Two Protocols

Secure RPC

Here Anna is trying to authenticate herself to the server Server. She is logging into the client Client. Client and Server communicate. The protocol as implemented by Sun Microsystems and other vendors uses Diffie-Hellman for the key exchange, but this protocol would actually work with any symmetric key exchange protocol.

1. Client sends Anna's name to Server.
2. Server sends the Server's public key, Anna's public key, and Anna's private key enciphered under the DES with her password as the key to the Client.
3. Client obtains Anna's password and deciphers Anna's private key using that password.
4. Client computes the conversation key (the protocol's term for an interchange key) using Anna's private key and Server's public key.
5. Server computes the same conversation key using Server's private key and Anna's public key.
6. Client generates a 56-bit random session key, appends a Validator in fixed form, enciphers the result using the conversation key, and sends it to Server.
7. Server uses its computed conversation key to decipher message.
8. Server checks Validator for presence of specific information (bits in specific places, *etc.*) If the Validator is correct, Server believes Anna has correctly authenticated herself, and both parties use the session key for further messages. If the Validator is incorrect, Server believes Anna has not correctly authenticated herself, and rejects the attempt.

Kerberos

Kerberos is an authentication protocol requiring one or more trusted authentication servers. The authentication server stores the user's password. In what follows, $T_{n,x}$ is a ticket identifying the user n to the service x, A_n is an authenticator identifying the user n, $K_{n,x}$ is a session key that n and x can use to communicate, and K_x is a cryptographic key (password) that user or service x and the authentication server share. The ticket $T_{n,x}$ contains the session key $K_{n,x}$. The notation $\{x\}y$ means the message x enciphered with cryptographic key y (using the DES).

Logging in

1. Client sends user name and request for a ticket to use the *ticket granting service* to the authentication server.
2. Authentication server responds with $\{\{T_{user,tgs}\}K_{tgs},K_{user,tgs},\ldots\}K_{user}$
3. Client uses locally entered user password K_{user} to decipher message and obtain $\{T_{user,tgs}\}K_{tgs}$ and $K_{user,tgs}$. If the password is incorrect, none of the messages between the authentication server and the user will decipher correctly, nullifying the use of any services.

Obtaining a Ticket for a Service

1. Client sends message: $service,\{T_{user,tgs}\}K_{tgs},\{A_{user}\}K_{user,tgs}$ to the ticket granting service *tgs*.
2. The ticket granting service uses its key K_{tgs} to obtain the ticket $T_{user,tgs}$and, from that, the session key $K_{user,tgs}$. It uses the session key to obtain the authenticator A_{user}. It then verifies that the ticket was issued to the entity named in the authenticator.
3. The ticket granting service replies to the client with the message $\{\{T_{user,service}\}K_{service},K_{user,service}\}K_{user,tgs}$.
4. The client uses the session key $K_{user,tgs}$ to obtain the session key $K_{user,service}$ to use with the desired service, and an enciphered message $\{T_{user,service}\}K_{service}$ to send the service.

Using a Service

1. Client sends message: $\{A_{user}\}K_{user,service},\{T_{user,service}\}K_{service}$ to the server.
2. The service uses its key $K_{service}$ to obtain the ticket $T_{user,service}$ and, from that, the session key $K_{user,service}$. It uses the session key to obtain the authenticator A_{user}. It then verifies that the ticket was issued to the entity named in the authenticator.
3. The server obtains the timestamp t from $T_{user,service}$ and replies with $\{t+1\}K_{user,service}$.