CS 4480: Computer Networks - Spring 2013

Programming Assignment: 3
Secure Messaging

There are **two** due dates for this assignment:

PA 3 - A: Preliminary Design Document:
Due on April 14th 2014

PA 3 - Final: Complete Assignment:
Due on April 26th 2014

Assignment details

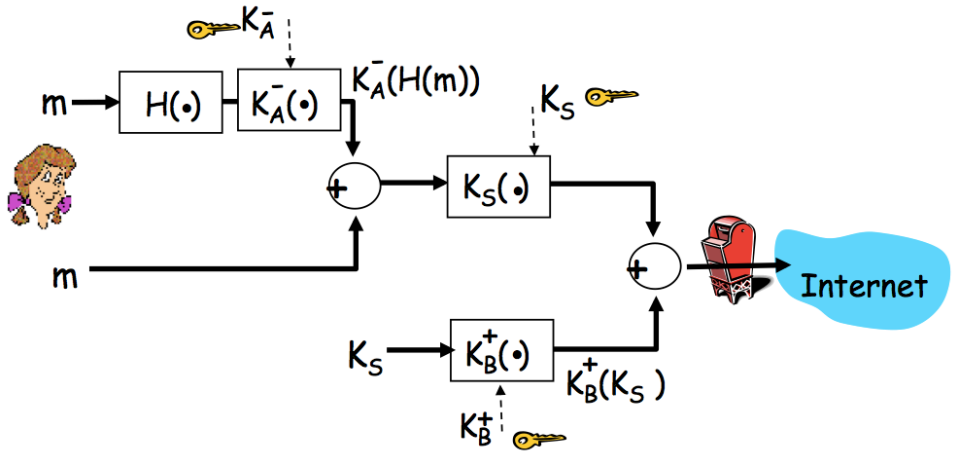

Figure 1: Sending side

In this programming assignment,[1] you will make use of socket programming and crypto libraries to create a secure messaging transfer between a program running on Alice's host and a program running on Bob's host. The program starts by Alice sending a hello message to Bob asking for his public key. Bob responds with its public key that is digitally signed using a private key whose corresponding public key is known to everyone including Alice. Alice uses this well-known public key to obtain Bob's public key. (In a real situation, Bob will send a signed certificate that will include his public key but you do not need to deal with actual certificates in this programming assignment.) Next, Alice uses her private key, a symmetric key, and SHA-1, to implement the block diagram shown in Figure 1, for securely transmitting a text message (the

[1] Credit: This programming assignment is due to a similar assignment created by Sneha Kasera for his CS5480/6480 Computer Networks course.

message might be contained in a file). Bob implements the inverse of this block diagram to obtain the text message.

Table 1 describes the functionality that your programs should implement.

Table 1: Programs representing Alice and Bob.

Alice:	**Bob:**
Obtain Bob's public key. Verify.	Provide Bob's public key together with a signed message digest of Bob's public key.
Use a crypto library for integrity protection, encryption and signing of a text message to realize Figure 1.	Wait.
Transfer encrypted , integrity protected and signed message, together with symmetric key.	Received secure message from Alice.
End.	Use a crypto library to implement the inverse of Figure 1 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 (K_A^+) and K_A^-) and stores them. Bob also generates a public and a private key (K_B^+ and K_B^-) and stores them. Generate an additional public and private key pair (K_C^+ and K_C^-) such that Alice knows the public key, K_C^+, and Bob knows the private key (K_C^-). Bob uses the private key K_C^- to digitally sign the message digest of its own public key K_B^+. (In a real situation, a certificate authority will sign the message digest of Bobs public key but for this assignment you do not need to worry about a certificate authority. Instead, Bob does the job of the certificate authority.) Bob knows the public key of Alice, K_A^+. This means that Alice does not have to send a certificate, containing her key, to Bob.

The main challenge in doing this assignment is to understand the crypto library that you choose to use. One such library is *openssl*. The benefit of *openssl* is that it can be used from the linux commandline, which is very useful for understanding the functionality and for design purposes. You should make use of the man pages for *openssl* (and its sub commands) and the examples provided below. **To get full credit you should, however, make use of a crypto library API rather than the commandline.** *openssl* has such an API, however it is notoriously difficult to use.

A viable alternative is to make use of the Java built in crypto library. Bouncy Castle [2] is another alternative for Java. For Python PyCrypto [3] can be used.

In addition to understanding and properly using a crypto library, the different types of message components including message digest, symmetric key, encrypted message, etc should be properly delimited, e.g., by sending it in a file or in separate files, so that they could be properly parsed/separated at the other end.

You should use RSA for public key encryption, 3DES for symmetric key cryptography and SHA1 for message digests.

Preparing your programs for grading You should instrument your programs to be able to report progress to "stdout". A command line option, e.g., $-v$, should be provided to trigger this reporting when your programs are run. Your reporting should clearly show the progress for each interaction between Alice and Bob. You should also print all values (in hex format) for every step in the process to allow easy visual verification between programs. (E.g., print, for both Alice's and Bob's programs, the value of the message,

[2] `http://www.bouncycastle.org/java.html`

[3] `https://www.dlitz.net/software/pycrypto/`

of the keys, of the IV, of the digest etc.) **This will be the primary means by which your programs will be evaluated.**

You should package your code into two separate tar balls, alice.tar and bob.tar, which respectively contains the code and supporting files (i.e., public/private keys) in order to run the two programs on two separate Cade Linux machines to realize the interaction.

Grading and evaluation

What to hand in

PA 3 - A: Preliminary Design Document You need to submit via Canvas a preliminary design document. In this document you should show your design and motivate your choice of crypto library. For example, an acceptable design document might show how you used the openssl commandline to "manually" design the proper interactions between Alice and Bob, and then show how those functions might be realized in using the API of the crypto library of your choice.

PA 3 - Final: Complete Assignment You submission should consist of a **single** tarball file which contains the following:

1. All the source code for your assignment. **(Packaged into two tar balls, alice.tar and bob.tar as described above.)**
2. A readme.txt file explaining how to compile and/or run your program(s).

Your submission tarball must be submitted on CADE machines using the handin command. To electronically submit files while logged in to a CADE machine, use:

`% handin cs4480 assignment_name name_of_tarball_file`

where `cs4480` is the name of the class account and `assignment_name` (pa1_a, pa1_final etc.) is the name of the appropriate subdirectory in the handin directory. Use pa3_final for this sub-assignment.

In addition, you should submit an Assignment Report as a pdf via Canvas.

This report should include the following sections:

- *Design* that describes the program design, how it works and any design tradeoffs considered and made.
- *Testing* that describes the tests you executed to convince yourself that the program works correctly. Also document any cases for which your program is known not to work correctly.
- *Output* that shows output that illustrates the correct functioning of your program.

Grading

Criteria	Points
Preliminary design document	10
Program implemented according to specification and works correctly	75
Inline documentation & exception handling	5
Assignment report	10
Total	100

Note that if you choose to implement the assignment by using the *openssl* commandline from within your program, the maximum points that you can get for the "Program implemented according to specification and works correctly" category will be 65.

Other important points

- Every programming assignment of this course must be done individually by a student. No teaming or pairing is allowed.
- Your programs will be tested on CADE Lab Linux machines. You can develop your program(s) on any OS platform or machine but it is your responsibility to ensure that it runs on CADE Lab machines. You will not get any credit if the TA is unable to run your program(s).

OpenSSL Examples

(More detailed examples could be found in the book Network Security with OpenSSL by John Viega, Matt Messier & Pravir Chandra, OReilly 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:cs4480
```

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 cs4480 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:cs4480
```

Contents of y.bin are decrypted using DES3 and the resulting plaintext is placed into file z.txt. The password cs4480 is used to generate the symmetric key.

```
openssl enc -P -des3 -pass pass:cs4480
salt=707258FB8C2212BE
key=EC2A0A6CDC4E3A152D582FDE54F8E09778384D600EE0AA07
iv =8054B20F9C0CC1C8
```

Uses the password cs4480 to generate a symmetric key, salt and IV, and print those values (without doing any encryption).

```
openssl enc -des3 -in x.txt -out y2.bin -S 707258FB8C2212BE
-K EC2A0A6CDC4E3A152D582FDE54F8E09778384D600EE0AA07 -iv 8054B20F9C0CC1C8
```

Use the previously generated key, salt and IV to encrypt x.txt and outputs the cipher text in y2.bin

```
openssl enc -des3 -d -in y2.bin -out x3.txt -S 707258FB8C2212BE
-K EC2A0A6CDC4E3A152D582FDE54F8E09778384D600EE0AA07 -iv 8054B20F9C0CC1C8
```

Use the previously generated key, salt and IV to decrypt y2.bin and outputs the plain text in x3.txt

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.