSSL. Для работы с ключами использовались функции с префиксом EVP из данной библиотеки. Основными задействованными в работе функциями являются:

Для генерации ключей:

* RSA\_new() - создание нового ключевого объекта
* RSA\_generate\_key\_ex() - генерация публичного и приватного ключей

Для считывания ключей:

* BIO\_new\_mem\_buf(const void\* buf, int len) - функция инициализации памяти под хранения ключа при считывании, аналог OpenSSL для FILE\*;
* RSA \*PEM\_read\_bio\_RSAPrivateKey (BIO \*bp, RSA \*\*x, pem\_password\_cb \*cb, void \*u) - функция считывания приватного ключа из памяти BIO-потока.
* RSA \*PEM\_read\_bio\_RSAPublicKey (BIO \*bp, RSA \*\*x, pem\_password\_cb \*cb, void \*u) - функция считывания публичного ключа из памяти BIO-потока.

Для генерации электронной подписи:

* Int EVP\_DigestSignInit(EVP\_MD\_CTX \*ctx, EVP\_PKEY\_CTX \*\*pctx, const EVP\_MD \*type, ENGINE \*e, EVP\_PKEY \*pkey).- инициализация контекста электронной подписи OpenSSL ;
* Int EVP\_DigestSignUpdate(EVP\_MD\_CTX \*ctx, const void \*d, unsigned int cnt ) - добавляет указанное число байт в указанный контекст электронной подписи;
* Int EVP\_DigestSignFinal(EVP\_MD\_CTX \*ctx, unsigned char \*sig, size\_t \*siglen ) - функция, непосред­ственно подписывающая сообщение;

Для проверки электронной подписи:

* Int EVP\_DigestVerifyInit(EVP\_MD\_CTX \*ctx, EVP\_PKEY\_CTX \*\*pctx, const EVP\_MD \*type, ENGINE \*e, EVP\_PKEY \*pkey)- функция инициализации контекста проверки электронной подписи;
* Int EVP\_DigestVerifyUpdate(EVP\_MD\_CTX \*ctx, const void \*d, unsigned int cnt ); - добавляет указанное число байт в указанный контекст электронной подписи;
* Int EVP\_DigestVerifyFinal(EVP\_MD\_CTX \*ctx, unsigned char \*sig, size\_t \*siglen ) - функция, непо­средственно осуществляющая проверку электронной полписи.

Где в качестве EVP\_MD мы передаем EVP\_sha256(), собтвенно искомую хэш функцию.

1. Описание интерфейса приложения

В приложении задействовано несколько отдельных функций, со стороны сервера:

1. функция генерации ключей сервера и клиента и помещения их в соответствующие директории:

RSA\* key\_generation

1. функции считывания ключей из соответствующих директорий:

RSA\* readPrivateKey

RSA\* readPublicKey

1. функция подписи, использующая внутри себя:

bool RSASign( RSA \* rsa, const unsigned char \* Msg, size\_t MsgLen, unsigned char \*\*EncMsg, size\_t MsgLenEnc)

RSA - ключ для подписи;

Msg - сообщение для подписи;

MsgLen - длина сообщения;

EncMsg - подписанное сообщение;

MsgLenEnc - длина подписанного сообщения.

1. EVP\_DigestSignInit
2. EVP\_DigestSignUpdate
3. EVP\_DigestSignFinal

Приведенные функции описаны выше.

1. функция проверки подписи, использующая внутри себя:

bool RSAVerifySignature( RSA \* rsa, unsigned char \* MsgHash, size\_t MsgHashLen, const char \*Msg, size\_t MsgLen, bool\* Authentic)

rsa- ключ для расшифровки;

MsgHash - подписанное сообщение;

MsgHashLen- длина подписанного сообщения;

Msg- сообщение;

MsgLen- длина сообщения.

1. EVP\_DigestVerifeInit
2. EVP\_DigestVerifeUpate
3. EVP\_DigestVerifeFinal
4. функция генерации случайных чисел   
   void generateRandomNumber (std::string & random)
5. функция генерации ответного сообщения (подписанного и нет):

std::string generateFirstMessageWithoutSign(const std::string& message, const std::string& name)

std::string generateFirstMessageWithSign(const std::string& message, const std::string& name)

message - часть отправляемого сообщения;

name- имя отправителя.

1. главная функция передачи сообщений:

void process\_connection(int socket, struct sockaddr\_in& stSockAddr, RSA\* private\_key, RSA\* public\_key)

Socket, stSockAddr - переменные, относящиеся к сокету;

private\_key - приватный ключ;

public\_key - публичный ключ;

1. функция проверки ключа на валидность по формату bool isValidPublicKeyOnly (EVP\_PKEY \* pkey)

В клиенте задействованы те же самые функции за исключением генерации ключей.

В силу того, что в алгоритме не задействованы сертификаты, сервер получает имя клиента из первого сообщения.

Алгоритм работы приложения :

* Сервер ожидает входящих соединений;
* Клиент подключается к серверу;
* Клиент отправляет сообщение со случайным числом и некоторым текстом;
* Сервер принимает сообщение;
* Сервер выделяет имя клиента, получает соответствующий публичный ключ;
* Сервер выделяет из сообщения случайное число RB;
* Сервер формирует ответное сообщение:
  + В соединяет случайное число RA, случайное число клиента RB, имя клиента В, некоторый текст (text) в одну строку;

std::string result = random +b\_random + b\_name + text ;

* + Подписывает result с помощью функции RSASign
  + соединяет подписанное сообщение и неподписанное;
  + отправляет полное сообщение клиенту.
* Клиент принимает сообщение сервера:
  + проверяет публичный ключ сервера на валидность по формату:

isValidPublicKeyOnly (EVP\_PKEY \* pkey)

* + Если ключ валиден:
    - проверяет подпись с помощью функции RSAVerify;
    - проверяет на равенство RB и received\_RB - хранящееся сгенерированное число и полученное подписанное число сервера; (1)
    - проверяет на равенство name\_B и received\_name\_B - собственный идентификатор и получен­ный идентификатор; (2)
    - Если условия 1 и 2 выполнены - продолжение работы программы, иначе - выход.
  + Ключ не валиден - выход;
* Если условия выполнены - клиент формирует ответное сообщение:
  + соединяет случайное число RB, случайное число сервера RA, имя сервера, некоторый текст в одну строку;

std::string result = random +a\_random + a\_name + text ;

* + подписывает гевиЕ с помощью функции RSASign;
  + соединяет подписанное сообщение и неподписанное;
  + отправляет полное сообщение серверу.
* Сервер принимает сообщение клиента:
  + проверяет публичный ключ клиента на валидность по формату:

isValidPublicKeyOnly (EVP\_PKEY \* pkey)

* + Если ключ валиден:
    - проверяет подпись с помощью функции RSAVerify;
    - Проверяет на равенство RA и received\_RA - хранящееся сгенерированное число и полученное подписанное число сервера;
  + Если ключ не валиден - выход;
* Завершение работы алгоритма.