Computer Networks (CS425)

Instructor: Dr. Dheeraj Sanghi

Prev | Next | Index

Network Security(Contd...)

Key Distribution Centre(Recap.)

There is a central trusted node called the Key Distribution Center (KDC). Every node has a key which is shared between it and the KDC. Since no one else knows node A's secret key K_A, KDC is sure that the message it received has come from A. When A wants to communicate with B it could do two things:

1. A sends a message encrypted in it's key K_A to the KDC. The KDC then sends a common key K_S to both A and B encrypted in their respective keys K_A and K_B. A and B can communicate safely using this key.
2. Otherwise A sends a key K_S to KDC saying that it wants to talk to B encrypted in the key K_A. KDC send a message to B saying that A wants to communicate with you using K_S.

There is a problem with this implementation. It is prone to **replay attack**. The messages are in encrypted form and hence would not make sense to an intruder but they may be replayed to the listener again and again with the listener believing that the messages are from the correct source. When A send a message $K_A(M)$, C can send the same message to B by using the IP address of A. A solution to be used is to use the key only once. If B sends the first message $K_A(A,K_S)$ also along with K(s,M), then again we may have trouble. In case this happens, B should accept packets only with higher sequence numbers.

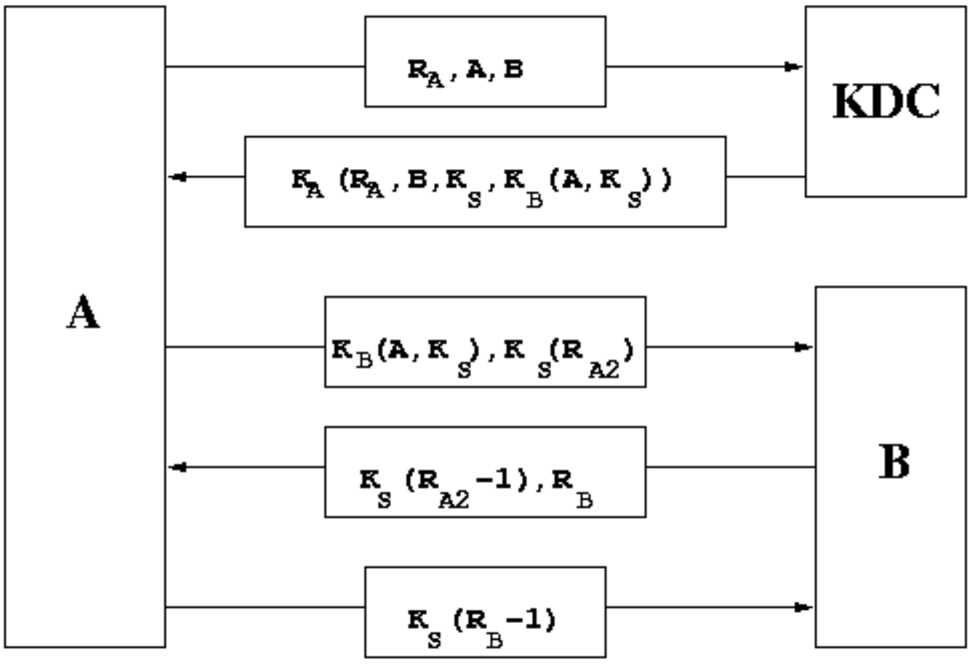
To prevent this, we can use:

- **Timestamps** which however don't generally work because of the offset in time between machines. Synchronization over the network becomes a problem.
- **Nonce numbers** which are like ticket numbers. B accepts a message only if it has not seen this nonce number before.

In general, 2-way handshakes are always prone to attacks. So we now look at an another protocol.

Needham-Schroeder Authentication Protocol

This is like a bug-fix to the KDC scheme to eliminate replay attacks. A 3-way handshake (using nonce numbers) very similar to the ubiquitous TCP 3-way handshake is used between communicating parties. A sends a random number R_A to KDC. KDC send back a ticket to A which has the common key to be used.



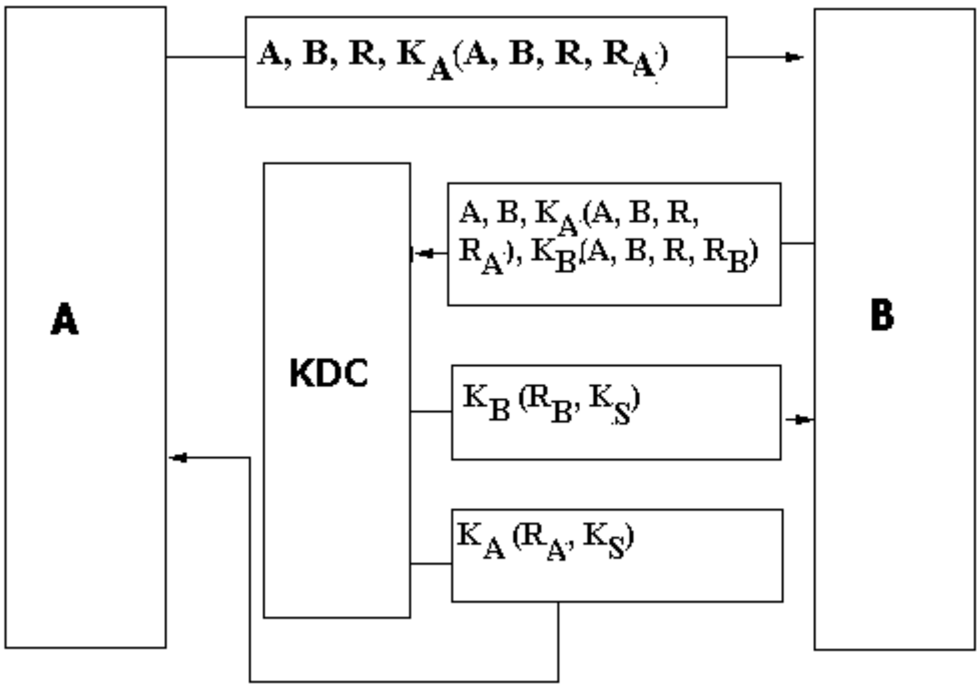
The Needham-Schroeder Authentication Protocol

R_A, R_B and R_{A2} are nonce numbers. R_A is used by A to communicate with the KDC. On getting the appropriate reply from the KDC, A starts communicating with B, whence another nonce number R_{A2} is used. The first three messages tell B that the message has come from KDC and it has authenticated A. The second last message authenticates B. The reply from B contains R_B, which is a nonce number generated by B. The last message authenticates A. The last two messages also remove the possibility of replay attack.

However, the problem with this scheme is that if somehow an intruder gets to know the key K_S (maybe a year later), then he can replay the entire thing (provided he had stored the packets). One possible solution can be that the ticket contains a time stamp. We could also put a condition that A and B should change the key every month or so. To improve upon the protocol, B should also involve KDC for authentication. We look at one possible improvement here. which is a different protocol.

Otway-Rees Key Exchange Protocol

Here a connection is initiated first. This is followed by key generation. This ensures greater security. B sends the message sent by A to the KDC and the KDC verifies that A, B, R in the two messages are same and R_A and R_B have not been used for some time now. It then sends a common key to both A and B.


A, B, R, K_A(A, B, R, R_A)
A
KDC
B
A, B, K_A(A, B, R, R_A), K_B(A, B, R, R_B)
K_B (R_B, K_S)
K_A (R_A, K_S)

THE OTWAY-REES KEY EXCHANGE PROTOCOL

In real life all protocols will have time-stamps. This is because we cannot remember all random numbers generated in the past. We ignore packets with higher time stamps than some limit. So we only need to remember nonces for this limit. Looking at these protocols, we can say that designing of protocols is more of an art than science. If there is so much problem in agreeing on a key then should we not use the same key for a long time. The key can be manually typed using a telephone or sent through some other media.

Challenge - Response Protocol

Suppose nodes A and B have a shared key K_{AB} which was somehow pre-decided between them. Can we have a secure communication between A and B ? We must have some kind of a three way handshake to avoid replay attack So, we need to have some interaction before we start sending the data. A *challenges* B by sending it a random number R_A and expects an encrypted reply using the pre-decided key K_{AB}. B then *challenge*s A by sending it a random number R_B and expects an encrypted reply using the pre-decided key K_{AB}.

A B

1. A, R_A------------>
2. <--------$K_{AB}(R_A)$, R_B
3. $K_{AB}(R_B)$--------->

Unfortunately this scheme is so simple that this will not work. This protocol works on the assumption that there is a unique connection between A and B. If multiple connections are possible, then this protocol fails. In replay attack, we could repeat the message $K_{AB}(M)$ if we can somehow convince B that I am A. Here, a node C need not know the shared key to communicate with B. To identify itself as A, C just needs to send $K_{AB}(R_{B1})$ as the response to the challenge-value R_{B1} given by B in the first connection. C can remarkably get this value through the second connection by asking B itself to provide the response to its own challenge. Thus, C can verify itself and start communicating freely with B.
Thus, replay of messages becomes possible using the second connection. Any encryption desired, can be obtained by sending the value as R_{B2} in the second connection, and obtaining its encrypted value from B itself.

A B

1st Connection: A, R_A------------>

<----------$K_{AB}(R_A)$, R_{B1}

2nd Connection: A, R_{B1}----------->

<--------- $K_{AB}(R_{B1})$, R_{B2}

1st Connection: $K_{AB}(R_{B1})$--------->

Can we have a simple solution apart from time-stamp ? We could send $K_{AB}(R_A, R_B)$ in the second message instead of $K_{AB}(R_A)$ and R_A. It may help if we keep two different keys for different directions. So we share two keys one from A to B and the other from B to A. If we use only one key, then we could use different number spaces (like even and odd) for the two directions. Then A would not be able to send R_B. So basically we are trying to look at the traffic in two directions as two different traffics. This particular type of attack is called **reflection attack**.

5 - way handshake

We should tell the sender that the person who initiates the connection should authenticate himself first. So we look at another protocol. Here we are using a 5-way handshake but it is secure. When we combine the messages, then we are changing the order of authentication which is leading to problems. Basically $K_{AB}(R_B)$ should be sent before $K_{AB}(R_A)$. If we have a node C in the middle, then C can pose as B and talk to A. So C can do replay attack by sending messages which it had started some time ago.

A B

1. A------------------>
2. <-----------------R_B
3. $K_{AB}(R_B)$---------->
4. R_A---------------->
5. <----------$K_{AB}(R_A)$

Fig: 5-way handshake in Challenge-Response Protocol

On initiating a connection B challenges A by sending it a random number R_B and expects an encrypted reply using the pre-decided key $K_{AB.}$ When A sends back $K_{AB}(R_B)$, B becomes sure that it is talking to the correct A, since only A knows the shared key. Now A challenges B by sending it a random number R_A, and expects an encrypted reply using the pre-decided key $K_{AB.}$ When B sends back $K_{AB}(R_A)$, A becomes sure that it is talking to the correct B, since only B knows the shared key.

However in this case also, if we have a node C in the middle, then C can pose as B and talk to A. So C can do replay attack by sending messages which it had stored some time ago !!

---

back to top
Prev| Next | Index

addressing. We could say that the network number is 21 bits (for 8 class C networks) or say that it is 24 bits and 7 numbers following that. For example : a.b.c.d / 21