CS261

Scribed by Pallavi Joshi

November 18, 2008

1 Key Distribution

Suppose there are n entities, and they all want to be able to communicate securely with each other. A naïve solution would be to have a shared secret for each pair of entities. This would result in $\binom{n}{2}$ (which is $O(n^2)$) shared secrets, or keys. Managing such a large number of keys becomes tedious. Moreover, whenever a new entity joins a network, it has to contact all other entities in the network to establish shared secrets with them.

Instead of having a shared key for each pair of entities, we can have a centralized Key Distribution Center (KDC) that hands out keys to different entities. When two different parties want to communicate with each other, then they can request the KDC to generate a shared key for them. The basic steps present in a protocol that utilizes a KDC are as follows.

1. Let us assume that Alice (A) wants to communicate with Bob (B). She contacts the KDC providing it with her identity, Bob's identity and requests for a key that she can use to communicate with Bob. She also provides the KDC with an expiration time (t_{exp}), and requests that the shared key between herself and Bob be valid till that time.

2. The KDC replies back with a shared key k_{AB}. It also includes Bob's identity in the message. The message is encrypted with the key that Alice shares with the KDC ($K_{A,KDC}$). The message also has a timestamp t_s.

3. The KDC also sends a message to Bob that has the shared key k_{AB}, and also Alice's identity in it. The message is encrypted with the key that Bob shares with the KDC ($K_{B,KDC}$). The message also has a timestamp t_s.

   Now Alice and Bob have a shared key that they can use to communicate with each other. If the expiration time is not included in the messages, then a malicious entity Eve can replay the message sent to Bob by the KDC, and trick Bob into communicating with her if she somehow knew the shared key between Alice and Bob.

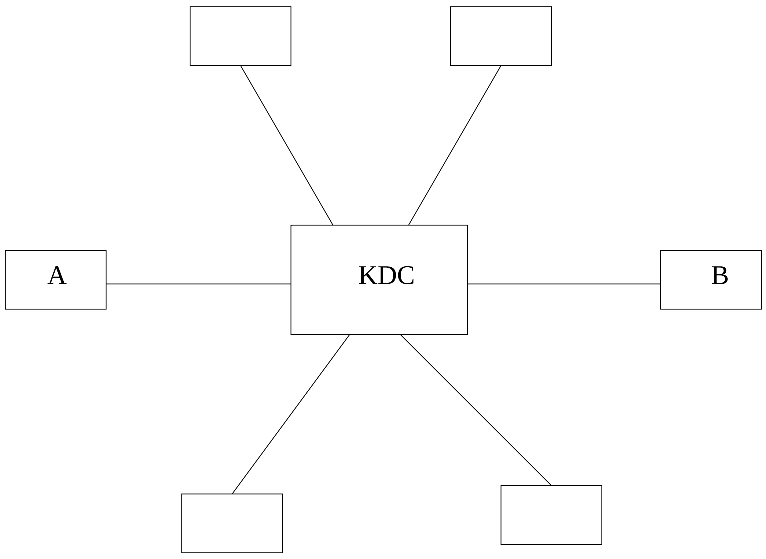

Figure 1: Key Distribution Center

4. Alice now starts communicating with Bob. To make sure she is communicating with Bob, she sends a nonce N_A to Bob that is encrypted with k_{AB}.

5. Bob sends back (N_A+1) and another nonce N_B to Alice. The message is encrypted with k_{AB}. Since Bob is the only entity that knows k_{AB}, the fact that Alice got back (N_A+1) in the message means that the message came from Bob, since Bob is the only entity that could have decrypted Alice's message and could have extracted N_A out of it. Bob sends another nonce N_B in his message to authenticate Alice.

6. Alice sends back (N_B+1) to Bob. The message is encrypted with k_{AB}. Now Alice and Bob know that they are communicating with the right entities (each has authenticated himself/herself to the other).

The Needham-Schroeder protocol is based on the above idea of a KDC.

2 Kerberos

As shown in the figure, the following steps are involved in the Kerberos protocol.

1. Alice sends a message to the KDC requesting for a session key to communicate with Bob. The message has an expiration time t_{exp} and a nonce N.

2. The KDC replies back with a ticket which consists of a session key k_{AB}, the IP address of Alice, a timestamp t_s, the expiration time t_{exp}, and the nonce N. The message is encrypted with the shared key between Alice and KDC,

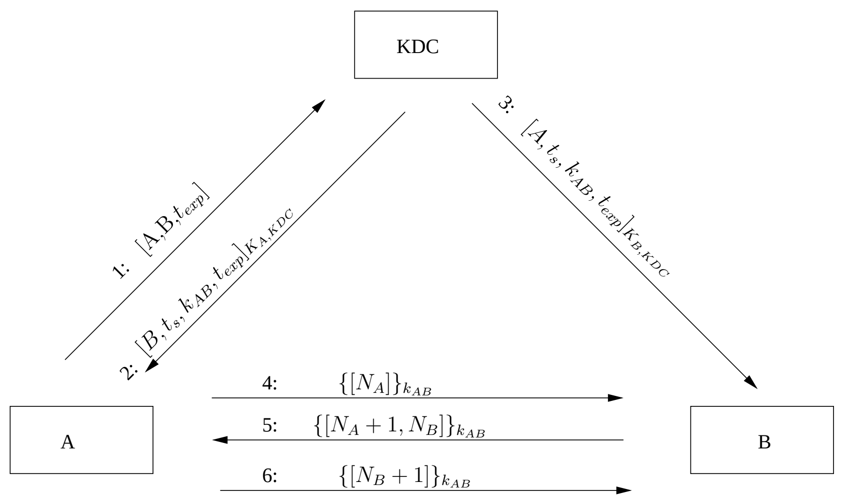
KDC
1: [A,B,t_{exp}]
2: [B, t_s, k_{AB}, t_{exp}]$_{K_{A,KDC}}$
3: [A, t_s, k_{AB}, t_{exp}]$_{K_{B,KDC}}$
4: $\{[N_A]\}_{k_{AB}}$
5: $\{[N_A + 1, N_B]\}_{k_{AB}}$
6: $\{[N_B + 1]\}_{k_{AB}}$
A
B

Figure 2: Needham-Schroeder Protocol

$K_{A,KDC}$. The KDC also encrypts the ticket with the key that Bob shares with it ($K_{B,KDC}$), and sends it to Alice.

3. Alice decrypts the message that is encrypted with $K_{A,KDC}$, and extracts the session key k_{AB} out of it. She then sends a message to Bob that has her identity, Bob's identity, and a timestamp t_s, and is encrypted with k_{AB}. She also sends the ticket that she received from the KDC, and which was encrypted with $K_{B,KDC}$.

4. Bob decrypts the ticket, and extracts the session key k_{AB} out of it. It then sends a message back to Alice that has his and Alice's identities, and the timestamp t_s with 1 added to it. The message is encrypted with k_{AB}.

Below are some attacks that are possible if we leave out some of the components in the messages above.

1. If we leave out Bob's identity in the messages from Alice, then a man-in-the-middle attack is possible. The man-in-the-middle intercepts the first message from Alice, and replaces Bob with Bob'. The KDC replies back with a ticket that is encryped with the shared key between itself and Bob'. The man-in-the-middle then intercepts the message from Alice to Bob, and forwards it to Bob', and forwards the response from Bob' to Alice.

2. A man-in-the-middle might also flip bits in message 1 (as shown in Figure 3) to make B as B', or flip bits in message 3 ((as shown in Figure 3)) in the ticket encrypted with $K_{B,KDC}$ to make A as A'.

We now discuss the design or system issues with Kerberos below.

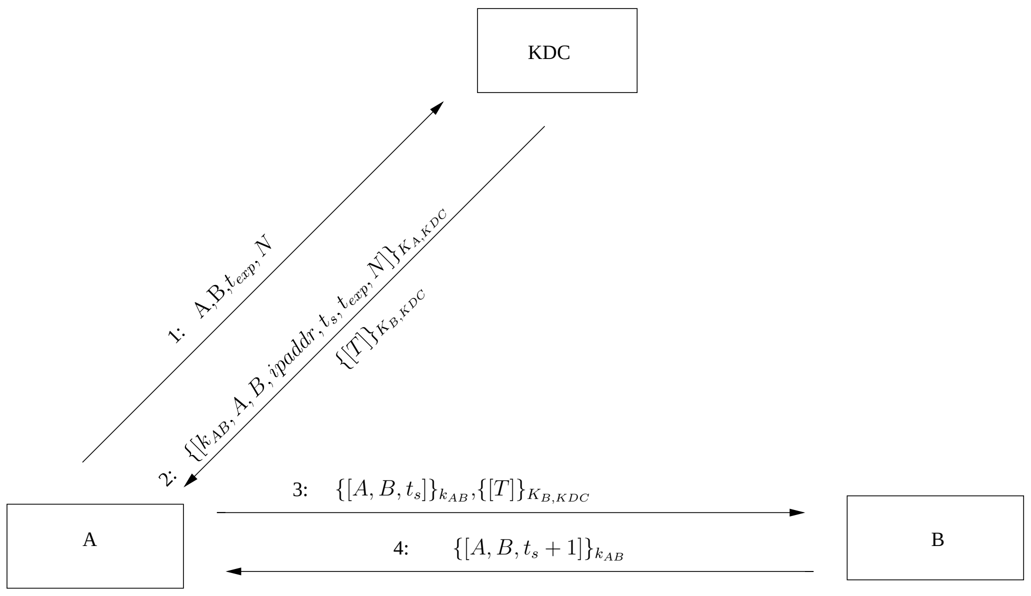
KDC
1: A,B,t_{exp}, N
2: $\{[k_{AB}, A, B, ipaddr, t_s, t_{exp}, N]\}_{K_{A,KDC}}$
$\{[T]\}_{K_{B,KDC}}$
A
3: $\{[A, B, t_s]\}_{k_{AB}}$,$\{[T]\}_{K_{B,KDC}}$
4: $\{[A, B, t_s + 1]\}_{k_{AB}}$
B

Figure 3: Kerberos

1. The TCB in this system consists of the KDC, and the time service. If the security of the KBC is compromised, then the whole system is compromised.

2. The KDC uses passwords to authenticate users, which is not good.

3. Revocation of a key is easy; to remove a user from the system, just remove the information of its key from the KDC.

Implementation issues of Kerberos are discussed below.

1. No MAC or checksum is included with the messages.

2. No IV (or Initialization Vector) is used for encryption.

3. srand(time(NULL)) and rand() are used in implementations. One can guess the time on a system, and hence guess the series of random numbers that were used in an instantiation of the protocol.

4. In older days, since encryption was very CPU-intensive, Kerberos had only authentication by default, and encryption was kept optional.