

CS 5480/6480: Computer Networks – Fall 2006
Programming Assignment 3
Due by 11:59:59 PM Wednesday December 6th 2006 – No Extensions

Important:

- **No cheating will be tolerated.**
- **No late submissions will be allowed.**

Total Points for this homework: 100

The goal of this programming assignment is to use **OpenSSL** and socket programs to implement a very simple secure text message transfer from end-host Alice to end-host Bob. The program starts by Alice sending a hello message to Bob asking for its public key. Bob responds with its public key that is digitally signed using a private key whose equivalent public key is known to everyone including Alice. Alice uses this well-known public key to obtain Bob's public key. Next, Alice uses its private key, a symmetric key, and SHA-1 to implement the block diagram shown in Figure 8.29 (Kurose and Ross, 3rd Edition) for securely transmitting a text message (contained in a file). Bob implements the inverse of the Figure 8.29 to obtain the text message.

You need to write two programs that will represent Alice and Bob.

| Alice | Bob |
|---|--|
| Obtaining Bob's public key. | Provide Bob's public key. |
| Use openssl commands, to generate a symmetric private key. Use openssl commands for integrity protection, encryption, and signing of the text message to implement Figure 8.29. | Wait, do nothing |
| Transfer encrypted, integrity protected, and signed message, together with the symmetric key. | Receive secure data from Alice. |
| End. | Use openssl commands to implement the inverse of Figure 8.29 to obtain Alice's message. Check for message integrity. Print the message. End. |

Before running the programs - Alice generates a public and a private key (using openssl commands) and stores them. Bob also generates a public and a private key and stores them. Generate an additional public and private key pair such that Alice knows the public key and Bob knows the private key. Bob uses this private key to sign its own public key. Bob also knows the public key of Alice.

The key challenge in doing this assignment is to understand OpenSSL commands. This should be done with the help of OpenSSL man pages (man openssl) and some examples

provided below. Additionally, the different types of message components including message digest, symmetric key, encrypted message, etc should be properly delimited and sent in a file or sent in separate files so that they could be properly parsed/separated at the other end.

Note: Instead of transmitting a symmetric key, use symmetric key encryption with the help of a password and encrypt this password using Bob's public key. Bob could decrypt the message by specifying this password in the openssl command line.

Use RSA for public key and 3DES for symmetric key cryptography respectively.

As before, submit a design document, the shell script and the socket program(s) together with sample outputs, all electronically, using *handin*. The grading policy is same as the one used for PA1.

OpenSSL Examples

(More detailed examples could be found in the book *Network Security with OpenSSL* by John Viega, Matt Messier & Pravir Chandra, O'Reilly 2002):

RSA commands:

```
openssl genrsa -out privatekey.pem 1024
```

Generates a 1024 bit RSA private key and writes it into the file `privatekey.pem`. The PEM format is widely used for storing keys, certificates etc..

```
openssl rsa -in privatekey.pem -pubout -out publickey.pem
```

Generates the corresponding RSA public key and writes it in `publickey.pem`.

```
openssl rsautl -encrypt -pubin -inkey publickey.pem -in x.txt -out y.txt
```

Contents of the file `x.txt` are encrypted and written to file `y.txt` using RSA public key from the file `publickey.pem`.

Symmetric Key Commands:

```
openssl enc -ds3 -in x.txt -out y.bin -pass pass:cs5480
```

Contents of `x.txt` are encrypted using DES3 in CBC (cipher block chain) mode and the resulting ciphertext is placed into `y.bin`. The password `cs5480` is used for generating the symmetric key. The "bin" extension indicates that the output is raw binary.

```
openssl enc -ds3 -d -in y.bin -out z.txt -pass pass:cs5480
```

Contents of y.bin are decrypted using DES3 and the resulting plaintext is placed into file z.txt. The password cs5480 is used to generate the symmetric key.

Message Digest Commands:

```
openssl dgst -sha1 -out y.txt x.txt
```

SHA1 hash function for the file named x.txt is computed and written into y.txt.

```
openssl sha1 -sign privatekey.pem -out y.bin x.txt
```

The SHA1 hash of the file named x.txt is signed using the RSA private key in the file rsaprivatekey.pem and the signature is written into the file y.bin.

```
openssl sha1 -verify publickey.pem -signature y.bin x.txt
```

The signature of the file x.txt that is contained in file y.bin is verified using SHA1 message digest and the public key in file publickey.pem.

Student's Questions and Answers from Fall 2004

Q. Is there any better logic for concatenation and deconcatenation? Book just shows them as + and -, but I am wondering how we deconcatenate more than two messages.

Can I put any separator between two messages and assume the other person knows what the separator is?.

Any idea for these concatenation and deconcatenation process?

Answer 1: I have done it as follows:

Suppose we have two files file1 and file2. I merge them in mergedfile with the following format:

```
mergedfile = <Number of bytes in file1><special character><contents of file1><contents of file2>
```

For example:

```
if
file1 = This is file1
file2 = This is file2
```

```
then mergedfile = 13kThis is file1This is file2
```

Here 13 is the number of bytes in file1, and k is the special character, which is not a number. It can be parsed easily and I have written a program to do that. I think a script can also be written do the same.

Answer 2: Here's how I did it:

First, I measure the number of bytes in the first file using the Perl 'stat' function (and save the value in a variable).

Then I concatenate the two files using the Unix 'cat' function.
cat file1 file2 > file3

Then when it comes time to break the files apart again, I use the Unix 'split' command.
split -b 25 file3 (if your first file had 25 bytes in it)

Of course there are more details than just that, but there's the basic idea.

Answer 3: Personally, I just use the "tar" to combine/uncombine the message body and signature etc together.

Answer 4: Some pieces of information are fixed length once they are encrypted.

I

used this information to pick the last 128 bytes off of an incoming stream with tail. Tar sounds like a good idea too though.

Q. Hi

Does anybody know the command to decrypt a message using public key?

If I understand the assignment correctly, Bob encrypts his public-key using the well-known-private key. Alice must therefore decrypt using the well-known-public key. However, the openssl -rsautl command wont let me decrypt using the public key (it gives an error: "A Private key is needed for this operation"). Could somebody enlighten me as to what I'm missing? Thanks in advance

Answer 1: Try digitally signing the public key, instead of encrypting it.

Q. I am having the problem that, when I transfer an integer file length and then a file using java sockets, the received file ends up with all nulls past a certain point - for example, all bytes after byte 1458 are null. Any ideas?

Thanks for any help

Answer: I'm just going to make a wild guess. It's difficult to know for sure what is happening without more details.

But a possible cause of the problem you are experiencing may be that you're not readin in the complete file. I'm assuming that you first read in an integer that tells you how many bytes are in the file then you create a byte array and try and read in the bytes. If this is the case and your using the "read" method of a InputStream you need to make sure that your checking the return value.

For example:

```
InputStream is = sock.getInputStream();
byte arr[] = new byte[2000];
```

```
is.read(arr);
```

May or may not read in 2000 bytes of data (even if 2000 bytes of data were sent over the socket). You need to do something like this:

```
InputStream is = sock.getInputStream();
byte arr[] = new byte[2000];
int read = 0, last;
```

```
while (read < 2000) {
    if ((last = is.read(arr))) {
        // handle the EOF
    } else {
        read += last;
    }
}
```

Q. When I try to sign bob's public key to send to alice I get an error saying that the data is too big.

Anyone else having this problem, and if so how did you get around it? If no one else has this

problem then does anyone know what I'm doing wrong?

I'm using the command "openssl rsautl -sign -inkey private.pem -in bobpublic.pem -out bobpublickey"

Answer 1: I got around that by decreasing the size of Bob's public key and/or increasing the size of the additional key. Just tweak the final parameter of your call to `openssl genrsa` until you find a combination that works with each other.

Answer 2: I found the wording a little confusing on the assignment, but it says that bob "digitally signs" the message. I couldn't get the -sign option to work. Well, I actually got the same error message as you. I worked around this by creating a digest of bob's public key and then encrypting that. That's what a digital signature is anyway.

Of course, that might not be an acceptable solution. This stuff is all new to me. The wording that throws me off is that Alice doesn't really need to encrypt anything to get the public key if it is only signed (she can, and does in my script, verify it is correct). Anyway, that's my two cents.

Answer 3: I was having exactly the same problem as you. It didn't work until I made the shared keys at 4x the size of Bob's keys. So, I generated Bob's public key at 512 bits and the shared key at 2048 bits.

Answer 4: If you see in the assignment sentence number 3, you are just supposed to digitally sign Bob's public key before sending to Alice. No need for encryption here.

Answer 5: The confusion is from sentence 3 of the bottom paragraph: "Bob uses this private key to encrypt its own public key."

So, from your email I will assume that we should substitute "sign" for "encrypt" in the bottom paragraph.

Answer 6: Yup! I was in the same confusion initially. But after reading the class mails again and again, I came to the conclusion that Bob's public key should only be signed. Also Signing it makes more sense.

Q. In the chart under Alice, we are told to encrypt the symmetric key with Bob's public key and transmit this encrypted symmetric key. But in the note later on, we are told to only use Bob's public key to encrypt the password used to generate the symmetric key and transmit this password. Which is it?

Answer 1: You have to encrypt password(which is used to generate the sym. key) using bob's key and transmit.

Answer 2: I believe it is:

Encrypt the message package using the symmetric key.
Then encrypt the symmetric key using bob's public key.
Package them up and ship 'em off


```
#!/bin/csh
```

```
# generate Bob's private and public keys
openssl genrsa -out bobprivatekey.pem 256
openssl rsa -in bobprivatekey.pem -pubout -out bobpublickey.pem

# sign Bob's public key using private key of well-known public key
openssl rsautl -sign -in bobpublickey.pem -inkey wkprivatekey.pem -out
sendpublickey.pem
```

I have read the man pages, but I'm not certain what I am doing wrong.

2. I am not a very experienced perl programmer, and have only used it a few times (but would like to learn more). I am unfamiliar with how to run an openssl command in a perl script. I can just use shell scripts if necessary, but would like to learn to use perl. How to I run an openssl command in a perl script? Also, how do I run a C program from a perl script?

3. Finally, I am trying to understand how once Bob has signed his public key and sent it on to Alice, how Alice will be able to obtain the actual key. The man page recommends the following for "recovering signed data", but does not give an argument for where to write the signed data (sendpublickey is the data sent from Bob and wkpublickey.pem is the well-known public key that Alice has).

```
openssl rsautl -verify -in sendpublickey.pem -inkey wkpublickey.pem
```

Answer: 1) Well, that error was a surprise to me too! I thought that you could sign messages of arbitrary lengths, but apparently you can't. What you can do instead is sign the sha1 hash of 'bobpublickey.pem' using 'wkprivatekey.pem' to create say 'signedhash.bin'

Then when Alice asks for Bob's public key, Bob will send Alice both 'bobpublickey.pem' and 'signedhash.bin'. Alice can now verify that no one has modified 'bobpublickey.pem' using the signed hash.

The sequence of commands would be like this:

Bob

```
openssl sha1 -sign wkprivatekey.pem -out signedhash.bin bobpublickey.pem
```

Bob --> Alice : bobpublickey.pem & signedhash.bin

Alice

```
openssl sha1 -verify wkpublickey.pem -signature signedhash.bin
bobpublickey.pem
```

The output of the above command should be:
Verified OK

Then, Alice can use bobpublickey.pem itself.

2) In perl, you can run system commands using either the system() function call or enclosing the command inside backticks (Ex: `ls -l`) I usually use the second method. You can get the output of the command by assigning it to a variable

Ex:

```
$filelist = `ls -l`
```

The variable \$filelist will now contain the list of files separated by newline characters.

3) I guess the answer in part 1 takes care of this, since Bob will be sending Alice 'bobprivatekey.pem' in whole.

Question: With regard to verifying Authenticity and Message Integrity:

The instructions for the assignment give the following commands as examples for the assignment:

Message Digest Commands:

```
openssl dgst -sha1 -out y.txt x.txt
```

SHA1 hash function for the file named x.txt is computed and written into y.txt.

```
openssl sha1 -sign privatekey.pem -out y.bin x.txt
```

The SHA1 hash of the file named x.txt is signed using the RSA private key in the file rsaprivatekey.pem and the signature is written into the file y.bin.

```
openssl sha1 -verify publickey.pem -signature y.bin x.txt
```

The signature of the file x.txt that is contained in file y.bin is verified using SHA1 message digest and the public key in file publickey.pem.

In my assignment, I didn't use the first command here (openssl dgst -sha1 -out y.txt x.txt), because it says this only generates a hash.

Instead, since the SECOND command says it generates the hash AND signs it, I used this.

Not only that, but from my testing, the verify command SEEMS to verify not only the authenticity but also message integrity.

Am I right about this? Or do I need to be using the first command here also? The documentation does say that it does both.

Answer: Yeah, you can use the first command. It helps verify the message integrity and authenticity.