**CS6740: Network Security** **Spring 2016**

# Project - Extra Credit

## Due Thursday May 5 2016, 10:00 pm. Use name ProjectEC when submitting.

Implementation should be conducted using the openssl (www.openssl.org).

You will implement a two party protocol where Alice sends Bob a file. Each of the tasks can be implemented separately.

- First they will establish a Diffie-Hellman secret key, where RSA keys provided in the form of certificates are used to sign and verify Diffie-Hellman messages.
- Use the key resulted from Diffie-Hellman to derive two keys one for encryption and one for integrity.
- You will have Alice send Bob the file via a secure channel. Alice will read from a file buffers of fixed sizes of 1500 bytes, encrypt each and pass it to Bob who will decipher it and then write it in a different file. Your program should take as parameters (in addition to user names, hosts, ports, etc) the name of file to be sent, and encryption/integrity method. Your program should support three methods encrypt_and_mac, mac_then_encrypt, encrypt_then_mac.

Include a README file with your submission containing anything we need to know to run/test your project.

## PART 1 - Certificate generation for RSA keys (20 points)

You need to generate certificates containing RSA keys. The format of files name should be username.priv and username.pub for private and public certificates. You can assume that both sides have the certificate of the CA present, needed to verify the certificates of Alice and Bob.

You can read more about how to generate certificates with openssl

`http://www.openssl.org/docs/manmaster/apps/openssl.html`

## PART 2 - Use Diffie-Hellman to generate encryption key and HMAC symmetric keys (20 points)

You will need to generate parameters for Diffie-Hellman, then compute a Diffie-Hellman key.

`http://www.openssl.org/docs/manmaster/crypto/dh.html`

You will implement an authenticated Diffie-Hellman, you need to sign the message with RSA keys and include that in the message, too. Once the Diffie-Hellman key was computed, encryption and mac keys are computed as follows.

To sign and verify messages see

```
http://www.openssl.org/docs/manmaster/crypto/EVP_SignInit.html

K_e = h(K_dh)

K_hmac = h(1 || K_e)
```

where K_e is key used for encryption, K_hmac is key used for hmac and h is SHA1 hash function, and K_dh is the key resulted from the Diffie-Hellman protocol.

You will need an IV for encryption.

```
IV_1 = h(100 || K_hmac)
```

You will need to change it for every encryption (i.e. every 1500 bytes), you can assume that next IV are computed by applying a hash on the previous IV and a counter you increment. For example, for the second packet will be

```
IV_2 = h(101 || IV_1)
```

Alice and Bob will need to do their own computation of IVs.

You can use SHA1 as a hash function. SHA1 outputs 160 bits, just take the leftmost bits you need, for example for AES you need a 128 bit key.

## PART 3 - Encryption, encrypt_and_mac (20 points)

Given the keys computed in Part 2, Alice will read 1500 bytes from a file then encrypt and compute hmac and send it to Bob who will decrypt and print it to a file. Alice needs to do this for the entire file, but it will be done in chuncks of 1500 bytes.

For encrypt_and_mac each read from the file of the 1500 bytes will have the corresponding encrypted version as follows:

```
C = E(K_e, M) || hmac(K_hmac, M)
```

where E is AES in CBC mode and hmac is HMAC used with SHA1.

Description of functions and examples:

```
http://www.openssl.org/docs/manmaster/crypto/EVP_EncryptInit.html

http://www.openssl.org/docs/manmaster/crypto/hmac.html
```

## PART 4 - Encryption - mac_then_encrypt (20 points)

Same as above where mac_then_encrypt each read from the file of the 1500 bytes will have the corresponding encrypted version as follows:

```
C = E( K_e, M || hmac(K_hmac, M) )
```

## PART 5 - Encryption - encrypt_then_mac (20 points)

Same as above where mac_then_encrypt each read from the file of the 1500 bytes will have the corresponding encrypted version as follows:

```
C = E(K_e, M) || hmac(K_hmac, E(K_e, M))
```

## Notes

Make sure you always check error codes. Openssl returns objects passed as pointers in the parameters list, but those pointers contain the right data only if the function succeeded which is specified in the return error code.

Read carefully what the functions expect you to do with respect to memory allocation, many will allocate the data for you but will expect you to free it. Make sure you set all your pointers on NULL so you don't free something twice.

Make sure you test your program with large files that are not text. You can start with a small text file, but in the end test it with larger binary files, pictures or executables.

You can implement Task 3, 4, 5 without computing the symmetric keys first, just use some static keys to demonstrate functionality for Task 3, 4, 5.

You can implement Task 2 without generating the certificates just by generating RSA keys - this is to demonstrate functionality of Task 2.