1. General Description

In this Project, you will implement a secure client-server system. You may leverage my code (I will give it to you if you accept) There are two key pieces to this project: (i) authentication of the client and server using a simplified Kerberos protocol; and (ii) file transfer from server to client, with transmission encrypted by the server.

2. Server – Client application: Requirements:

There are three pieces of python code you have to implement in three different files -- a client, an authentication server and a server. We will give you the high level details of the protocol.

Authentication:

In the description that follows, Kc, and Ks are secret keys shared by the Authentication Server with the client and the server respectively. Kc,s is a secret key generated by the Authentication Server, and shared between the client and the server. Furthermore, the packet types that contain the information for each step of the authentication protocol will be presented with the description below (Refer to Figure #1):

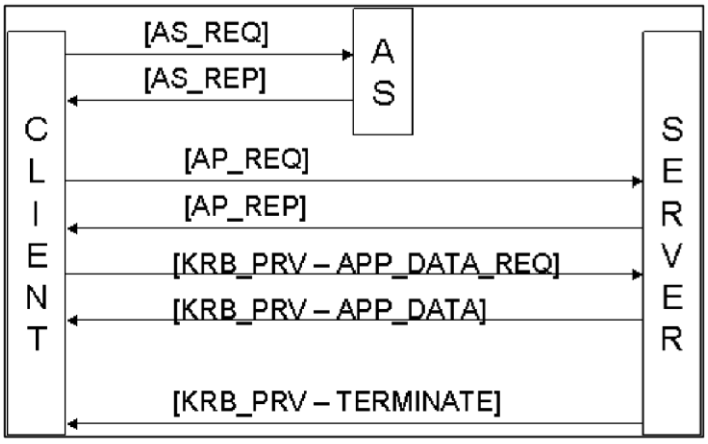

Figure #1 Functional specification

1.

The client sends a message to the AS of the form (AS_REQ message):

<ClientID> <ServerID><TimeStamp1>

The AS responds to the client with a message of the form (AS_REP message):

E_{Kc}[Kc,s || ClientID||ServerID || TimeStamp2 || Lifetime2 || Tkt]

The Tkt is of the form:

E_{Ks}[Kc,s || ClientID || ClientIP || ServerID || TimeStamp2 || Lifetime2]

The client sends the server a message of the form (AP_REQ message):

Tkt || Authenticator

The Authenticator is of the form:

$E_{Kc,s}$[ClientID || ClientIP || TimeStamp3]

The server returns a message to the client of the form (AP_REP message):

$E_{Kc,s}$[TimeStamp3+1]

2.

3.

4.

Data Transmission:

The server transmits data to the client, encrypted with the secret key Kc,s that was established between them during the authentication process. The packet types that contain the information for each step of the data transmission protocol will be presented with the description below (Refer to Figure #1). Please note that each of the packets mentioned below is encrypted and then transmitted through the network using UDP protocol:

1. The client transmits an APP_DATA_REQUEST packet to the server.
2. Once the server verifies the packet type to be a APP_DATA_REQUEST, it reads the file to

   be transmitted. The file is broken into several segments (depending on the size of the file), and each segment is stored in a APP_DATA packet. This packet is then encrypted using the secret key Kc,s, using a L-AES-128 encryption and OFB mode, and using a random IV.

3. When the server completes sending the file, it will transmit a TERMINATE packet which marks the end of the file download. Included in this packet, is a file digest (SHA1 digest) that the client will use to verify the integrity of the received file.

Notes

- • You may assume that the server handles only one client at a time and need not address issues associated with supporting multiple simultaneous clients.
- • You can use UDP protocol for all the message in the authentication and data transfer phases.
- • The encryption used for the authentication and data transfer phases is a **L-AES-128**

encryption and **OFB** mode

- The client, server and AS can run in the same machine or different machines.

Command Line Arguments:

- Your authentication server code must be executed from the command line as follows:

./authserver <authserverport> <clientID> <clientkey> <serverID> <serverkey>

<clientID> and <serverID> are strings not to exceed a length of 40 characters. <clientkey> is the secret key shared between the authentication server and the client, and <serverkey> is the secret key shared between the authentication server and the server. The keys must be represented as a string comprised only of hexadecimal digits.

- The server code must be executed from the command line as follows:

./server <server port> <authserverkey> <input file>

<authserverkey> is the secret key shared between the server and authentication server. The keys must be represented as a string comprised only of hexadecimal digits. The input file is the path to the file that the server will send to the client.

- The client code must be executed from the command line as follows:

./client <authservername> <authserverport> <authserverkey> <server name> <server port> <output file> <clientID> <serverID>

The <authserverkey> is the secret key shared between the client and the authentication server. The keys must be represented as a string comprised only of hexadecimal digits. The output file argument is the file name to assign to the file the server will send to the client. <clientID> and <serverID> are strings not to exceed a length of 40 characters.

If any of the arguments is invalid, your code should return an appropriate message to the user. Be sure to consider the case when the keys are invalid.

3. The output:

- - A report of the project with screenshots to show how your program implements the authentication and file transfer step by step.
- - The server.py, client.py, and authserver.py source code.
- - Readme file showing how to run your code.