1.

2. A man-in-the-middle attack on a Diffie-Hellman key exchange works by having the attacker (person C) intercept all key exchanges between person A and person B and setting up their own key exchange between the two (creating one shared secret key with A, and another secret key with B). During the initial key process, p and g are shared unknowingly with C, with C simply passing along these values to the other party. When A shares their public key, C uses it to make a shared secret key with A, and when B shares their public key, C uses it to make a separate shared key with B. Then, whenever one of the parties, say A, passes a message meant for the other, C decrypts it the shared key created with A, and then re-encrypts it with the shared key created with B before passing the message along to B.

Station-to-station combines Diffie-Hellman with digital signatures for authentication. After selection the values for p and g, this protocol works by having person A initiate the key exchange by sending their public key person B. B then calculates their public key and the shared secret key K, and then signs a message containing both A and B’s public key with B’s private RSA key, and encrypts the message with K. A then uses the information in this message to calculate the shared secret key and sends a confirmation message back to B, containing A and B’s public keys, signed with A’s private RSA key, and encrypted with K. The RSA key authentication is what fixes the previous vulnerability. Attacker C is unable sign the back-and-forth messages with A or B’s private RSA keys. And if they attempted to sign them with their own private RSA keys, the signature could only be confirmed with C’s public RSA key, not A or B’s, revealing the attack.

3. The authentication server verifies the user’s access rights in the database. If the verification is successful, the authentication server creates a ticket-granting ticket and a session key, encrypts these using a key derived from the user’s password, and then sends it back to the user. The user can then use these after decrypting them by providing their password.

After the ticket and session key are decrypted, the ticket and an authenticator containing the user’s name, network address, and the time are sent encrypted to the ticket granting server. The TGS then decrypts the ticket and authenticator and verifies the request. If the verification is successful, the TGS creates and returns to the user a ticket for the requested service.

The separation of the authentication server and the ticket-granting server enables Kerberos to function as a Single Sign On system. The authentication server authenticates the user once with a long timeout (typically 8-10 hours, workday length or so) and grants the user a TGT ticket that can be used throughout that period. This ticket is then given to the TGS whenever a new service needs to be accessed, or previous access to a service has timed out. Overall, the system is designed to allow effective access control to a variety of services while only requiring the user to authenticate themselves once.

4.

Step 1: The user logins into the domain and requests services.

Step 2: This request is handled by the authentication server. After verification, a ticket-granting ticket (**TGT**) containing a session key (**SK1**) is sent back to the user.

**TGT**: Long term ticket that will be given to the ticket-granting server and allows the user to obtain service tickets.

**SK1**: Combined with the TGT the session keys form a set of credentials. Determines how long a session is valid for and can also be used when encrypting and decrypting messages.

Step 3: User decrypts message containing **SK1** and **TGT**, then sends the **TGT** and authentication information to the ticket-granting server (TGS), encrypting the message with **SK1**.

Step 4: TGS decrypts and verifies the request containing the user’s **TGT**. The TGS then sends a new **service specific ticket** and session key (**SK2**) back to the user.

**Service** **Ticket**: Access ticket to a specific service.

**SK2**: Specifies how long a session is valid for and is used to encrypt the request to the service.

Step 5: User decrypts message from the TGS using **SK1** and sends the **service ticket** and authentication information to the service being requested. This message is encrypted using SK2.

Step 6: The service verifies the **service ticket**, **SK2**, and authentication data and **grants access** if it checks out.