CS6500 - Network Security
Assignment 2: PKI-based key establishment

Instructor: Krishna M. Sivalingam
Assigned on: *Feb. 16, 2022*
Due: *Mar. 5, 2022, 11PM*; 10% Penalty – *Mar. 6, 2022, 11PM*;

The objective of this assignment is to implement a minimal Public Key Infrastructure-based key establishment (exchange) for use with symmetric key encryption algorithms.

There will be two programs: (a) Public Key Infrastructure (PKI) Certificate Authority (CA), and (b) Client (C). At start of execution, open at least 3 Terminal windows, one each for the CA and two clients. In each terminal, invoke the corresponding program as described later. All communications between the CA and the clients will be via TCP sockets.

First, the CA generates a 2048-bit RSA public-private keypair - denoted as $\mathrm{PU}_P, \mathrm{PR}_P$. The public-key is assumed to be known to call clients. Each client (e.g. A) also generates a 2048-bit RSA public-private keypair - denoted as $\mathrm{PU}_A, \mathrm{PR}_A$.

1 Certificate Authority (CA)

A client sends a certificate request to the CA, with its public-key relevant parameters. The CA will fill all appropriate fields in the certificate, hash the certificate contents, and then digitally sign the hash of the complete certificate with the CA's RSA private key. The CA sends the signed certificate back to the client. The details are provided below.

The CA is invoked as follows:

```
./ca -p portid -o outfilename
```

The **ca** process will listen on port number (*portid*) specified in the command line; the diagnostic output messages (that describe its activities) will be stored in the file named, *outfilename*

For example:

```
./ca -p 12345 -o out.txt
```

A message from a client is received (via the CA server's network socket) with the following three fields:

```
| 301 | ClientPublicKey | ClientNameEnc
```

In the above message:

- the first field ("301") is of type integer; Here, the code 301 indicates that this is a certificate request message from the client to the CA.

- the next field is the Client's RSA Public Key, (with base64 encoding).

  The format of Public Key contents is as given below:

  ```
  -----BEGIN PUBLIC KEY-----
  MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEApq7qCzF2Fzs4NOyrDHxe
  RwKg57bLpcsXwM7b+XFdJ3h8pKjoY44jO6QN50+CWMa89TQlDVG1OjSolrDe/3Ii
  tz7owH2JE/ARqOA7xgnFgmVOpzlKxIHSBszn9/+GBJNiIn2XcvF4vmSKWrFr2o29
  fp/fxv4/KkoBne2vxsmYWI4ibC9wWvOgClg4k5cKFijAM8TL4kocJmuU2B6+iEmv
  +pAAeM9AGXZYSqiarDZfa8zJPuwVZo+GH1w8Ft+BO2GN9TMnBp7RWzPodjYFCyie
  JUiFNMUvWNNv4JUHGSlF6u1tmhSzS9ClcHUX1bEJMUZjkgHWzTNnCTOTqH+Uh6BL
  VwIDAQAB
  -----END PUBLIC KEY-----
  ```

- the next field is base64-encoded encrypted text, using the CA's public key, of the client's name, which is a string of length 6 bytes (only lowercase letters from 'a' – 'z' are permitted), e.g. "ashoka" or "swathi". The CA decrypts this part of the message, using the CA's private key, to obtain the client's name.

The CA then sends a message to the client in the following format:

```
| 302 | ClientName| CertificateInfo
```

The field ClientName is the 6-byte client name (plaintext).
Define:

```
CERTDATA: ClientName || Nonce || Client's Public Key || Start Date || End Date
```

The symbol, '||' denotes concatentation of message components. You are free to use your own message field separator character.
In the certificate details, the end-date is set to 1 year from the start date (date of request). The date format is: YYYY-MM-DD. `Nonce` is a 8-byte random string from the character set, `{'A'-'Z', 'a'-'z', '0'-'9'}`.
The field `CertificateInfo` is made up:

```
CERTDATA || E_PU_A[E_PR_CA [Hash (CERTDATA)]]
```

The hash is first signed using CA's private key and then encrypted using client A's public key. The `sha256` hashing algorithm is to be used.
When the client receives this message with code "302", it extracts the certificate data and then retrieves the hash value using its RSA private key for decryption. It then verifies the digital signature using CA's public key. It then certificate contents stores in a local file for further use. The stored certificate form is:

```
CERTDATA || E_PR_CA [Hash (CERTDATA)]
```

2 Client-to-Client Certificate Exchange and Key Establishment

A client will act either as a sender or a receiver.
If it is a **sender**, the client program is invoked as follows:

```
./client -n myname -m S -a caip -p caport -q clientport
```

Here, *myname* denotes the client's (sender's) name; *inputfile* contains the contents that have to be encrypted and sent to the receiver; *caip* indicates the CA's IP address; *caport* indicates the CA's TCP port number; *clientport* indicates the port number that this client's TCP server is listening to.
If it is a **receiver**, the client program is invoked as follows:

```
./client -n myname -m R  -i inputfile -d senderIP -q senderport
   -s outenc -o outfile -a caip -p caport
```

Here, *myname* denotes the client's (receiver's) name; *inputfile* denotes the file requested from the sender; *senderIP* and *senderport* denote the sender's IP address and port number; *outenc* contains the encrypted contents received by the receiver; *outfile* contains the decrypted contents received by the receiver; *kdcip* and *kdcport* are as described above.
The client's activities are:

- Generate its public-private keypair and initiate the registration with the CA, as explained above.
- After completion of initialization step, the client will sleep for 15 seconds, to ensure that all other clients are registered.
- If the client is a receiver (R), it will initiate the protocol for client-to-client communication and encrypted data transfer, as given below.

  ```
  R -> S: 501 | ReceiverName
  S -> R: 502 | SenderName | SenderCertificate
   (R verifies the sender certificate signature using CA's public key.)
   (Next, R generates a session key K_s and encrypts this key using S's Public Key)
  R -> S: 503 | EncSessionKey | FileName
  S -> R: 504 | FileName | EncrFileContents
    (R outputs decrypts contents)
  ```

  For each step of the interaction, the sender and receiver will print to the screen, appropriate messages confirming the activities taken.
- If the client is a receiver, the encrypted data file sent by the sender will be received and stored in the receiver's output file (specified in the command line).
- Any other needed specific message formats are to be designed by you and explained in the report.

3 Sample Run

3.1 CA Terminal Window

Assume that the CA is running on a system with IP address of `10.21.22.23`.

```
Prompt> script cascript
System outputs: Script started, output file is kdcscript
Prompt> ./ca -p 12345
...
Starts TCP server to listen on port 12345
Wait for messages from clients..
Type Ctrl-C to finally quit
Prompt> exit
System outputs: Script done, output file is cascript
Prompt> cat cascript
```

3.2 Sender Client Terminal Window

Assume that the sender is running on a system with IP address of `10.21.22.24`.

```
P> script sendscript
...
P> ./client -n ashoka -m S -a 10.21.22.23 -p 12345 -q 23456
...
Generates Public Key..
Sleeps for 15 seconds ..
Contacts CA and requests Certificate ..
Receives certificate from CA ..
Sleeps for 15 seconds ..
Waits for request from Receiver ..
Follows file transfer protocol...
Quits after sending the file to bob.
P> exit
System outputs: Script done, output file is sendscript
P> cat sendscript
```

3.3 Receiver Client Terminal Window

```
Prompt> script recvscript
...
Prompt> ./client -n swathi -m R -i movie1.txt -d 10.21.22.24 -q 23456
 -a 10.21.22.23 -p 12345  -s outenc.txt -o out.txt

...
Sleeps for 30 seconds ..
Contacts Sender for file transfer
Quits after recving the file from alice.
Prompt> exit
System outputs: Script done, output file is recvscript
Prompt> cat recvscript
```

4 What to Submit on IITM Moodle

A single tar.gz file with name ROLLNO-Lab2.tgz containing:

- Source files and Makefile
- A README File that explains how to compile and run the programs; whether your programs works correctly or whether there are any known bugs/errors in your program.
- Using the "script" Linux command, record the session from the Terminal window for CA, sender and receiver; thus, there will be three typescript files, named as above.

Your system may be tested with the CA, Sender and Receiver running on different machines (or) virtual machines on a given system.

There should be two tests: (i) showing the correct certificate sent to the receiver that passes the signature verification; and (ii) showing an incorrect certificate (a few random bytes in the sender's name can be modifed) sent to the receiver that fails the signature verification.

5 Grading

- CA implementation: 34 points
- Sender implementation: 25 points
- Receiver implementation: 15 points
- Generation of typescript for 2 tests: 16 points
- Viva Voce Session (to answer questions about the program, demo. of running code, etc.): 10 points

6 Policies

- This is an INDIVIDUAL assignment. Please refer to first day handout regarding penalties for any form of academic dishonesty, plagiarism, etc. There should be no downloaded code. You should not communicate/share any code with any other student in the class or elsewhere.
- Plagiarism Checking Software will be used.