Chapter 8

Protocols

Sometimes the protocol we follow can be crucial to the security of a system. We will look at systems in which the protocol plays a large part:

1. Kerberos protocol for distributing keys
2. Voting protocols
3. Contract signing protocols

These three protocols are by no means the only ones. There are many key distribution protocols, and also key transfer protocols such as those used in Clipper key escrow. There are protocols for oblivious transfer, in which two parties can complete a joint computation, without either party revealing any unnecessary data. For example Alice has two secret strings, only one of which she can disclose to Bob, using Bob's choice of secret. In addition, Bob may not want Alice to learn which secret he chose. This is an oblivious transfer problem.

8.1 Kerberos

Kerberos is a network ***authentication*** protocol. It is designed to provide strong authentication for client/server applications by using public key cryptography. Kerberos is freely available from MIT, in a similar way that they provide the X11 window system. MIT provides Kerberos in source form, so that anyone who wishes may look over the code for themselves and assure themselves that the code is trustworthy. Kerberos is also available in commercial products.

The Kerberos protocol uses strong cryptography so that a client can prove its identity to a server (and vice versa) across an insecure network connection. After a client and server have used Kerberos to prove their identity, they can also encrypt all of their communications to assure privacy and data integrity as they go about their business.

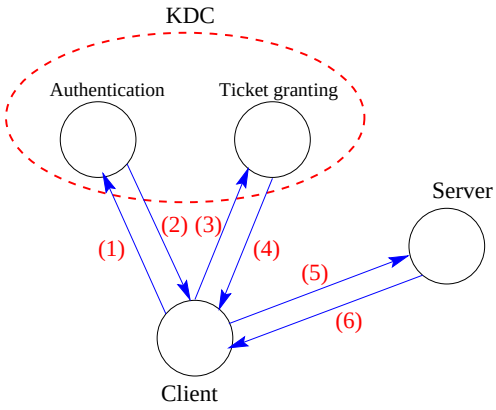

Figure 8.1: Kerberos components

In Figure 8.1, we see that when a client first authenticates to Kerberos, she:

1. talks to the Authentication Service on the KDC, to ...
2. get a *Ticket Granting Ticket* (encrypted with the client's password).
3. When the client wants to talk to a Kerberized service, she uses the *Ticket Granting Ticket* to talk to the *Ticket Granting Service* (which also runs on the KDC). The *Ticket Granting Service* verifies the client's identity using the *Ticket Granting Ticket* and ...
4. issues a ticket for the desired service.
5. (and so on) - the client may then use the ticket, to interact with the server.

The reason the Ticket Granting Ticket exists is so a user doesn't have to enter in their password every time they wish to connect to a Kerberized service or keep a copy of their password around. If the Ticket Granting Ticket is compromised, an attacker can only masquerade as a user until the ticket expires.

8.1.1 Kerberos protocol

Kerberos uses a variant of the Needham-Schroeder protocol described in [NS78]. There are two sorts of credentials used, ***tickets*** and ***authenticators***. A *ticket* $T_{c,s}$ contains the client's name and network address, the server's name, a timestamp and a session key. This is encrypted with the server's secret key (so that the client is unable to modify it). An *authenticator* $A_{c,s}$ contains the client's name, a timestamp and an optional extra session key. This is encrypted with the session key shared between the client and the server. A *key* $K_{x,y}$ is a session key shared by both x and y. When we encrypt a message M using the key $K_{x,y}$ we write it as $\{M\}K_{x,y}$. If Alice and Bob both share keys with a trustee (Ted), and Alice wants to get a session key for communication with Bob, we use the following sequence.

- Alice sends a message to Ted containing her own identity, Ted's TGS identity, and a one-time value (n) : $\{a, tgs, n\}$.
- Ted responds with a key encrypted with Alice's secret key (which Ted knows), and a ticket encrypted with the TGS secret key: $\{K_{a,tgs}, n\}K_a \;\; \{T_{a,tgs}\}K_{tgs}$.
  Alice now has an initial (encrypted) ticket, and a session key: ($\{T_{a,tgs}\}K_{tgs}$ and $K_{a,tgs}$).
- Alice can now prove her identity to the TGS, because she has a session key $K_{a,tgs}$, and the *Ticket Granting Ticket*: $\{T_{a,tgs}\}K_{tgs}$.

Later, Alice can ask the TGS for a specific service ticket:

- When Alice wants a ticket for a specific service (say with Bob), she sends an *authenticator* along with the *Ticket Granting Ticket* to the TGS: $\{A_{a,b}\}K_{a,tgs} \;\; \{T_{a,tgs}\}K_{tgs}\,, b, n$.
- The TGS responds with a suitable key and a ticket: $\{K_{a,b}, n\}K_{a,tgs} \;\; \{T_{a,b}\}K_b$.
- Alice can now use an authenticator and ticket directly with Bob: $\{A_{a,b}\}K_{a,b} \;\; \{T_{a,b}\}K_b$.

8.1.2 Weaknesses

Host security: Kerberos makes no provisions for host security; it assumes that it is running on *trusted* hosts with an *untrusted* network. If your host security is compromised, then Kerberos is compromised as well. If an attacker breaks into a multi-user machine and steals all of the tickets stored on that machine, he can impersonate the users who have tickets stored on that machine, but only until those tickets expire.

KDC compromises: Kerberos uses a principal's password (encryption key) as the fundamental proof of identity. If a user's Kerberos password is stolen by an attacker, then the attacker can impersonate that user with impunity. Since the KDC holds all of the passwords for all of the principals in a realm, if host security on the KDC is compromised, then the entire realm is compromised.

Salt: This is an additional input to the one-way hash algorithm. If a salt is supplied, it is concatenated to the plaintext password and the resulting string is converted using the one-way hash algorithm. In Kerberos 4, a salt was never used. The password was the only input to the one-way hash function. This has a serious disadvantage; if a user happens to use the same password in two Kerberos realms, a key compromise in one realm would result in a key compromise in the other realm.
In Kerberos 5 the complete principal name (including the realm) is used as the salt. This means that the same password will not result in the same encryption key in different realms or with two different principals in the same realm. The MIT Kerberos 5 KDC stores the key salt algorithm along with the principal name, and that is passed back to the client as part of the authentication exchange.

8.2 Voting protocols

A voting protocol is one in which independent systems vote in a kind of election, and afterwards we can check that the vote was correct. Each voter is only allowed a single vote, and the system should be corruption-proof.

A voting protocol is described in [DM83], using an example with Alice, Bob and Charles (!), who vote and then encrypt and sign a series of messages using public-key encryption. For example, if Alice votes v_A, then she will broadcast to all other voters the message

$$R_A(R_B(R_C(E_A(E_B(E_C(v_A))))))$$

where R_A is a random encoding function which adds a random string to a message before encrypting it with A's public key, and E_A is public key encryption with A's public key. Each voter then signs the message and decrypts one level of the encryption. At the end of the protocol, each voter has a complete signed audit trail and is ensured of the validity of the vote.

8.3 Contract signing

Signing contracts can be difficult. If one party signs the contract, the other may not, and so we have one party bound by the contract, and the other not. In addition, both may sign, and then one may say "*I didn't sign any contract!*" afterwards.

An oblivious transfer protocol is a notional protocol, which is central to some other protocols. In an oblivious transfer, randomness is used to convince participants of the fairness of some transaction, to any degree of certainty wanted (except 1). In the coin-tossing example, Alice knows the prime factors of a large number, and if Bob can factorize the number, then Bob wins the coin toss. A protocol (described in [DM83]) allows Alice to either divulge one of the prime factors to Bob, or not, with equal probability. Alice is unable to tell if she has divulged the factor, and so the coin toss is fair.

Oblivious transfer protocols may be used to construct contract-signing protocols in which

- Up to a certain point neither party is bound by the contract
- After that point both parties are bound by the contract
- Either party can prove that the other party signed the contract

Alice and Bob exchange signed messages, agreeing to be bound by a contract with ever-increasing probability (1%, 2%,...). In the event of early termination of the contract, either party can take the messages they have to an adjudicator, who chooses a random probability value (42% say) before looking at the messages. If both messages are over 42% then both parties are bound. If less then both parties are free.

8.4 Summary of topics

In this section, we introduced the following topics:

- Kerberos protocol
- Voting protocol
- Contract signing protocol

Supplemental questions for chapter 8

1. Investigate the voting protocol in [DM83]. List in order the messages received and sent by Bob.
2. The voting protocol in [DM83] has a serious drawback that precludes it from being used for (say) the Singapore election. What is this drawback?
3. Design a contract signing protocl, which uses a third party to oversee the contract.

Further study

- Textbook, section 10.
- DeMillo paper on *Protocols for Data Security* [DM83] at http://www.demillo.com/papers.htm.
