CS 165 Lab 7 / Project Preparation

November 19, 2012

Your class project will involve setting up a secure socket layer (SSL) connection between a client and a server, and having the client retrieve an encrypted file from the server. We will discuss the project in more detail in subsequent labs.

In Part 0 of this lab, you will create a github account. All source code for the lab and assignment must be correctly submitted using github.

In Part 1 of this lab, you'll use the command-line version of OpenSSL to create a RSA public/private key pair for the server. You will also create the file to be sent from the server to the client. You will create a hash/message digest of the file using the SHA1 hash algorithm, then sign it and verify the signature using the created RSA private/public keys, thereby ensuring that the keys work as expected.

In Part 2, you'll begin coding using the C++ OpenSSL library, performing operations similar to those done in Part 1.

Part 0

1. Create a github account by going to https://github.com/signup/free
2. Log in to github.com. In the top right corner, there is a button labeled "Create new repository." Follow the directions on this page. This repository will house your lab and your final project for the class.
3. Log in to ilearn, and submit the url of your repository for the assignment github.

Part 1

From the command line on a system with OpenSSL installed, do the following:

1. Create cleartext private and public 1024 bit RSA keys. The outputs are base 64 encoded and contain information about the encryption scheme details used, separate keys for signing, etc. The private key file contains the public key as well:

   ```
   openssl genrsa -out rsaprivatekey.pem 1024
   openssl rsa -in rsaprivatekey.pem -pubout -out rsapublickey.pem
   ```

2. Create a new plaintext file called Yourlastname.txt, and put the contents of some computer security article in it, with your name at the top. This will be the file you will send from server to client in your project.
3. Test the RSA keys by signing and verifying a hash of the file.

(a) Produce ASCII hexadecimal hash/message digest using SHA1 (an old *secure hash algorithm*):

```
openssl sha1 -out hash.txt Yourlastname.txt
```

(b) Remove the prefix text "SHA1(Yourlastname.txt)= " (including the space) from hash.txt.

(c) Convert the hash to binary before signing:

```
xxd -r -ps hash.txt > hash.bin
```

(d) Sign the hash:

```
openssl rsautl -sign -inkey rsaprivatekey.pem -in hash.bin
                                              -out hash-signature.bin
```

(e) Verify the signature:

```
openssl rsautl -verify -pubin -inkey rsapublickey.pem
                       -in hash-signature.bin -out hash-verify.bin
```

(f) Convert the signature verification output back from binary to base 64 / plain text. Verify that the original hash and the signature verification output are the same. If so, you can assume that your RSA keys are working properly.

```
xxd -ps hash-verify.bin > hash-verify.txt
diff hash.txt hash-verify.txt
```

Part 2

1. Using the OpenSSL library, write a C++ program to do the following. This program will share much functionality with the server/client in the project. Use the OpenSSL BIO tools for I/O wherever practical. See below for a sample file and documenation links. Use the following to compile in the lab:

```
g++ simple.cpp -l ssl -l crypto
```

(a) Read in the file `Yourlastname.txt`

(b) Compute the SHA1 hash

(c) Read in the RSA private key from the file `rsaprivatekey.pem`. Do *not* programatically generate a new key. Use the function:

```
PEM_read_bio_RSAPrivateKey
```

(d) Sign the hash using the RSA private key and output the signed hash to `hash-code-signature.bin`. Use the function:

```
RSA_private_encrypt(....., RSA_PKCS1_PADDING)
```

(e) Read in the RSA public key from the file `rsapublickey.pem`. Use the function:

```
PEM_read_bio_RSA_PUBKEY
```

(f) Recover the hash from the signature using the RSA public key, thus verifying the signature. Use the function:

```
RSA_public_decrypt(....., RSA_PKCS1_PADDING)
```

(g) Compare the hash code recovered from the signature with the original hash.

(h) Write a copy of the text file to `DocOut.txt`

2. Verify that the signed hash matches the signed hash output that was obtained using the command line approach above (that is, that `hash-signature.bin` and `hash-code-signature.bin` match)

3. Demo your program, and commit/push it to your git repository.

Documentation

Documentation on BIOs can be found at the following OpenSSL documentation locations, in decreasing order of personally perceived usefulness:

1. Customized instructive sample code (simple.cpp) has been placed on iLearn. Get it to run first.

2. *Network Security with OpenSSL* Online Book (Probably Limited Licenses), Ch. 4, Abstract Input/Output Section. http://proquest.safaribooksonline.com/0-596-00270-X

3. http://www.openssl.org

   http://www.openssl.org/docs/crypto/bio.html# (Specifically)

4. http://www.columbia.edu/~ariel/ssleay/

   http://www.columbia.edu/~ariel/ssleay/bio.html (Specifically)

5. Two SSL intro documents have been placed on iLearn, though they have more applicability to later stages of the project, where SSL will be used more extensively.

Error Message Printing

You'll want to add these two lines somewhere in your code (probably just after library initialization) to convert the error codes to readable strings:

```
ERR_load_crypto_strings();
SSL_load_error_strings();
```

Then use something like this to print out error messages:

```
char buf[256];
int err;
while ((err = ERR_get_error()) != 0) {
ERR_error_string_n(err, buf, sizeof(buf));
printf("*** %s\n", buf);
}
```