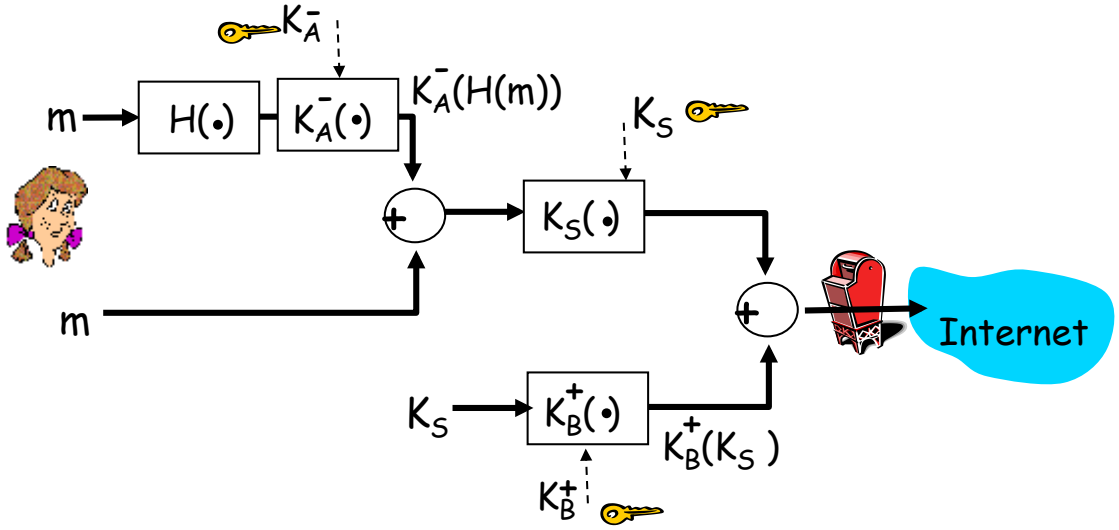
CS 5480/6480: Computer Networks – Spring 2012
Programming Assignment 3
Due by 11:59:59 PM MT April 23rd 2012

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 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 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 the figure below for securely transmitting a text message (contained in a file). Bob implements the inverse of this block diagram to obtain the text message.



You need to write two programs that will represent 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 *openssl* commands for integrity protection, encryption, and signing of the	Wait, do nothing.

text message to implement the above figure.	
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 the above figure 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 Bob's 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 *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 (look at the Q&A from the last few years to get some suggestions on this issue) so that they could be properly parsed/separated at the other end.

Note: In the figure above, a symmetric key, K_S, is generated and encrypted using Bob's public key. In this assignment, instead of generating a symmetric key and encrypting the message with it, choose a password and encrypt the message using the *openssl* command that takes a password as an input (the password is used to generate the symmetric key but you do not see that). Now, encrypt the password with Bob's public key and send it to Bob together with the encrypted message. Bob will retrieve the password, using his private key, and then use the password to decrypt the message. So, instead of using a symmetric key directly, we are using a password that Alice generates and sends to Bob.

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

As before, submit a design document, your program(s), and a document providing detailed instructions to run your program, together with sample outputs, all electronically, using *handin*. The grading policy is same as the one used for PA1 and PA2.

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.