Cryptography and Network Security
Chapter 14

Authentication
Applications
Authentication Applications

• will consider authentication functions
• developed to support application-level authentication & digital signatures
• will consider Kerberos – a private-key authentication service
• then X.509 - a public-key directory authentication service
Kerberos

- trusted key server system from MIT
- provides centralised private-key third-party authentication in a distributed network
  - allows users access to services distributed through network
  - without needing to trust all workstations
  - rather all trust a central authentication server
- two versions in use: 4 & 5
Kerberos Requirements

• its first report identified requirements as:
  – secure
  – reliable
  – transparent
  – scalable

• implemented using an authentication protocol based on Needham-Schroeder
Kerberos v4 Overview

• a basic third-party authentication scheme
• have an Authentication Server (AS)
  – users initially negotiate with AS to identify self
  – AS provides a non-corruptible authentication credential (ticket granting ticket TGT)
• have a Ticket Granting server (TGS)
  – users subsequently request access to other services from TGS on basis of users TGT
A Simple Authentication Dialogue:

- The user logs on to a workstation C and requests access to server V.
- C requests the user’s password and then sends a message to authentication server AS.
  - The message includes user’s ID, the servers ID and user’s password.
- The AS checks its database
  - to see if the user has supplied the proper password for this user ID.
  - and whether this user is permitted access to server V.
- If both tests are passed
  - the AS accepts the user as authentic.
  - and convince the server that this user is authentic.
- To do so,
  - the AS creates a ticket that contains the user’s ID, network address of C and the server’s ID.
  - and encrypts the ticket using the secret key shared by the AS and server V.
  - As sends the encrypted ticket to C (because the ticket is encrypted it cannot be altered by C or by an opponent.
- With this ticket C can now apply to V for service.

C -> AS: \( \text{ID}_C \ | \ | \ P_C \ | \ | \ \text{ID}_V \)

AS -> C: Ticket

C -> V: \( \text{Id}_C \ | \ | \ \text{Ticket} \)

Ticket = \( E_{KV}[\text{ID}_C \ | \ | \ \text{AD}_C \ | \ | \ \text{ID}_V] \)
Drawback

- User needs a new ticket for every different service.
- Plaintext password could be easily captured by an eavesdropper.
Kerberos v4 Dialogue

1. obtain ticket granting ticket from AS
   • once per session
2. obtain service granting ticket from TGT
   • for each distinct service required
3. client/server exchange to obtain service
   • on every service request
A MORE SECURE AUTHENTICATION DIALOGUE:

To solve these additional problems, we propose a new server called ticket-granting server (TGS) and avoiding plaintext password.

- **Once per user logon session**

  C -> AS:  IDC || ID_{tgS}
  AS -> C:  E_{KC}[Ticket_{tgS}]

- **Once per type of service**

  C -> TGS:  IDC || IDV || Ticket_{tgS}
  TGS -> C:  TicketV

- **Once per service session**

  C -> V:  IDC || TicketV

  Ticket_{tgS} = E_{K_{tgS}} [IDC || ADc || ID_{tgS} || TS1 || Lifetime1]
  TicketV = E_{KV} [IDC || ADc || IDV || TS2 || Lifetime2]
Figure 14.1  Overview of Kerberos

1. User logs on to workstation and requests service on host.

3. Workstation prompts user for password and uses password to decrypt incoming message, then sends ticket and authenticator that contains user's name, network address, and time to TGS.

5. Workstation sends ticket and authenticator to server.

2. AS verifies user's access right in database, creates ticket-granting ticket and session key. Results are encrypted using key derived from user's password.

4. TGS decrypts ticket and authenticator, verifies request, then creates ticket for requested server.

6. Server verifies that ticket and authenticator match, then grants access to service. If mutual authentication is required, server returns an authenticator.
### Table 14.1 Summary of Kerberos Version 4 Message Exchanges

<table>
<thead>
<tr>
<th>(a) Authentication Service Exchange: to obtain ticket-granting ticket</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) C → AS: ( ID_c \parallel ID_{tgs} \parallel TS_1 )</td>
</tr>
<tr>
<td>(2) AS → C: ( E_{K_c}\left[ K_{c,tgs} \parallel ID_{tgs} \parallel TS_2 \parallel Lifetime_2 \parallel Ticket_{tgs} \right] )</td>
</tr>
<tr>
<td>( Ticket_{tgs} = E_{K_{tgs}}\left[ K_{c,tgs} \parallel ID_C \parallel AD_C \parallel ID_{tgs} \parallel TS_2 \parallel Lifetime_2 \right] )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(b) Ticket-Granting Service Exchange: to obtain service-granting ticket</th>
</tr>
</thead>
<tbody>
<tr>
<td>(3) C → TGS: ( ID_v \parallel Ticket_{tgs} \parallel Authenticator_c )</td>
</tr>
<tr>
<td>(4) TGS → C: ( E_{K_{c,tgs}}\left[ K_{c,v} \parallel ID_v \parallel TS_4 \parallel Ticket_v \right] )</td>
</tr>
<tr>
<td>( Ticket_{tgs} = E_{K_{tgs}}\left[ K_{c,tgs} \parallel ID_C \parallel AD_C \parallel ID_{tgs} \parallel TS_2 \parallel Lifetime_2 \right] )</td>
</tr>
<tr>
<td>( Ticket_v = E_{K_v}\left[ K_{c,v} \parallel ID_C \parallel AD_C \parallel ID_v \parallel TS_4 \parallel Lifetime_4 \right] )</td>
</tr>
<tr>
<td>( Authenticator_c = E_{K_{tgs}}\left[ ID_C \parallel AD_C \parallel TS_3 \right] )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(c) Client/Server Authentication Exchange: to obtain service</th>
</tr>
</thead>
<tbody>
<tr>
<td>(5) C → V: ( Ticket_v \parallel Authenticator_c )</td>
</tr>
<tr>
<td>(6) V → C: ( E_{K_{c,v}}[TS_5 + 1] ) (for mutual authentication)</td>
</tr>
<tr>
<td>( Ticket_v = E_{K_v}\left[ K_{c,v} \parallel ID_C \parallel AD_C \parallel ID_v \parallel TS_4 \parallel Lifetime_4 \right] )</td>
</tr>
<tr>
<td>( Authenticator_c = E_{K_{c,v}}\left[ ID_C \parallel AD_C \parallel TS_5 \right] )</td>
</tr>
</tbody>
</table>
See MIT web-site: http://web.mit.edu/kerberos/www/krb5-1.4/krb5-1.4.2/doc/krb5-user/

**WHAT IS TICKET?**

- Your Kerberos *credentials*, or "tickets", are a set of electronic information that can be used to verify your identity. Your Kerberos tickets may be stored in a file, or they may exist only in memory.

- The first ticket you obtain is a *ticket-granting ticket*, which permits you to obtain additional tickets. These additional tickets give you permission for specific services. The requesting and granting of these additional tickets happens transparently.

- A good analogy for the ticket-granting ticket is a three-day ski pass that is good at four different resorts. You show the pass at whichever resort you decide to go to (until it expires), and you receive a lift ticket for that resort. Once you have the lift ticket, you can ski all you want at that resort. If you go to another resort the next day, you once again show your pass, and you get an additional lift ticket for the new resort. The difference is that the Kerberos V5 programs notice that you have the weekend ski pass, and get the lift ticket for you, so you don't have to perform the transactions yourself.
WHAT IS A Kerberos Principal?

A Kerberos principal is a unique identity to which Kerberos can assign tickets. By convention, a principal is divided into three parts: the primary, the instance, and the realm. The format of a typical Kerberos V5 principal is `primary/instance@REALM`.

- The *primary* is the first part of the principal. In the case of a user, it's the same as your username. For a host, the primary is the word `host`.

- The *instance* is an optional string that qualifies the primary. The instance is separated from the primary by a slash (`/`). In the case of a user, the instance is usually null, but a user might also have an additional principal, with an instance called `~admin`, which he/she uses to administrate a database. The principal `jennifer@ATHENA.MIT.EDU` is completely separate from the principal `jennifer/admin@ATHENA.MIT.EDU`, with a separate password, and separate permissions. In the case of a host, the instance is the fully qualified hostname, e.g., `daffodil.mit.edu`.

- The *realm* is your Kerberos realm. In most cases, your Kerberos realm is your domain name, in upper-case letters. For example, the machine `daffodil.mit.edu` would be in the realm `ATHENA.MIT.EDU`. (See figure 14.2)
Kerberos Realms

- A Kerberos environment consists of:
  - A Kerberos server
  - A number of clients, all registered with server
  - Application servers, sharing keys with server
- This is termed a realm
  - Typically a single administrative domain
- If have multiple realms, their Kerberos servers must share keys and trust
TICKET MANAGEMENT

- On many systems, Kerberos is built into the login program, and you get tickets automatically when you log in.
- Other programs, such as rsh, rcp, telnet, and rlogin, can forward copies of your tickets to the remote host. Most of these programs also automatically destroy your tickets when they exit.
- However, MIT recommends that you explicitly destroy your Kerberos tickets when you are through with them, just to be sure.
- One way to help ensure that this happens is to add the kdestroy command to your .logout file.
- Additionally, if you are going to be away from your machine and are concerned about an intruder using your permissions, it is safest to either destroy all copies of your tickets, or use a screensaver that locks the screen.

See MIT web-site: http://web.mit.edu/kerberos/www/krb5-1.4/krb5-1.4.2/doc/krb5-user/
OBTAINING TICKETS WITH \texttt{kinit}

- If your site is using the Kerberos V5 login program, you will get Kerberos tickets automatically when you log in.
- If your site uses a different login program, you may need to explicitly obtain your Kerberos tickets, using the \texttt{kinit} program.
- Similarly, if your Kerberos tickets expire, use the \texttt{kinit} program to obtain new ones.
- To use the \texttt{kinit} program, simply type \texttt{kinit} and then type your password at the prompt.
- For example, Jennifer (whose username is \texttt{jennifer}) works for Bleep, Inc. (a fictitious company with the domain name \texttt{mit.edu} and the Kerberos realm \texttt{ATHENA/MIT.EDU}).

She would type:
\begin{verbatim}
shell\% kinit
Password for jennifer@ATHENA.MIT.EDU: <-- [Type jennifer's password here.]
\end{verbatim}
If you type your password incorrectly, kinit will give you the following error message:

```
shell% kinit
Password for jennifer@ATHENA.MIT.EDU: <-- [Type the wrong password here.]
kinit: Password incorrect
```
and you won't get Kerberos tickets.

Notice that `kinit` assumes you want tickets for your own username in your default realm.

Suppose Jennifer's friend David is visiting, and he wants to borrow a window to check his mail. David needs to get tickets for himself in his own realm, FUBAR.ORG.

He would type:

```
shell% kinit david@FUBAR.ORG
Password for david@FUBAR.ORG: <-- [Type david's password here.]
```
David would then have tickets which he could use to log onto his own machine. Note that he typed his password locally on Jennifer's machine, but it never went over the network.

Kerberos on the local host performed the authentication to the KDC (Key Distribution Center) in the other realm.
If you want to be able to forward your tickets to another host, you need to request *forwardable* tickets. You do this by specifying the `-f` option:

```
shell% kinit -f
Password for jennifer@ATHENA.MIT.EDU: <-- [Type your password here.]
shell%
```

Note that `kinit` does not tell you that it obtained forwardable tickets; you can verify this using the `klist` command.

Normally, your tickets are good for your system's default ticket lifetime, which is ten hours on many systems. You can specify a different ticket lifetime with the `-l` option. Add the letter `s` to the value for seconds, `m` for minutes, `h` for hours, or `d` for days. For example, to obtain forwardable tickets for david@FUBAR.ORG that would be good for three hours, you would type:

```
shell% kinit -f -l 3h david@FUBAR.ORG
Password for david@FUBAR.ORG: <-- [Type david's password here.]
shell%
```

You cannot mix units; specifying a lifetime of `3h30m` would result in an error. Note also that most systems specify a maximum ticket lifetime. If you request a longer ticket lifetime, it will be automatically truncated to the maximum lifetime.
Viewing Your Tickets

The klist command shows your tickets. When you first obtain tickets, you will have only the ticket-granting ticket. The listing would look like this:

```
she11% klist
Ticket cache: /tmp/krb5cc_ttypa
Default principal: jennifer@ATHENA.MIT.EDU

Valid starting   Expires    Service principal
06/07/96 19:49:21 06/08/96 05:49:19 krbtgt/ATHENA.MIT.EDU@ATHENA.MIT.EDU
she11%
```

The ticket cache is the location of your ticket file. In the above example, this file is named /tmp/krb5cc_ttypa. The default principal is your kerberos principal.

The "valid starting" and "expires" fields describe the period of time during which the ticket is valid. The service principal describes each ticket. The ticket-granting ticket has the primary krbtgt, and the instance is the realm name.
Now, if jennifer connected to the machine daffodil.mit.edu, and then typed klist again, she would have gotten the following result:

```
shell% klist
Ticket cache: /tmp/krb5cc_tttyp
Default principal: jennifer@ATHENA/MIT/EDU

Valid starting       Expires       Service principal
06/07/96 19:49:21    06/08/96 05:49:19    krbtgt/ATHENA/MIT/EDU@ATHENA/MIT/EDU
06/07/96 20:22:30    06/08/96 05:49:19    host/daffodil.mit.edu@ATHENA/MIT/EDU
shell%
```

Here's what happened: when jennifer used telnet to connect to the host daffodil.mit.edu, the telnet program presented her ticket-granting ticket to the KDC and requested a host ticket for the host daffodil.mit.edu. The KDC sent the host ticket, which telnet then presented to the host daffodil.mit.edu, and she was allowed to log in without typing her password.

Suppose your Kerberos tickets allow you to log into a host in another domain, such as trillium.fubar.org, which is also in another Kerberos realm, FUBAR.ORG. If you telnet to this host, you will receive a ticket-granting ticket for the realm FUBAR.ORG, plus the new host ticket for trillium.fubar.org. klist will now show:
```
shell% klist
Ticket cache: /tmp/krb5cc_ttypa
Default principal: jennifer@ATHENA.MIT.EDU

<table>
<thead>
<tr>
<th>Valid starting</th>
<th>Expires</th>
<th>Service principal</th>
</tr>
</thead>
<tbody>
<tr>
<td>06/07/96 19:49:21</td>
<td>06/08/96 05:49:19</td>
<td>krbtgt/ATHENA.MIT.EDU@ATHENA.MIT.EDU</td>
</tr>
<tr>
<td>06/07/96 20:22:30</td>
<td>06/08/96 05:49:19</td>
<td>host/daffodil.mit.edu@ATHENA.MIT.EDU</td>
</tr>
<tr>
<td>06/07/96 20:24:18</td>
<td>06/08/96 05:49:19</td>
<td>krbtgt/FUBAR.ORG@ATHENA.MIT.EDU</td>
</tr>
<tr>
<td>06/07/96 20:24:18</td>
<td>06/08/96 05:49:19</td>
<td>host/trillium.fubar.org@ATHENA.MIT.EDU</td>
</tr>
</tbody>
</table>
```

dell%
Kerberos Version 5

- developed in mid 1990’s
- specified as Internet standard RFC 1510
- provides improvements over v4
  - addresses environmental shortcomings
    - encryption alg, network protocol, byte order, ticket lifetime, authentication forwarding, interrealm auth
  - and technical deficiencies
    - double encryption, non-std mode of use, session keys, password attacks
X.509 Authentication Service

- part of CCITT X.500 directory service standards
  - distributed servers maintaining user info database
- defines framework for authentication services
  - directory may store public-key certificates
  - with public key of user signed by certification authority
- also defines authentication protocols
- uses public-key crypto & digital signatures
  - algorithms not standardised, but RSA recommended
- X.509 certificates are widely used
X.509 Certificates

• issued by a Certification Authority (CA), containing:
  – version (1, 2, or 3)
  – serial number (unique within CA) identifying certificate
  – signature algorithm identifier
  – issuer X.500 name (CA)
  – period of validity (from - to dates)
  – subject X.500 name (name of owner)
  – subject public-key info (algorithm, parameters, key)
  – issuer unique identifier (v2+)
  – subject unique identifier (v2+)
  – extension fields (v3)
  – signature (of hash of all fields in certificate)
• notation \texttt{CA<<A>>} denotes certificate for A signed by CA
X.509 Certificates
Obtaining a Certificate

• any user with access to CA can get any certificate from it
• only the CA can modify a certificate
• because cannot be forged, certificates can be placed in a public directory
CA Hierarchy

• if both users share a common CA then they are assumed to know its public key
• otherwise CA's must form a hierarchy
• use certificates linking members of hierarchy to validate other CA's
  – each CA has certificates for clients (forward) and parent (backward)
• each client trusts parents certificates
• enable verification of any certificate from one CA by users of all other CAs in hierarchy
CA Hierarchy Use
Using Forward and Reverse Certificates

• The certificate path from A to B is 
  X<<W>>W<<V>>V<<Y>>Y<<Z>>Z<<B>>>

• What is the path from B to A?
Certificate Revocation

• certificates have a period of validity
• may need to revoke before expiry, eg:
  1. user's private key is compromised
  2. user is no longer certified by this CA
  3. CA's certificate is compromised
• CA’s maintain list of revoked certificates
  – the Certificate Revocation List (CRL)
• users should check certificates with CA’s CRL
Authentication Procedures

• X.509 includes three alternative authentication procedures:
  • One-Way Authentication
  • Two-Way Authentication
  • Three-Way Authentication
  • all use public-key signatures
One-Way Authentication

- 1 message (A->B) used to establish
  - the identity of A and that message is from A
  - message was intended for B
  - integrity & originality of message
- message must include timestamp, nonce, B's identity and is signed by A
- may include additional info for B
  - eg session key
Two-Way Authentication

- 2 messages (A->B, B->A) which also establishes in addition:
  - the identity of B and that reply is from B
  - that reply is intended for A
  - integrity & originality of reply
- reply includes original nonce from A, also timestamp and nonce from B
- may include additional info for A
Three-Way Authentication

• 3 messages (A->B, B->A, A->B) which enables above authentication without synchronized clocks
• has reply from A back to B containing signed copy of nonce from B
• means that timestamps need not be checked or relied upon
Summary

(a) One-way authentication
1. $A(t_A, r_A, ID_B, sgnData, E[PU_b, K_{ab}])$

(b) Two-way authentication
1. $A(t_A, r_A, ID_B, sgnData, E[PU_b, K_{ab}])$
2. $B(t_B, r_B, ID_A, r_A, sgnData, E[PU_a, K_{ba}])$

(c) Three-way authentication
1. $A(t_A, r_A, ID_B, sgnData, E[PU_b, K_{ab}])$
2. $B(t_B, r_B, ID_A, r_A, sgnData, E[PU_a, K_{ba}])$
3. $A(r_B)$
X.509 Version 3

• has been recognised that additional information is needed in a certificate
  – email/URL, policy details, usage constraints
• rather than explicitly naming new fields defined a general extension method
• extensions consist of:
  – extension identifier
  – criticality indicator
  – extension value
Certificate Extensions

• key and policy information
  – convey info about subject & issuer keys, plus indicators of certificate policy

• certificate subject and issuer attributes
  – support alternative names, in alternative formats for certificate subject and/or issuer

• certificate path constraints
  – allow constraints on use of certificates by other CA’s
Public Key Infrastructure

Users, servers, routers, etc.

CRL is certificate revocation list