Symmetric-key based Security Protocols

Simon Foley

October 20, 2015

Entity Authentication: process of identifying a claimed identity

Suppose that Bob is a fileserver that accepts incoming connections from other systems (workstations, etc). It authenticates these connections by simply believing any message that it receives:

Msg1 $A \rightarrow B$: I'm Alice

In this case, the claim "I'm Alice" might correspond to the source IP address of an attempted connection.
A protocol *run* corresponds to a single use of the protocol between principals. We distinguish different runs by α, β, etc. For example,

Msg.α1 $A \rightarrow B$: I'm Alice
Msg.β1 $D \rightarrow B$: I'm Dan

An attacker Eve can masquerade as another user:

Msg.γ1 $A[E] \rightarrow B$: I'm Alice

Entity Authentication: Challenge Response I

Bob and Alice share secret key K_{AB} not known by anybody else.

Bob authenticates Alice.

Msg1 $A \rightarrow B$: I'm Alice

Msg2 $B \rightarrow A$: R

Msg3 $A \rightarrow B$: $\{R\}_{K_{AB}}$

R is a *nonce*: a number used only once. The nonce provides a way for Bob to test the *freshness* of a message, that is, whether he has seen it before.

Remember that $\{...\}_{K_{AB}}$ denotes symmetric encryption using K_{AB} providing integrity and secrecy.

This is a *Statefull* protocol since Bob needs to keep track of the challenges issued and responses between steps 2 and 3.

Entity Authentication: Challenge Response I

Different runs of the protocol may interleave in different ways.

Msg.$\alpha 1$	$A \rightarrow B:$	I'm Alice
Msg.$\alpha 2$	$B \rightarrow A:$	R_1
Msg.$\beta 1$	$D \rightarrow B:$	I'm Dan
Msg.$\beta 2$	$B \rightarrow D:$	R_2
Msg.$\alpha 3$	$A \rightarrow B:$	$\{R_1\}_{K_{AB}}$
Msg.$\beta 3$	$D \rightarrow B:$	$\{R_2\}_{K_{DB}}$

Bob must keep track of the challenges (nonces) issued and the principals from which the response is pending.

The purpose of the nonce is to allow Bob to test the freshness of the message (has never seen a given response before).

Bob needs to be careful how he implements nonce-generation so that they really are numbers used only once; he should not have to remember nonces across different protocol runs.

Entity Authentication: Time Based

Use timestamp to support testing of message freshness.

$$\text{Msg1} \quad A \to B: \quad \{\text{I'm Alice}, T_A\}_{K_{AB}}$$

Bob tests that the time T_A is within the current time window.

This is a *stateless* protocol since nothing about the run needs to be recorded between message exchanges.

May be vulnerable to attack if the message is replayed within the current time window. Possible defense: Alice includes a nonce in her message and Bob keeps track of nonces presented within current time window.

$$\text{Msg1} \quad A \to B: \quad N, \{\text{I'm Alice}, N, T_A\}_{K_{AB}}$$

However, attack is still possible—more on this later!

Mutual Authentication (MA)

Alice Authenticates Bob and Bob authenticates Alice.

Msg1	$A \rightarrow B:$	I'm Alice
Msg2	$B \rightarrow A:$	R_1
Msg3	$A \rightarrow B:$	$\{R_1\}_{K_{AB}}$
Msg4	$A \rightarrow B:$	R_2
Msg5	$B \rightarrow A:$	$\{R_2\}_{K_{AB}}$

Reducing protocol to three message exchanges makes it more efficient.

Msg1	$A \rightarrow B:$	I'm Alice, R_2
Msg2	$B \rightarrow A:$	$R_1, \{R_2\}_{K_{AB}}$
Msg3	$A \rightarrow B:$	$\{R_1\}_{K_{AB}}$

Mutual Authentication: Reflection Attack

Consider the shorter (more efficient) MA protocol. Eve pretending to be Alice, starts a run of the protocol as (note principal notation):

$$\begin{aligned} \text{Msg}\alpha.1 \quad & A[E] \rightarrow B: \quad \text{I'm Alice}, R_2 \\ \text{Msg}\alpha.2 \quad & B \rightarrow A[E]: \quad R_1, \{R_2\}_{K_{AB}} \end{aligned}$$

Eve cannot continue this run, but starts another run with Bob.

$$\begin{aligned} \text{Msg}\beta.1 \quad & A[E] \rightarrow B: \quad \text{I'm Alice}, R_1 \\ \text{Msg}\beta.2 \quad & B \rightarrow A[E]: \quad R_3, \{R_1\}_{K_{AB}} \end{aligned}$$

Eve cannot continue this run, but has tricked Bob into acting as an 'oracle' and providing $\{R_1\}_{K_{AB}}$ and can now continue α run:

$$\text{Msg}\alpha.3 \quad A[E] \rightarrow B: \quad \{R_1\}_{K_{AB}}$$

Bob now thinks he is communicating with Alice!

Avoiding a Reflection Attack

General Principal: *do not have Alice and Bob do exactly the same thing!*

Alice and Bob use different keys to authenticate each other. For example, Alice uses K_{AB} to authenticate Bob and Bob uses $K_{AB} + 1$ to authenticate Alice.

Insist that the challenge from initiator (Alice) looks different (in a testable way) from the challenge from responder (Bob). For example, initiator challenges are odd numbers and responder challenges are even numbers.

It would seem that the reflection attack cannot be done on the first (longer) MA protocol since the initiator(Alice) must be the first to prove its identity.

Exercise: is this really the case?

Symmetric-Key Mutual Authentication Protocol

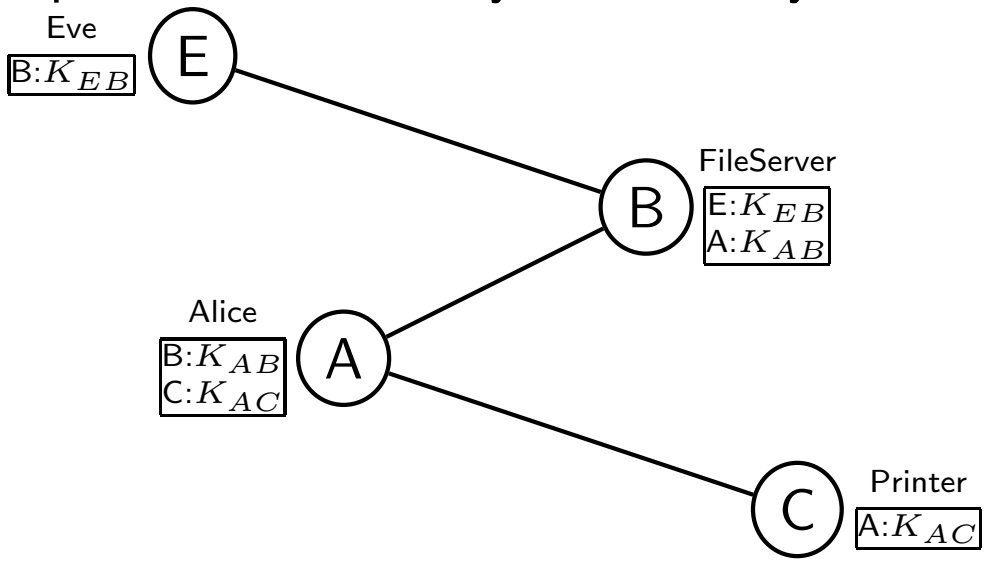
ISO/IEC 9798-2 three-pass version of protocol avoids a reflection attack.

$$\begin{aligned} \text{Msg1}: &\quad A \rightarrow B \quad A, N_a \\ \text{Msg2}: &\quad B \rightarrow A \quad \{N_a, N_b, A\}_{K_{ab}} \\ \text{Msg3}: &\quad A \rightarrow B \quad \{N_a, N_b\}_{K_{ab}} \end{aligned}$$

Secure Key Distribution Protocols

Secure Connections

Networked principals share secret symmetric keys.



Principals only accept messages encrypted with keys they recognize.

$$A \rightarrow C \quad : \quad \{print : document, \cdots\}_{K_{AC}}$$

Goal is to provide integrity and confidentiality of data and *data origin authentication* whereby principal can determine origin of message.

Deployment Challenges

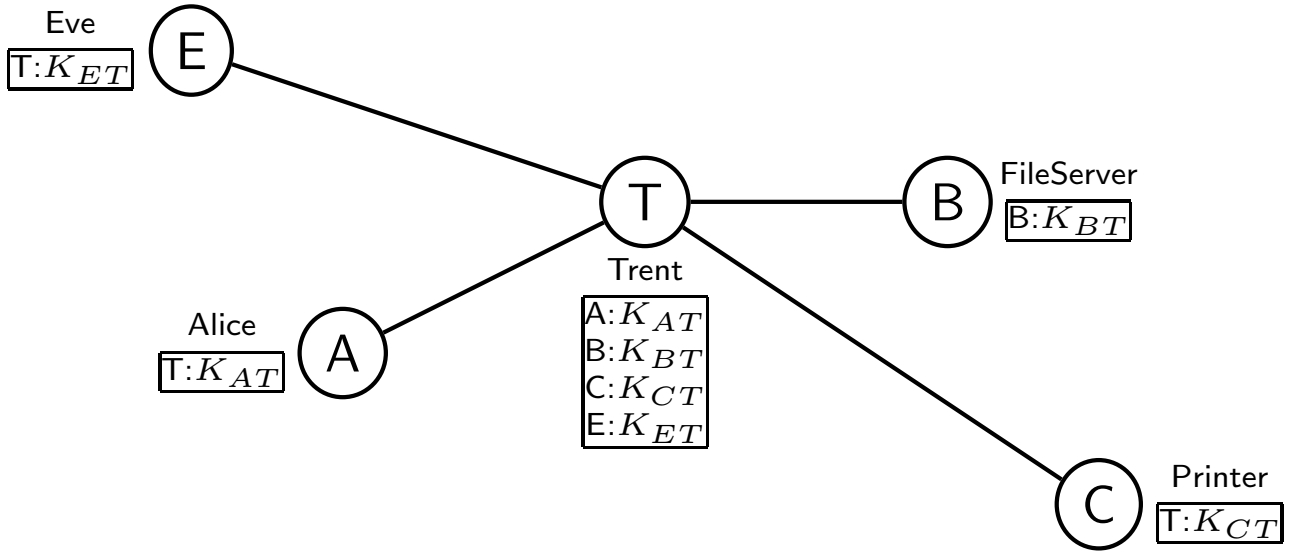
The previous example assumes that keys have already been deployed across network.

- Secret keys provide the basis for authentication, secrecy and integrity: need different key for each principal relationship.
- Given n principals, then can have up to n^2 keys to deploy.
- How is one principal 'introduced' to another if they do not already share a key?
- What should be done when a shared key is compromised?
- How to deal with principals that are not connected/available?

Any solution should address these issues and be scalable.

Key Distribution Center (Wide Mouth Frog)

Strategy: rely on a trusted third party to perform principal introductions.



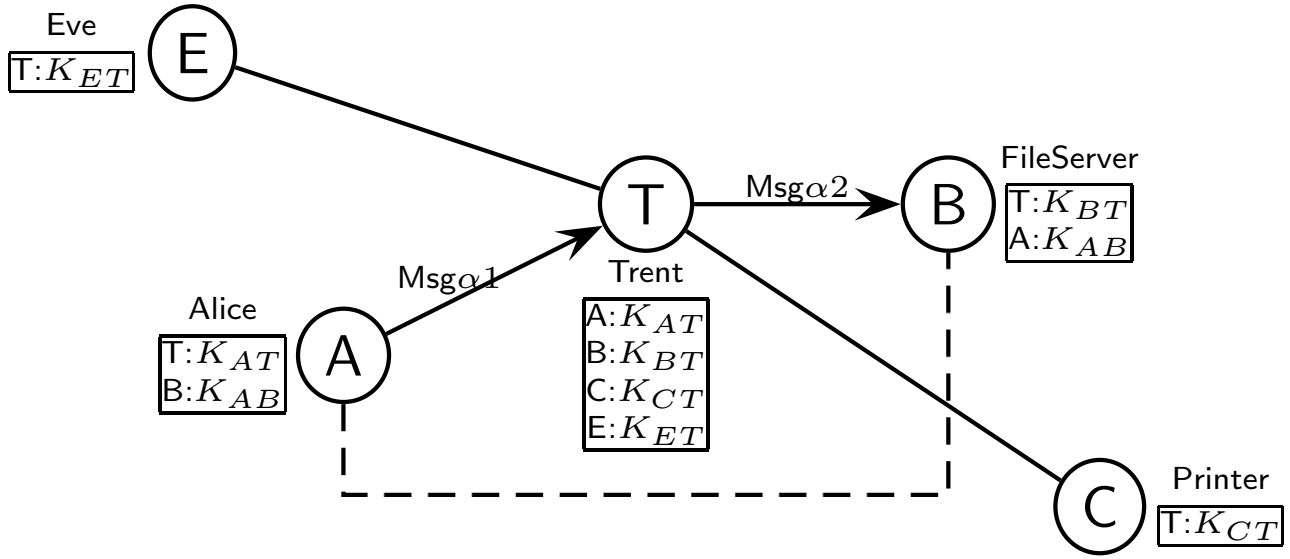
Alice uses the following security protocol in order to be introduced to Bob and to establish a shared key between herself and Bob.

$$\begin{array}{lll} \text{Msg1} & A \to T: & (B, \{K_{AB}\}_{K_{AT}}) \\ \text{Msg2} & T \to B: & (A, \{K_{AB}\}_{K_{BT}}) \end{array}$$

This protocol provides both authentication and key exchange and is known as the *wide mouth frog protocol*.

KDC Example (Wide Mouth Frog)

Alice sets up a key with Bob.



Represented by a run of the protocol:

$$\begin{aligned} \text{Msg.}\alpha 1 \quad & A \rightarrow T: \quad (B, \{K_{AB}\}_{K_{AT}}) \\ \text{Msg.}\alpha 2 \quad & T \rightarrow B: \quad (A, \{K_{AB}\}_{K_{BT}}) \end{aligned}$$

Can be used by any pair of principals that share keys with Trent.

KDC Notes

- A security protocol is often referred to as a *key exchange protocol* or *authentication protocol*.
- The keys shared between Trent and other principals are called *long term keys*. For example, they could correspond to a (one-way hash of) user's password.
- The keys that are exchanged as a result of the protocol are called *session keys*, or *short-term keys*.
- Trent is a *trusted third party* that is trusted (by principals) to perform authentic key exchange.

Wide Mouth Frog: Replay Attack

Trent offers this trusted introduction service to all principals and therefore shares a secret key K_{ET} with principal Eve.
Eve listens in on message exchanges between Alice, Bob and Trent.

$$\begin{array}{lll} \text{Msg}\alpha.1 & A \to T: & (B, \{K_{AB}\}_{K_{AT}}) \\ \text{Msg}\alpha.2 & T \to B: & (A, \{K_{AB}\}_{K_{BT}}) \end{array}$$

Eve picks up a copy of $(B, \{K_{AB}\}_{K_{AT}})$, modifies it to $(E, \{K_{AB}\}_{K_{AT}})$ and replays it to Trent, imitating Alice.

$$\begin{array}{lll} \text{Msg}\beta.1 & A[E] \to T: & (E, \{K_{AB}\}_{K_{AT}}) \\ \text{Msg}\beta.2 & T \to E: & (A, \{K_{AB}\}_{K_{ET}}) \end{array}$$

We must preserve the integrity of principal's name in message exchange.
Revised Protocol

$$\begin{array}{lll} \text{Msg1} & A \to T: & (\{B, K_{AB}\}_{K_{AT}}) \\ \text{Msg2} & T \to B: & (\{A, K_{AB}\}_{K_{BT}}) \end{array}$$

Wide Mouth Frog: Key Revocation

Suppose that Alice is no longer trusted: perhaps K_{AT} has been compromised by a malicious principal and Alice needs to replace her old key K_{AT} with a new key K'_{AT}.

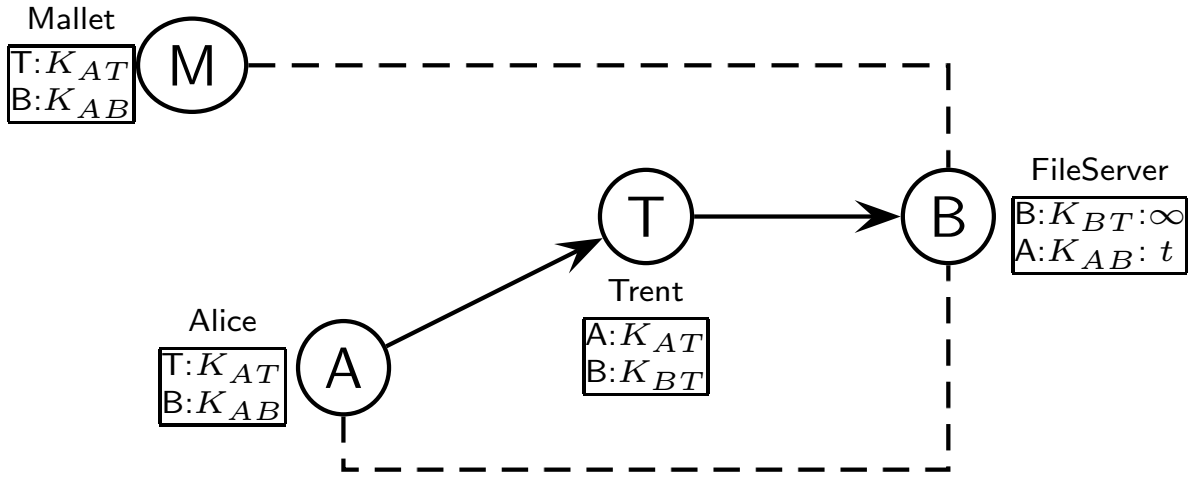
Trent needs to be informed not to accept requests from K_{AT}. We must also deal with the session keys that Alice currently holds since they could be compromised as the attacker may have copies of past key exchange requests encrypted by K_{AT}.

Possible Strategies:

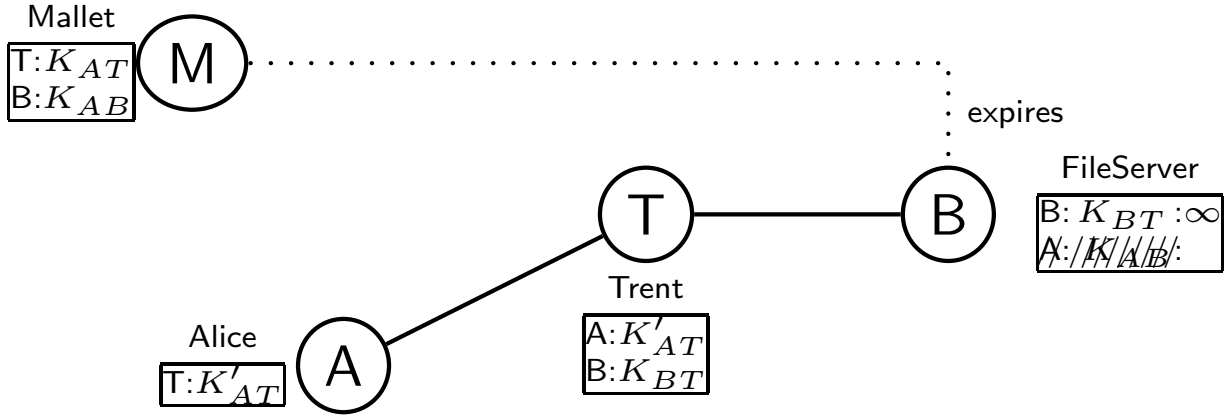
- Every principal must be notified and asked to remove keys shared with Alice. This is not practical.
- More typical strategy is that *session keys*, such as K_{AB} have a limited lifetime and expire after a short time. If Alice is no longer trusted then revocation amounts to Trent removing K_{AT} from his database of keys. Now K_{AT} may no longer request, or get, new session keys. Any existing keys in her possession will soon expire (principals expected to discard expired keys).

Revocation: Example

Mallet obtains a copy of Alice's key K_{AT}.



Alice rekeys



Wide Mouth Frog: Replay Attack 2

Mallet keeps copies of past traffic between Alice, Bob and Trent.

$$\text{Msg.}\alpha 1 \quad A \to T: \quad (\{B, K_{AB}\}_{K_{AT}})$$
$$\text{Msg.}\alpha 2 \quad T \to B: \quad (\{A, K_{AB}\}_{K_{BT}})$$

At a later stage Mallet compromises Alice's key. Alice rekeys to long term key K'_{AT} and the session key K_{AB} expires.

Mallet uses stolen key K_{AT} to decrypt Msg.$\alpha 1$ to obtain K_{AB}, and replay

$$\text{Msg.}\beta 2 \quad T[M] \to B: \quad (\{A, K_{AB}\}_{K_{BT}})$$

Bob thinks this is a legitimate message from trent stating that session key K_{AB} can be used to speak with Alice.

Wide Mouth Frog Redesigned

To prevent this attack each principal must test the message for *freshness*: have we ever seen this message before? It is not realistic to store a history of every message received by a principal. Instead, we timestamp messages.

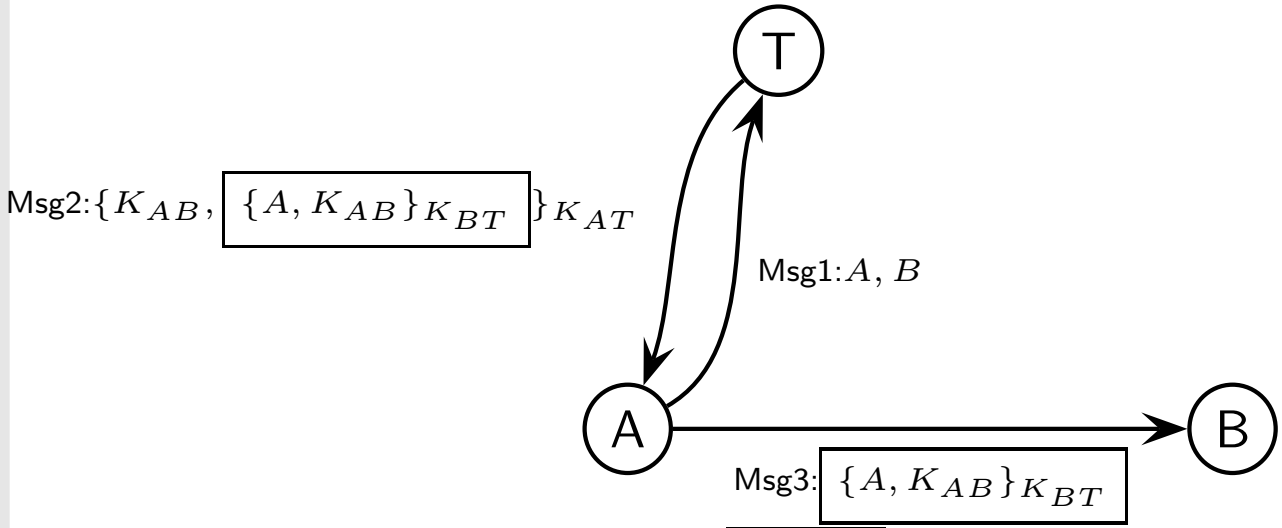
$$\begin{array}{lll} \text{Msg1} & A \to T: & (\{T_A, B, K_{AB}\}_{K_{AT}}) \\ \text{Msg2} & T \to B: & (\{T_T, A, K_{AB}\}_{K_{BT}}) \end{array}$$

T_A, T_T = timestamps provided by Alice and Trent, with a time window within which principals are prepared to accept a key as fresh. Potential problems:

- Time must be synchronised across network
- Alice must be trusted to generate a sensible key
- Not practical: too much work for Trent
- Can you find another attack on this protocol?

Improved Strategy for Protocol Operation (outline)

Trent issues *tickets* permitting principals communicate securely.



- T issues session key and ticket to protocol *initiator* A.
- A presents ticket $\{A, K_{AB}\}_{K_{BT}}$ to B, allowing A and B to communicate securely.
- Trent trusted to know the identity of principals/generate suitable key.
- Since trent only responds to initiator, near-stateless implementation.

(Symmetric) Needham-Schroeder Protocol (1978)

$$
\begin{array}{lll}
\text{Msg1} & A \to T: & A, B, N_A \\
\text{Msg2} & T \to A: & \{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BT}}\}_{K_{AT}} \\
\text{Msg3} & A \to B: & \{K_{AB}, A\}_{K_{BT}}, \{N'_A\}_{K_{AB}} \\
\text{Msg4} & B \to A: & \{N'_A - 1, N_B\}_{K_{AB}} \\
\text{Msg5} & A \to B: & \{N_B - 1\}_{K_{AB}}
\end{array}
$$

Nonce N_A assures Alice that she is talking to Trent (Key Distribution Centre KDC) and that its not a replay from an earlier protocol run.

Msg4, Msg5: Mutual authentication between Alice and Bob.

Nonce's N_A, N'_A and N_B fix messages to a run of the protocol.

Timestamp could be used instead of N_A if we can believe that clocks are synchronized.

Needham-Schroeder Protocol Attack I

Nonces are used for authentication and freshness; integrity of principle name protected by encryption. However, there is no constraint on the time between steps 2 and 3. Consider partial protocol run

$$\begin{array}{lll} \text{Msg}\alpha 1 & A \rightarrow T: & A, B, N_A \\ \text{Msg}\alpha 2 & T \rightarrow A: & \{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BT}}\}_{K_{AT}} \end{array}$$

and then Alice holds onto key K_{AB} and its ticket. When the Alice's key K_{AT} is revoked she can continue on with the protocol run, even though K_{AT} is no longer valid!

If A's secret key is compromised by an intruder, then the intruder can use the key to talk to any other principal and continue to use the session keys even after A's key has been revoked.

Needham-Schroeder Protocol Attack II

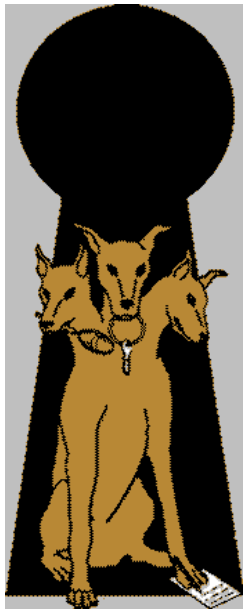
Suppose that attacker M manages to obtain a copy of the session key K_{AB}, keeps a copy of the ticket $\{K_{AB}, A\}_{K_{BT}}$, and then at a later date (M) replays the message.

$$\text{Msg}\beta 3 \quad A[M] \rightarrow B : \{K_{AB}, A\}_{K_{BT}}, \{N\}_{K_{AB}}$$

Then, it re-establishes a connection to B, but masquerading as A.

The attacker can then replay any messages encrypted under K_{AB} and B believes that they come from A.

Kerberos: A Protocol for Secure Distributed Systems



A network authentication protocol providing strong authentication for client/server applications.

Developed at MIT as part of 'Project Athena': a client server 'network' of Unix running on early PCs.

Kerberos provides for authentication, integrity and secrecy.

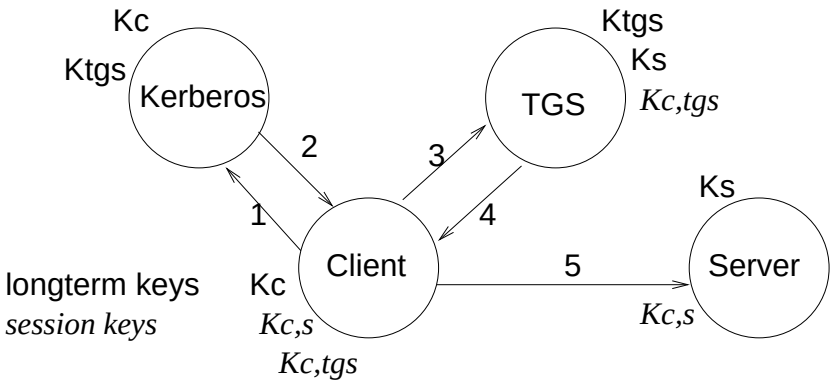
A Needham-Schroeder style protocol.

Supported by many systems/variations:

- *nix, Mac OS X, Windows (2000 and later), NFS, X-Windows, ...
- Eudora, Apache, Oracle RDBMS, JAAS, ...,
- PAM, GSS-API, RCP, SOCKS, ...

IETF concluded WG:
`https://datatracker.ietf.org/wg/krb-wg/documents/`

Kerberos Protocol Outline



1. Request for TGS Ticket
2. TGS Ticket
3. Request for Service Ticket
4. Service Ticket
5. Request for Service.

All communication via secure channels.

Kerberos Server (Authentication Server) authenticates clients

Ticket Granting Server (TGS) grants tickets to clients to use services.

Service (Server) available to those who can present valid tickets.

Ticket granting ticket $T_{c,tgs} = \{C, timeperiod, K_{c,tgs}\}_{K_{tgs}}$
Ticket for server: $T_{c,s} = \{C, timeperiod, K_{c,s}\}_{K_S}$

Kerberos: Client Authentication - Getting Initial Ticket

A user wishes to login to a client workstation. Client's long-term key is derived from a hash of the user's password: $K_c = h(password)$. Kerberos Server also knows user passwords.

Client C requests a ticket for desired TGS from Kerberos Server.

$$\begin{array}{lll} \text{Msg1} & C \rightarrow K: & C, tgs, N_c \\ \text{Msg2} & K \rightarrow C: & \{tgs, K_{c,tgs}, T_{c,tgs}, N_c, timeperiod\}_{K_C} \end{array}$$

If access permitted (Kerberos decides) then ticket is granted and session-key $K_{c,tgs}$ allows client to communicate with tgs. Client is trusted to throw password away once authenticated. (important)

If incorrect password provided then ticket and session key cannot be extracted (no channel)—user cannot login.

Kerberos (authentication) Server hosted on a hardened system.

Kerberos: Requesting a Ticket for a Service

Client with ticket $T_{c,tgs}$ for TGS, requests ticket for a server.

Msg3 $C \rightarrow TGS: \quad C, S, N'_c, T_{c,tgs}, \{C, address, time\}_{K_{c,tgs}}$

Msg4 $TGS \rightarrow C: \quad \{S, K_{c,s}, T_{c,s}, N'_c, timeperiod\}_{K_{c,tgs}}$

Authenticator $\{C, address, time\}_{K_{c,tgs}}$ is used by TGS to verify that client knows session key $K_{c,tgs}$.

Mutual Authentication

Client and Server establish a secure channel based on session key $K_{c,s}$.

Msg5 $C \rightarrow S: \quad \{C, address, time'\}_{K_{c,s}}, T_{c,s}$

Msg6 $S \rightarrow C: \quad \{S, time'\}_{K_{c,s}}$

Kerberos Discussion

Every service must be individually modified (*kerberized*) for use with kerberos.

Don't forget that Kerberos does not secure the OS: client/servers rely on the operating system security (and administrator) to protect the keys of different users.

Kerberos requires a secured (hardened) and continuously available Kerberos authentication Server

Kerberos stores all passwords encrypted with a single key

Kerberos does not protect against modifications to system software

Provides basis for *single-signon* based authentication.

A protocol with a different 'shape' ISO/SEC 9798-2

ISO/SEC 9798-2 Symmetric Key 5 Pass Mutual Authentication Protocol is useful when the initiator does not have access to the trusted third party.

$$
\begin{array}{lll}
\text{Msg1}: & A \rightarrow B & : A, N_a \\
\text{Msg2}: & B \rightarrow S & : A, B, N_a, N_b \\
\text{Msg3}: & S \rightarrow B & : \{N_a, K_{ab}, B\}_{K_{as}}, \{N_b, K_{ab}, A\}_{K_{bs}} \\
\text{Msg4}: & B \rightarrow A & : \{N_a, K_{ab}, B\}_{K_{as}}, \{N_a, N'_b\}_{K_{ab}} \\
\text{Msg5}: & A \rightarrow B & : \{N'_b, N_a\}_{K_{ab}}
\end{array}
$$

When do you think this protocol would be useful?