CS 465 Cybersecurity Principles and Practice
Programming Assignment #1

Assigned: Tuesday, February 11, 2020
Deadline for electronic submission: 12:59 pm on Monday, February 24, 2020
Deadline for hard copy: at beginning of the class on Tuesday, February 25, 2020
No assignments will be accepted after 11:59 pm on February 28, 2020

Summary

- Write a program which implements the steps of SSL/TLS using a simulated client-server connection via writing/reading to text files to complete the Handshake phase; then encrypt and decrypt the contents of a short text file that will be sent from the server to the client in the Data Transfer phase.
- The secret key will be generated using Diffie-Hellman during the Handshake phase, which will be the used by AES 128-bit to encrypt/ decrypt the content of the text file (data.txt).
- The server and the client have to run in a single main method so that one file may be called from the command line.
- Your program should log the results of each step in a text file **ssl.txt.**

Getting and verifying the Server's ID and Server's pubic key from the Certificate Authority (CA)

A. The **Client** needs to verify the **Server** information provided by the CA, which will be stored in **CA.txt** file provided for the assignment. The **CA.txt** file contains:

- CA ID, **Server** ID, **Server** public key
- hash (i.e., fingerprint) of the above information using SHA-2 256-bit, and
- is signed (i.e., encrypted using RSA with the CA's private key).

Calculations by the client:

- Check that the message is coming from the CA by using the CA's public key (given in the CA-RSA-Public.txt). This step is actually decrypting the content of the CA.txt using RSA with the CA's public key.
- Independently calculate the hash of the CA ID, **Server** ID, and **Server** public key, using SHA-2 256-bit and verify it against the one found in the CA.txt file (provided for the assignment).
- For this step, log: results of certificate calculations.
- There are two possible outcomes:
  (1) The hashes do not match because either the public key of the server or other information has been tampered with (the hash is wrong). Then,
    - The Client should not establish connection with the malicious server.
    - For this step, log: "The certificate is invalid. Connection terminated."
    - Terminate the program.

  (2) The certificate is signed and valid.
    - If the certificate is valid, the **Client** should continue with the Handshake phase as described below.
    - For this step, log: "The client has initiated the Handshake phase."

The Handshake Phase

Highlighted cipher suites/ algorithms are the ones to be used for this assignment.

Log each step.

We will simulate a network connection by reading and writing to files **ClientToServer.txt** and **ServerToClient.txt**.

B. Start a client-server connection using port 443
- For this step, log: “The connection is complete.”

C. **Client** starts with a Client Hello initial message carrying the following information. We will simulate sending info to the **Server** by writing the following to the file **ClientToServer.txt:**
- TLS version: 1.3
- List/ array of the following cipher suites supported:

  TLS_AES_256_GCM_SHA384, TLS_CHACHA20_POLY1305_SHA256, TLS_AES_128_GCM_SHA256

- Generated **Client’s** public key **x** (see the Diffie-Hellman algorithm below) calculated from client private key **a** (selected by the client) and the modulus **P** and base **G** (provided in DH.txt file).
- The modulus (**P**) is inside [ ], and the base (**G**) is inside {} in the text file.
- The client private key a, and server private key b need to be generated using a random number method. These numbers must be greater than 1 and fewer than 10 digits to facilitate calculations.
- For this step, log: “Client Hello: TLS version 1.3: TLS_AES_256_GCM_SHA384, TLS_CHACHA20_POLY1305_SHA256, TLS_AES_128_GCM_SHA256”
  Log the calculations of **x**

Client	Server
Available to both: Modulus **P**, Base **G**	Available to both: Modulus **P**, Base **G**
Select Private key: a	Select Private key: b
Public Key Generated: **x** = $G^a \ mod \ P$	Public Key Generated: **y** = $G^b \ mod \ P$
Generated public keys exchanged	
Key received = y	Key received = x
Generated Secret Key: $k_a = y^a \ mod \ P$	Generated Secret Key: $k_b = x^b \ mod \ P$
Secret keys are identical. $k_a = k_b$	

D. **Server** selects its private key **b,** reads the file **ClientToServer.txt** and generates the secret key **k_b** (for the Data Transfer phase) using the received Client’s public key **x**.

E. **Server** responds with Server Hello message consisting of the following information written in the file **ServerToClient.txt.**
- Cryptographic algorithm chosen from options provided by the client (AES 128-bit).
- Generated public key **y** (see the Diffie-Hellman algorithm above) calculated from **Server’s** private key **b** (selected in step D) and the modulus **P** and base **G** (provided in the same DH.txt file used by the Client).
- For this step, log: “Server Hello: {Server ID}:{Server Generated Key}”

F. **Client** reads the file **ServerToClient.txt** and generates the secret key $\mathbf{k_a}$ (for the Data Transfer phase). Note that $\mathbf{k_a = k_b}$ is the Secreat key to be used with AES 128-bit in the Data Transfer phase.

G. **Client** sends the Finish Message with:
- “Key-handshake complete Client \nAll future messages will be sent with the generated secret key.”
- Log: the Finish Message, above.
- For this step, also log: “Client ready to enter the Data Transfer phase.”

H. **Server** responds with the Server Finish Message:
- “Key-handshake complete Server \nAll future messages will be sent with the generated secret key.”
- For this step, log: the Finish Message, above.
- For this step, also log: “Server ready to enter Data Transfer phase.”

Data Transfer phase

I. Server
- Reads the content of the file **data.txt** from the local directory (examples provided with the assignment).
- Pre-processes the text. Assume that the plain text in **data.tx** consists of printable ASCII characters (20 – 7E hex, that is, 32 – 126 decimal). Only digits (‘0’ – ‘9’) and letters (‘A’ – ‘Z’, ‘a’ - ‘z’) will be encrypted, that is, all other characters should be dropped from the text prior to the steps that follow below. (Note that spaces must also be removed. The resulting ASCII text will consist only of digits and letters.)
- Computes the hash of the content of the data.txt file provided from local directory using SHA-2 256-bit
- Append the hash to the content of the data.txt file.
- The hash and data should be separated (delimited) by a newline (carriage return).
- Signs the ServerID (from the Handshake phase) with the Server’s private key (i.e., encrypts the ServerID using RSA with the with the Server’s private key), which is given in the file **Server-RSA-Private.txt**. Note that Server’s private key used in this step corresponds to the Servers’ public key from the **CA.txt.**
- **E**ncrypts the above (original content appended with the hash and the signed ServerID) using AES 128-bit with the secret key $\mathbf{k_b}$ generated during the Handshake phase. Each paragraph is separated by a newline.
- Log “The file has been encrypted.”
- The communication from the Server to the Client is simulated by writing the encrypted message to the file **Secret.txt.**
- Log “The file has been sent from server.”

J. **Client**

- Reads the content of the **Secret.txt** file.
- Decrypts the message using AES 128-bit with **k$_a$** generated during the Handshake phase, then verifies the Server ID (using RSA with the Servers public key from the CA.txt file), and verifies the hash using SHA-2 256-bit.
- There are two possible outcomes of the previous step:

  (1) If the hash in the file is different from the hash computed by the client independently, the file has been tampered with.
    - Since the file has been tampered with, it should be discarded.
    - Log: "The file has been tampered with."

  (2) If the two hashes are identical, the content is valid.
    - Display the contents of the file in the console and log it
    - Log "The file has been decrypted by the client."

K. "Close" all connections and files

- Log "The connection is closed."
- Terminate the program.

Acceptable Library Usage

- Standard I/O libraries.
- You may use **openssl** for the cipher suites: SHA-2 256-bit, RSA, and AES 128-bit
- Alternatively, cipher packages (e.g. `cryptography` in Python & `javax.crypto.Cipher` in Java) may be used to invoke AES 128-bit, RSA, and SHA-3 256-bit.
- PACKAGES OTHER THAN THE ONES MENTIONED ABOVE MUST NOT BE USED.
- DIFFIE-HELLMAN ALGORITHM SHOULD BE IMPLEMENTS AS DESCRIBED IN THE HANDSHAKE PHASE. No credit will be given for using Diffie-Hellman implement in openssl or any language library.

Input to your program:

All inputs to your program with be given in files (listed below), which are provided in eCampus. Save these files locally, in the same directory as your program and read the corresponding inputs.

- CA.txt
- CA's public key (for the Client to use) given in CA-RSA-Public.txt
- Server's private RSA key, given in Server-RSA-Private.txt
- **p** and **g** for Diffie-Hellman algorithm, given in the file DH.txt
- data.txt file with the content to be encrypted by the server.

Output of the program

Text file **ssl.txt** which will serve as a log file to aid in testing and debugging (and in grading). Your implementation should log the results of each step (as described in the Handshake Phase and Data Transfer phase). Include the decrypted data.txt file contents in the log as well.

The format of the log for each step is provided above. Each log entry must be on a new line.

- **ssl.txt** that contain logs for each step of the assignment.
- **ClientToServer.txt** used for simulating the communication from the Client to the Server during the Handshake phase
- **ServerToClient.txt** used for simulating the communication from the Server to the Client during the Handshake phase
- **Secret.txt** contains the content from the **data.txt** file, appended hash produced by SHA-2, appended by the ServerID signed using RSA, and then encrypted with AES by the Server.

Implementation Languages

The following compilers / interpreters available on LCSEE UNIX server shell are allowed for implementation:

- gcc (Ubuntu 5.4.0-6ubuntu1~16.04.5) 5.4.0 20160609
- Java 8 (openjdk 1.8.0_151)
- Python 2.7.12 & 3.5.2

The program must compile and run on this UNIX machine.

Submission Elements

1. **Submit the program files electronically on eCampus by the due date.**
   1.1. A comment block at the beginning of the file(s) must contain the same information provided in the cover and status sheets, described in 2.1 and 2.2.
   1.2. In addition, include a README.TXT file which describes exactly how the program is compiled on the LCSEE UNIX server shell.
   1.3. Zip in a single file all program files, output data files, and the README.TXT file and submit the zipped file in eCampus.

2. **Submit the following paper copies in the beginning of the class, on the due date:**
   2.1. A cover sheet stating student's name, WVU student ID number, the programming assignment number, and the date.
   2.2. A status sheet which states if your program does not compile or, for working programs, specifies clearly any aspect that does not work properly. This information must be accurate and up to date.
   2.3. Listing of your code (using small font, e.g. 10 point). Please make sure that your code is commented (inline comments) to explain what parts of the program do.
   2.4. Documentation must include
      2.4.1. Description of the programming environment (language, system, all the assumptions such as libraries, system resources, etc.)
      2.4.2. Detailed description which will allow the reader to understand your approach and algorithms used.

3. **Grading**
   Programs that cannot be compiled will receive at most 40% of the assignment grade, assuming significant amount of functionality has been implemented and adequate documentation was provided. Only well written, nicely structured, well documented programs, which produce correct output will receive full credit.

4. **Academic Honesty**
   The work submitted for the programming assignment must be your own work. You are not allowed to use any publicly available programs nor programs written by your peers. For the detailed policy of West Virginia University regarding the definitions of acts considered to fall under academic dishonesty and penalties for academic dishonesty, please see the West Virginia University Academic Standards Policy (http://catalog.wvu.edu/undergraduate/coursecreditstermsclassification).

Additional Resources

https://www.youtube.com/watch?v=NmM9HA2MQGI

https://markusholtermann.eu/2016/09/ssl-all-the-things-in-python/

https://wiki.openssl.org/index.php/TLS1.3

https://www.cloudflare.com/learning/ssl/what-happens-in-a-tls-handshake/

https://blog.cloudflare.com/rfc-8446-aka-tls-1-3/

https://medium.com/@sadatnazrul/diffie-hellman-key-exchange-explained-python-8d67c378701c

https://blog.qualys.com/ssllabs/2013/06/25/ssl-labs-deploying-forward-secrecy

Cipher Suites: https://www.cloudinsidr.com/content/tls-1-3-and-tls-1-2-cipher-suites-demystified-how-to-pick-your-ciphers-wisely/