Mediated Authentication:

Different from mutual authentication, in mediated authentication, a trusted KDC (Key Distribution Center) is responsible for shared secret generation and distribution between Alice and Bob. Alice and Bob each share a secret key and use it to secretly communicate with KDC. The principal of mediated authentication is as following:

![Figure 5. KDC principle](image)

The above protocol however is not efficient due to the distribution of $K_{AB}$ to Bob: firstly, Bob is likely a server and often busy; secondly, Bob might receive $K_{AB}$ much later - Alice’s request might arrive before.

To eliminate the explicit message to Bob that includes $K_{AB}$, the mediated authentication protocol can be changed to the following protocol:

![Figure 6. KDC in practice](image)

**Needham-Schroeder Protocol:**

NS protocol is a classic mediated authentication protocol designed by Needham and Schroeder. It similar to the protocol described above. However, in the NS protocol, after establishing the shared secret $K_{AB}$, mutual authentication is performed using additional messages (3,4,5):
Why does NS protocol use N1 (nonce)?

N1 is used to assure Alice is really talking with KDC but not Trudy. For example, if for some reason, Bob has lost his key $K_{Bob}$, Trudy can impersonate KDC and listen to Alice request as a fake KDC and use an old recorded message $K_{Alice} \{ \text{Bob}, K_{AB}, \text{Ticket} \}$ to reply to Alice. Next, Trudy impersonates Bob. When Alice sends message 3 to Bob, Trudy gets the message and is able to perform the rest of the authentication because it know $K_{Bob}$ (and $K_{AB}$).

Why “Bob” in (2)?

If Trudy eavesdrops and changes “Alice wants Bob” to “Alice want Trudy”, without “Trudy” in the reply message from KDC, Alice has no idea that she has received a $K_{AT}$ ($K_{Alice-Trudy}$ but not $K_{AB}$) and uses this $K_{AT}$ to actually communicate with Trudy (now impersonating Bob).

The original NS protocol has additional vulnerabilities when not implemented carefully.

It is not only the protocol but also the encryption method (or other factors) that can make authentication methods vulnerable to attacks. For example, if DES in conjunction with electronic code book is used to encrypt the messages between Alice and Bob, than $K_{AB}\{N_2 - 1, N_3\}$ is essentially $K_{AB}\{N_2 - 1\}$, $K_{AB}\{N_3\}$. Trudy can actually now impersonate Alice by causing a reflection attack on Bob.

Expanded NS protocol:

If Alice somehow loses her $K_{Alice}$, then by recording messages (1) (2), Trudy can figure out $K_{AB}$ and the ticket and can impersonate Alice. This is because Bob does not know that Alice has lost $K_{Alice}$. It will still accept the ticket because there is nothing wrong with it. To prevent this, the expanded NS protocol adds two additional messages to ensure that when Bob receives a ticket from Alice, he knows that Alice has been to the KDC first.
Without going through the KDC, the ticket Alice sends to Bob cannot include a correct $N_B$ and will be considered invalid by Bob.

**Session Keys:**

Session key is the key used only in one communication session. The benefits of using session keys instead of long term keys are:

- Long term keys can age.
- Trudy can replay packets encrypted using the long terms keys from old sessions into newer sessions.
- Session keys can guarantee PFS (perfect forward secrecy) when using Diffie-Hellman.
- Session keys can be assigned to untrusted software without worrying about exposing the long term secret key.
- Session keys can be computationally efficient (when long term public keys are used for authentication and the session key is a shared secret).

Authentication protocols, in addition to authenticating both the sides using long term keys, must also result in session key establishment. The session key or session keys are then used to secure the rest of the communication (and also not allow the session to be hijacked after the authentication.

What’s a good session key?

$\{K_{AB}\}{R + 1}$ is not a good session key. $\{K_{AB}+1\}{R}$ is a good session key but doesn’t offer perfect forward secrecy. Why?
Assume authentication has already taken place between Alice and Bob, and now we’d like to establish a session key. They each have private and public keys.

\[
\begin{array}{c}
A \\
K_A^* \quad K_B^*(S) \quad K_B^* \\
K_A^- \quad K_B^-
\end{array}
\]

There are many problems with \(K_B^*(S)\). For example Trudy can replace \(S\) with \(S_T\) and send \(K_B^*(S_T)\) instead.

What if Alice signs her message?

\[
\begin{array}{c}
A \\
K_A^* \{ K_B^*(S) \} \\
K_A^- \{ R_A \} \\
K_B^* \{ R_B \}
\end{array}
\]

This way everybody can verify the integrity of Alice’s message with Alice’s public key, but there are two problems with it:

1. The session key is encrypted using a long-term secret, therefore, there is no perfect forward secrecy.
2. If somebody breaks into Bob, and steals \(K_B^-\), then he/she knows the session key.

What if Alice chooses random number \(R_A\) and Bob chooses \(R_B\)?

\[
\begin{array}{c}
A \\
K_B^* \{ R_A \} \\
K_A^* \{ R_B \}
\end{array}
\]

Now, they agree on session key \(R_A \times R_B\). The problem with this would be that Trudy can change one or both random numbers \(R_A\) and \(R_B\) to \(R_A^T\) and \(R_B^T\), and she can cause disruption. However, she cannot make Alice and Bob to settle on a key of her choice. The reason being that Bob thinks the session key is \(R_A^T \times R_B\), while Alice thinks the key is \(R_A \times R_B^T\).

If Trudy breaks into Bob and find \(K_B^-\), but cannot determine \(R_A \times R_B\).

**Session Keys Using Diffie Hellman:**

\[
\begin{array}{c}
A \\
K_B^* \{ g^{S_A} \ mod \ p \}
\end{array}
\]
This will produce the session key $g^{SA \cdot SB \mod p}$. Once the session is over and $S_A$ and $S_B$ forgotten after that, there is no way for Trudy to get the $S_A$ and $S_B$.

--- This was chapter 11! ---

**Lamport’s Hash Function:**

It is not currently in use, only on some legacy systems perhaps. But, the concepts are still useful.

Note: $\text{hash}(\text{hash}(\text{hash} \text{(passwd)})) = \text{hash}^3 \text{(passwd)}$

Here, Bob already knows:

- Alice
- her $\text{hash}^n \text{(passwd)}$
- $n$

Now, Bob computes and compares $\text{hash}(x)$ with $\text{hash}^n \text{(passwd)}$ that he already has in database. If there is a match, then Bob authenticates Alice, reduces $n$ by 1 and updates database, replacing $n$ by $n-1$. He also replaces $\text{hash}^n \text{(passwd)}$ with $x=\text{hash}^{n-1} \text{(passwd)}$. This method works, since you can compute $\text{hash}^n \text{(passwd)}$ from $\text{hash}^{n-1} \text{(passwd)}$, but the reverse is not possible.

Once $n$ is 0, then you have to change the passwd and reset $n$, or you can use salt (in a manner different from its use in Unix passwords).

Along with $n$, one can also store an integer (salt) and actually compute $\text{hash}(\text{passwd | salt})$.

**Positive points:**

- It is robust to eavesdropping.
- Even if somebody breaks into Bob, they still have to figure out password (however,
they can do a dictionary attack.)

Problems:

- If \( n \) is large, there one must deal with the overhead of computing \( \text{hash}^n (\text{passwd}) \).
- It is a one-way authentication.
- It also suffers from the small ‘n’ attack.
  - Trudy impersonates Bob and sends \( n=50 \) to Alice. In return she gets value for \( \text{hash}^{49} (\text{passwd}) \) from Alice. Trudy stores the value.
  - Later on, Trudy impersonates Alice and gets (for example) \( n=100 \) from Bob.
  - Now Trudy can compute \( \text{hash}^{99} (\text{passwd}) \) from previous value of \( \text{hash}^{49} (\text{passwd}) \).

Lamport’s hash, although a secret key-based approach, is robust against eavesdropping AND database break-ins. However, it has other vulnerabilities as discussed before.