BBM465 – INFORMATION SECURITY LAB.

ASSIGNMENT 4

23.10.2024

Group ID: 15

ECE NUR TAŞ – 2210356039

BÜŞRA BURHAN – 2220356051

PROCESS OF CODE

1. Accessing Interface Data (Application):

- The interface data is accessed.
- Necessary functions are called to start the process.

2. Registration Process:

- The register functions in the KDC class are called.
- The server and client are registered:
    - **RSA public and private keys** are generated for both parties.
    - These keys are added to the respective maps stored in KDC.
- Server and client objects are added to their respective maps based on their IDs:
    - allClients: <clientID, clientObject>
    - allServers: <serverID, serverObject>
- The password and ID information for the client and server are saved in **Database.csv**.
- The relevant maps (registeredServers for the client and registeredClients for the server) are updated as follows:
    - serverID and clientID values are added.
    - The value is set to "full" (ticket information will be updated during login).

3. Login Process:

- The following steps are carried out during login:
    1. The provided client ID is checked in the allClients map, and its object is found.
    2. The specified server ID is checked in the registeredServers map of the client object:
        - If the key is "full" and no ticket has been added yet, the process continues.
    3. The user's password is validated against Database.csv.
    4. Upon successful validation, a ticket is created through TicketGrant:
        - Two ticket objects are created:
            - **For the client:** A ticket encrypted with the client's public key.
            - **For the server:** A ticket encrypted with the server's public key.
    5. These tickets are added to the maps as follows:
        - For the client: registeredServers[serverID] = ticket.
        - For the server: registeredClients[clientID] = ticket.
- If a map's value is already set to "ticket," the login is considered complete.

4. Communicate (Messaging) Phase:

- "Access to server" allows access to the target server object through allServers.
- The **Session Key** is compared:

  Clients decrytes its own session key with function from kdc and sends it to server by accessToServer function after that server decrytes its own sessio key and checks if ticket is still valid . Then comapres these two session key and if it is same saves the message .

EXPLANATION OF CLASSES

1. Client Class

Purpose: Represents a client in the Hybrid Kerberos System. Handles client registration, ticket storage, and secure communication with the server.

Fields:

- **clientId**: The unique identifier for the client.
- **password**: The password used for client authentication.
- **registeredServers (HashMap<String, Ticket>)**: Stores tickets for the servers the client has access to, mapping server IDs to Ticket objects.
- **allClients (static HashMap<String, Client>)**: A static map to store all registered clients by their ID.

Functions:

- **Client(String clientId, String password)**: Constructor that initializes the client with an ID and password.
- **accessToServer(String serverId, String message)**: Handles communication with a server. Checks if a ticket exists and is valid, renews the ticket if expired, or denies access if not logged in.
- **getClientId() / getPassword()**: Return client ID and password, respectively.
- **setTicket(String serverId, Ticket ticket)**: Associates a ticket with a server for this client.
- **isRegistered(String serverId)**: Checks if the client is registered with the specified server.
- **setRegisteredServers(String serverId)**: Adds a new server to the registeredServers map with no ticket initially.
- **getRegisteredServers()**: Returns the map of registered servers.

2. Server Class

Purpose: Represents a server in the Hybrid Kerberos System. Manages client tickets and handles client-server communication.

Fields:

- **registeredClients (HashMap<String, Ticket>)**: Stores tickets for the clients registered with the server, mapping client IDs to Ticket objects.
- **allServers (static HashMap<String, Server>)**: A static map to store all registered servers by their ID.
- **serverId**: The unique identifier for the server.

Functions:

- **Server(String serverId)**: Constructor that initializes the server with a unique ID.
- **getServerId()**: Returns the server ID.
- **requestFromClient(Client client, String sessionKey, String message)**: Validates the session key and allows communication if valid.
- **setTicket(String clientId, Ticket ticket)**: Associates a ticket with a client for this server.
- **setRegisteredClients(String clientId)**: Adds a new client to the registeredClients map with no ticket initially.
- **getRegisteredClients()**: Returns the map of registered clients.

3. Ticket Class

Purpose: Represents a secure ticket containing an encrypted session key and timestamp for ticket expiration.

Fields:

- **encrypted**: The session key encrypted using RSA.
- **currentTime**: The timestamp when the ticket was created.

Functions:

- **Ticket(String encrypted)**: Constructor that initializes the ticket with the encrypted session key.
- **isValid()**: Checks if the ticket is still valid based on a 5-minute expiration window.
- **getEncrypted()**: Returns the encrypted session key.
- **getCurrentTime()**: Returns the ticket's creation timestamp.

4. TicketGrant Class

Purpose: Extends KDC and generates session tickets for secure communication between clients and servers.

Functions:

- **generateTickets(Client client, Server server)**: Generates session keys using AES, encrypts them with RSA, and creates Ticket objects for the client and server.
- **generateAESKey()**: Creates a 128-bit AES session key.
- **encryptRSA(String data, PublicKey publicKey)**: Encrypts data using RSA and the provided public key.

5. KDC Class

Purpose: Acts as the central Key Distribution Center. Handles client/server registration, authentication, and key management.

Fields:

- **fileName**: Path to the CSV file storing client data.
- **clientKeysDatabase / serverKeysDatabase (static HashMap<String, KeyPair>)**: Store RSA key pairs for clients and servers.
- **tgs (static TicketGrant)**: Handles ticket generation.

Functions:

- **registerClient(Client client)**: Registers a client, generates RSA keys, and stores them in a CSV file.
- **registerServer(Server server)**: Registers a server and generates its RSA key pair.
- **generateKeys()**: Generates an RSA public-private key pair.
- **saveToCSV(String fileName, String clientID, String password, KeyPair pair)**: Saves client details, including their keys, to a CSV file.
- **authenticateClient(String clientId, String password, String serverId)**: Authenticates a client by checking the CSV file and generates a ticket if valid.
- **decryptRSA(String encryptedData, Client/Server)**: Decrypts data using the private key of the client or server.
- **newTicket(Client client, Server server)**: Creates new tickets for the client-server pair.This function is just used for to connect with ticketGrant