San Jose State University
ELECTRICAL ENGINEERING DEPARTMENT

EE 282/CMPE209

SECURE CLIENT-SERVER FILE TRANSFER

PART – 2

"..We learn by doing.."

1. General Description and requirements

In this experiment you will implement a secure TCP client-server system. Your implementation would modify the previous experiment, Experiment1, by providing the following security services:

a. user-password is not vulnerable to dictionary attacks
b. mutual authentication of the server and the client
c. confidentiality of the transmitted file
d. integrity of the file received

2. Client-Server application:

The security features are based on using EKE (Encrypted Key Exchange) protocol where a file is securely transmitted from the server to client, only after successful mutual authentication.

3. Requirements and Specifications:

Similar to Experiment 1, there are two pieces of code that you have to implement in two different files, preferably hosted in different machines: a client and a server. Below are the specifications which provides the details you need to incorporate in these programs (refer to the figure below):

1. Message1: is the client request which consists of the ClientID, and DH parameters being encrypted with client password.

a. The DH parameters are :

p=197221152031991558322935568090317202983, g=2

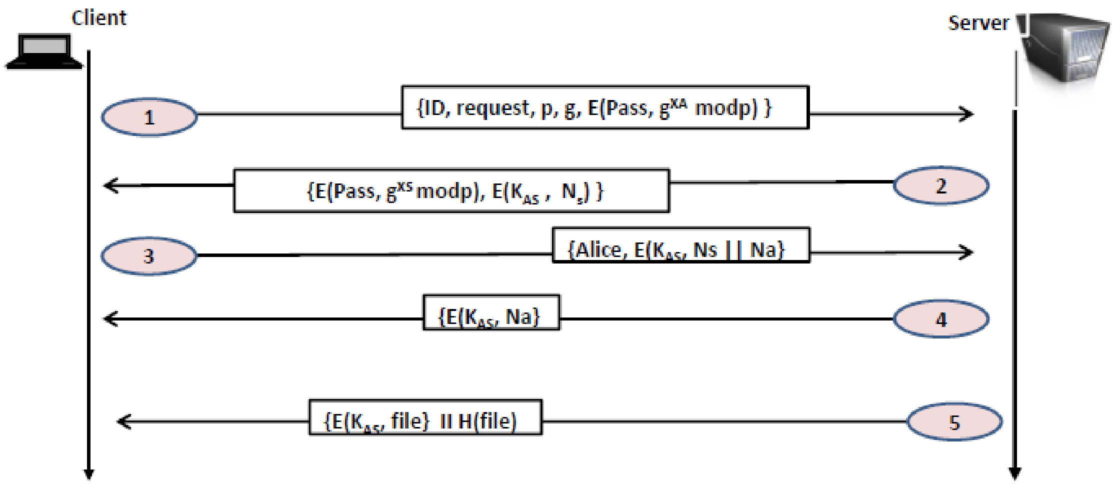
b. Your design should allow the testing of different users secrets (X_A, X_S), and Nonce values (N_a, N_S) which must provided as input parameters.
c. Your coding should allow the testing of different values for p and g parameters (either defined as global variables, or preferably, as input values sent from the client to the server)
d. The Client and Server secrets can be any random numbers.
e. To generate random numbers use Blum-Blum-Shub PRNG discussed in class.
f. For padding and whenever need it, use bit-sequence "000000.."
g. To optimize CPU execution of your overall design, you are encouraged to exploit some of the techniques we learned in class (e.g Fermat's theorems).

h. clientID (also serverID) is a string not to exceed a length of 40 characters.

i. The shared secret key, *PASS*, between the server and the client, can be any word or combination of words but needs to be converted to a string comprised only of hexadecimal.

j. The encryption algorithm during DH key generation phase (Message1) is an XOR operation of *PASS* and the message being exchanged (g^{XA}). For simplicity you may use padding of "0000.." to carry out the XOR operation. *[Note: According to EKE developers, "it is sufficiently secure to use XOR as the cipher algorithm since the exchanged messages during the DH phase are numbers", as we also discussed in class]*

2. Message 2: consists, mainly, of 2 ciphertexts:
   a. The 1st ciphertext is encrypted with *PASS* key as in Message 1.
   b. The 2nd ciphertext consists of the random number, nonce N_S, of size equal to at least the cipher algorithm bock-size.
   c. The encryption algorithm used in 2nd ciphertext, as well as for the rest of the exchanged messages, is *AES (128)*.
   d. In this project we <u>must use</u> OpenSSL toolkit implementation and its available library to execute AES and any other crypto-protocols needed (please do <u>not</u> use any other third-party crypto-implementations). OpenSSL [1] is the industry-academia de-facto open standard for the implementation of SSL and TLS protocols. You are provided with the skeleton template file, *tempAES.c*, that you would need to use and modify to perform the encryption and decryption functions of AES. The *helpme.txt* file provides more information how to use *tempAES.c* and other functions that you would need to use to complete the project.


Client
Server
1
{ID, request, p, g, E(Pass, g^{XA} modp) }
2
{E(Pass, g^{XS} modp), E(K_{AS} , N_s) }
3
{Alice, E(K_{AS}, Ns || Na}
4
{E(K_{AS}, Na}
5
{E(K_{AS}, file} II H(file)

Secure TCP File transfer using EKE protocol

(see also lecture notes)

3. Message 3: the client uses AES with DH generated-shared key to send the encrypted message that consists of the catenation of N_S and its own nonce *Na*.
4. Message 4: consists of AES (128) ciphertext of N_a
5. Message 5: consists of the ciphertext requested file along with its message digest. For this project we will assume the requested file is *helpme.text*. As for the message digest function, H(M), we use SHA-1, from OpenSSL implementation. Similarly to AES you are provided with skeleton template file, *tempSHA1.c*, that you would need to use and modifies to perform the SHA1 functions. The *helpme.txt* file also gives more information on how to use *tempSHA1.c.*

4. **Procedure:**

Please also refer to Expirement1. You can leverage your previous work and codes

Step 1: Client connects to the server using the IP address and port number specified, you can do this by either passing command line arguments or by hard-coding them in your code.

Step 2: Server should authenticate the client.
Following the DH key generation, the server creates a login page on the client side, the credentials are to be sent to the server for authentication. You need to create a list of username/password combinations, at least 10, and store them in a database on the server side or you can just have one username/password combination hard-coded into your code, choice is yours. The server should use this to verify the credentials.

Step 3 : In case of invalid entries you can give the user 3 attempts to login.
After 3 attempts close the connection from the server side saying access denied.
Step 4: the server transmits the file and its MD to clients. The ciphertext consists of different blocks, so AES ECB is assumed
Step 5: the client need to verify the integrity of the file.
Step6: implementation must define obvious error responses that include responses for at least the following errors: (a) client authentication-error, sent from server to the client, (b) DH shared key mismatched-values, can be sent from either the server or the client, and (c) integrity-file error sent from the client to the server. In all error messages three attempts should be given before the session terminates

5. Submissions:

- Submit a report which provides detailed explanations of your implementation of EKE protocol and observations that are interesting or surprising. It includes:

    1. Wireshark traces of messages exchanged between the client during the a successful session.

    2. Snaps of your output sample for the experiment in Step2 (successful authentication, as well as Step6 (error responses)

    3. Need to show all messages values send/ received during each exchange at both sides, client and servers.

    4. All source codes used to complete the experiment. Include enough explanations/comments on your coding so others can easily follow the logic steps of your algorithms. Define all variables (systems and yours) used and their purposes (recall the common good programming habits: there is no such thing as "too much comments", also choose good names for your variables, that reflect their purposes…)

    5. Submit all of the above items in Canvas. Please protect your work. Canvas can detect similar work which would result in receiving zero or failing grade in the course.

    6. In few lines provides the security analysis of the this security system just implemented outlining, very briefly, the its main strengths and weaknesses (similar to the analyses done in class; comparing to original DH) in particular at what steps the server detected invalid user (Trudy)

    7. A demo of your work is mandatory for this part. Individual Demos are scheduled on the last day of our class, Thursday, May 12, 2016 (FCFS).

- Name your files as *name1.name2.PartxxappxxxProtocol.c* where *name1*, name2, are names of the three members of your group, *Partxx* is the project part being submitted, *app* is the client or server application, and *Protocol* is usually TCP. For example: *sanjay.rtorresg.govinda.Part2clientTCP.c*

5. Background and References:

1. http://www.openssl.org/ (official site to learn more about OpenSSL)

Other supporting material will be provided on Canvas, as needed.