YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

CPSC 467a: Cryptography and Computer Security

Notes 24 (rev. 1)

Professor M. J. Fischer *December 1, 2005*

Lecture Notes 24

[The lecture consisted of a presentation of Zheng Ma's slides on the practical Kerberos, SSH, and SSL protocols. The notes below supplement the part on Kerberos.]

1 Kerberos[1]

Kerberos is a widely-used authentication system developed by M.I.T.'s Project Athena. It uses symmetric cryptography with a trusted server to enable secure authentication and key exchange between client nodes. The basic idea is that end-users use long-lived, memorized passwords to get short-lived session keys that aren't worth stealing from insecure desktop computers.

In slightly more detail, each client node has a secret *Kerberos key* that it uses to authenticate itself to the Kerberos server. This key is used only for encrypting session keys. Traffic from server to client is the encrypted using these short-lived session keys. The server stores a database of all client keys, so it must of course be kept highly secure. Ideally, the server is dedicated to this one purpose and does not permit user logins or run any other services.

When Alice wants to talk to Bob, she identifies herself to the server and requests to talk to Bob. The server generates a random session key k for Alice and Bob to use and encrypts it twice, once with Alice's Kerberos key and once with Bob's. It then sends back both encryptions to Alice. Alice decrypts her part and learns k. She sends the other part (called a *ticket*) to Bob, who then decrypts it and also learns k.

If one were only concerned with passive eavesdroppers, this would be enough. All traffic is encrypted with keys not known to Eve, so Eve learns nothing from eavesdropping, assuming of course that the underlying cryptosystem is secure. However, an active eavesdropper could successfully impersonate Alice to Bob by resending the ticket to him at a later time. Although Eve does not know k, she could also resend valid messages encrypted with k that Alice sent earlier and cause Bob to accept them as new messages. This could have serious consequences in a banking application, for example, where Bob is an ATM (automatic teller machine), and Alice's message is the instruction from the bank for the machine to dispense \$100.

The real Kerberos protocol is considerably more complicated than the above. First, it has features to protect against various kinds of attacks by active eavesdroppers. Second, it has additional features dictated by practical considerations. These additional features, which we will not discuss in much detail, have to do with making the protocol scalable to large networks and with allowing multiple Kerberos communities to coexist on the same network.

Two versions of Kerberos are in widespread use—versions 4 and 5. While we do not present either protocol in full detail, the simplified protocol we present in the next section relates to the newer version 5.

[1] I thank Don Davis for several helpful comments and corrections to a previous version of this section. Any remaining errors are entirely my own.

1.1 Simplified protocol

A simplified version of the Kerberos protocol is shown in Figure 1. In step 1, Alice requests a ticket from Trent (the trusted Kerberos server) to use in talking to Bob. Trent sends back the triple (k, A, L) encrypted twice, once with Alice's private key and once with Bob's. Alice uses the nonce N_A to match the reply to her request. This prevents her from accepting old messages possibly being replayed by the adversary. She also checks that A is her own ID and that the expiration time L is still in the future.

Notation:

A	client
B	server
T	trusted Kerberos server
K_{AT}	private key shared by A and T
K_{BT}	private key shared by B and T
k	session key chosen by T
N_A	nonce (random string) chosen by A
T_A	timestamp on A's local clock
L	lifetime (expiration time)
A^*_{subkey}	secret chosen by A
B^*_{subkey}	secret chosen by A

1. Alice sends (A, B, N_A) to Trent (the trusted server).
2. Trent sends two items back to Alice:
   - $E_{K_{AT}}(k, A, L, N_A)$
   - $\text{ticket}_B = E_{K_{BT}}(k, A, L)$
3. Alice decrypts the first item and checks for validity.
   Alice sends two items to Bob:
   - $\text{ticket}_B = E_{K_{BT}}(k, A, L)$
   - $\text{authenticator} = E_k(A, T_A, A^*_{\text{subkey}})$
4. Bob decrypts the ticket using K_{BT} and checks for validity.
   Bob decrypts the authenticator using k, checks validity, and learns A^*_{subkey}.
   Bob sends $E_k(T_A, B^*_{\text{subkey}})$ to Alice.
5. Alice decrypts Bob's message using k.
   Alice checks Bob's message for validity and learns B^*_{subkey}.

Figure 1: Simplified Kerberos protocol.

Alice forwards the ticket to Bob in step 3, and Bob decrypts it in step 4, revealing the session key k. Bob checks that the ticket was issued to Alice and has not yet expired. He then uses k to decrypt the authenticator and checks that it was created by Alice. He also checks that the timestamp T_A is close to the present time on his clock and that Alice has not previously presented him with an authenticator with the same timestamp. (This is also to prevent replay attacks.) At this point, Bob knows that k is the current session key to be used to communicate with Alice.

Bob's transmission to Alice in step 4, and Alice's checks in step 5, let her know that Bob has accepted her credentials and that they have agreed on k as the current session key. The optional subkeys A^*_{subkey} and B^*_{subkey} are additional information that Alice and Bob exchange securely under the protection of k. They can be used for various purposes. For example, Alice and Bob might decide to encrypt their files using $A^*_{\mathrm{subkey}} \oplus B^*_{\mathrm{subkey}}$ as key. Even an untrustworthy Trent that kept a record of all session keys generated could not later decrypt the files unless he was also able to capture the network traffic between Alice and Bob in which A^*_{subkey} and B^*_{subkey} were exchanged.

The purpose of Alice's authenticator is to prevent Eve from using ticket_B to impersonate Alice. Eve can't generate a valid authenticator because she doesn't know k. The timestamp T_A is used to prevent Eve from intercepting a genuine authenticator from Alice and replaying it at a later time. Bob will only accept T_A once, so unless Eve is sitting between Alice and Bob, Bob will get the real authenticator from Alice before he gets a replay from Eve. Even if Eve is sitting in the middle, the messages that pass by her in steps 3 and 4 are encrypted with keys that she does not possess, so she is neither able to forge such messages nor to learn what they contain.

Note that the protocol assumes that Alice and Bob have well-synchronized clocks. If Bob's clock is very slow, then he could be tricked into accepting an old authenticator. Because clocks are synchronized using a network time synchronization protocol, one possibility for compromising Kerberos is to attack the underlying time synchronization protocol.

Don Davis `<dtd@world.std.com>` writes:

> "It might interest you that Kerberos v5 no longer depends on a separate time-synch service. In '95, I proposed (with Geer & Ts'o) a way to allow Kerberos to work without strictly-synch'ed clocks, yet without the overhead of replacing all timestamps with challenge-response handshakes, and in fact without requiring any changes to the v5 protocol spec:
>
> `http://world.std.com/~dtd/#synch`
>
> Our proposal was adopted and implemented in the MIT implementation of Kerberos, and I believe this relaxation of Kerberos's time-synch requirement has contributed substantially to Kerberos's more widespread adoption.
>
> Our paper had two basic ideas:
>
> - Instead of synchronizing clocks, each Kerberos participant should keep track of the clock-skew between its own clock and the kdc's clock, so as to infer the kdc's time-of-day when preparing & interpreting timestamps;
> - Each kerberos user sends a single random challenge in its initial ticket request at login, so as to verify the freshness of the kdc's first ticket's timestamp. the user's kinit client then initializes its record of the clock-skew, by recording the difference between his local login-time and his first ticket's timestamp."

1.2 Practical considerations

As a practical protocol, Kerberos has to be concerned with both security and efficiency. Security considerations dictate that credentials should expire very quickly (on the order of seconds) so that an intercepted ticket cannot be reused later by an attacker to impersonate Alice. Efficiency considerations say the opposite, that credentials should have a relatively long lifetime (on the order of hours), for the more often tickets must be issued, the greater the load on the trusted server. Kerberos solves this dilemma by having two-part credentials. Tickets have a relatively long lifetime and can be used many times. Authenticators have a relatively short lifetime and can be used only once. Only the trusted server can create tickets, but Alice can create a new authenticator each time she wishes to talk to Bob.

Even with tickets having relatively long lifetimes, the need for a single centralized ticket server would prevent Kerberos from being scalable beyond a modest-sized network. To overcome this problem, the full Kerberos protocol splits the trusted server into two parts, an *authentication server (AS)* and a *ticket-granting server (TGS)*. This allows the nodes in a large network to be partitioned into several groups, each with its own server that knows only about the other servers and about the nodes in its group. Now, for Alice to contact Bob, she first goes to her AS to get a ticket that lets her talk securely to Bob's TGS. She then gets a ticket from Bob's TGS that lets her talk to Bob.

Further details of Kerberos are given in Section 12.4 of your textbook (Wenbo Mao).