Security protocols: Kerberos

Carlo U. Nicola, SGI FH Aargau
With extracts from publications of:
William Stallings; ISG, Royal Halloway UCL;
Samson Garfinkel, Gene Spafford, Chris Clifton, Purdue University.
Needham-Schroeder protocol

N : Number used once (nonce);
||: concatenation

(1) A → T: A || B || N_A
(2) T → A: \{N_A || B || K || \{K || A\}_{K_{B,T}}\}_{K_{A,T}}
(3) A → B: \{K || A\}_{K_{B,T}}
(4) B → A: N_B
(5) A → B: \{N_B || B\}_K
1. In messages 1 and 2, Alice and Tom (the Trusted Third Party (TTP)) interact:
   → Tom gives Alice a session key K and authenticates himself.
2. In messages 3, 4 and 5, the interaction is between Alice and Bob.
   → Alice transfers an encrypted copy of the session key K to Bob in message 3.
   → Alice is authenticated to Bob in messages 4 and 5 using the key K that they now share.
   → Actual Needham-Schroeder protocol uses a slightly different mechanism in this step.
Security issues

1. T is authenticated by A.
2. A is not authenticated by T, but A can only decrypt message 2 if she has the correct key $K_{A,T}$.
3. A is authenticated to B.
4. If key K is used for subsequent encryption or MAC-ing, then we get implicit authentication of B to A.
   → Recipient of message 3 can only obtain K if he knows the correct key $K_{B,T}$.
5. Can make authentication between A and B mutual and explicit.
   → e.g. A issues a challenge to B as part of message 3 and B responds in message 4.
6. Session key established: chosen by T, the KDC (Key Distribution Centre).
Advantages

1. Key storage efficiency: only $n$ keys to look after at KDC.
2. Only one long-term key per client ($K_{A,T}$) instead of $n-1$.
3. Only uses symmetric key cryptography.
4. Bob can be off-line in steps 1 and 2 and TTP can be off-line in steps 3, 4 and 5.
5. It also possible for Alice to obtain $K$ from TTP, cache it and then use it later with Bob.
Disadvantages

1. KDC is a single point of failure: security and availability.
2. Potential computation/communication bottleneck at KDC.
3. Requirement for an on-line, trusted server.
   → TTP knows all session keys and all long-term keys.
4. How can we ensure clients look after their long-term keys properly?
   → If long-term key compromised, then entities can be impersonated.
   → This is an issue in non-TTP based solutions too.
5. Old session keys are useful to Oscar.
A corporate network, with KDC (a server managed by IT department), and hosts being networked resources or users’ machines like: printers, storage, computational resources. The Needham-Schroeder approach simplifies key management in such a network.

KDC can also act as an access control or authorization server.

- In message 1, A requests a key for communication with B (and access to) host B.
- KDC can decide whether or not to provide A with access to B, by sending or withholding message 2.
- KDC decision should be based on an access control policy.
- B configured not to grant access to A unless A’s requests in message 3 are of the correct form.
- Protocol messages can be extended to include more detailed authorizations specifying to which services at host B the requesting host should be granted access.
Quiz: A simple authentication dialogue

(1) C → AS: ID_c || P_c || ID_v  (P_c is a random number generated by C)
(2) AS → C: Ticket
(3) C → V: ID_c || Ticket

Ticket = \{ID_c || P_c || ID_v\}_{K_v}
Key concerns are **confidentiality** and **timeliness**:

1. to provide **confidentiality** we must encrypt identification and session key info;
   → which requires the use of a previously shared private or public keys
2. need **timeliness** to prevent **replay attacks**
   → which is provided by using **sequence numbers, timestamps** or **challenge/response** protocols.
Kerberos

1. Kerberos is a TTP-aided authentication protocol based on Needham-Schroeder.
2. It’s also a software package implementing that protocol, currently Kerberos v5 Release 1.10.
3. Kerberos is developed by Project Athena at MIT.
   → http://web.mit.edu/kerberos/
   → Versions 1-3 unreleased, v4 obsolete but still used.
5. A version of Kerberos is integral to Windows BS.
6. Kerberos is integrated into many versions of Unix and used by kerberized applications.
Problem: Users want to access services on different servers in a network. Three threats exist:

- User \textit{pretends} to be another user.
- User \textit{alters} the network address of a workstation.
- User \textit{eavesdrops} on exchanges and use a replay attack to fake its true identity.

Kerberos: In Greek mythology, a many headed dog, the guardian of the entrance of Hades (Hell). It becomes the name of the first practical SSO (Single Sign On) protocol.
Kerberos solution: general idea

1. It provides a centralized authentication server to authenticate users to servers and servers to users.
2. Relies on conventional encryption, making no use of public-key encryption.
3. It issues time limited tickets to authenticated users in order to use resources in the network.
Kerberos glossary

C = Client
AS = Authentication Server
V = server whose services or applications we need
ID_c = Identifier of user on C
ID_v = identifier of V
P_c = password of user on C
AD_c = network address of C
K_v = secret encryption key shared by AS and V
TS = timestamp
|| = concatenation
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.
Version 4/5: Authentication dialogue

Authentication Service Exchange: To obtain Ticket-Granting Ticket
(1) \( C \rightarrow AS: \quad ID_c \mid \mid ID_{tgs} \mid \mid TS_1 \)
(2) \( AS \rightarrow C: \quad \{K_{c,tgs} \mid \mid ID_{tgs} \mid \mid TS_2 \mid \mid \text{Lifetime}_2 \mid \mid \text{Ticket}_{tgs}\} \quad KC \)

Ticket-Granting Service Exchange: To obtain Service-Granting Ticket
(3) \( C \rightarrow TGS: \quad ID_v \mid \mid \text{Ticket}_{tgs} \mid \mid \text{Authenticator}_c \)
(4) \( TGS \rightarrow C: \quad \{K_{c,v} \mid \mid ID_v \mid \mid TS_4 \mid \mid \text{Ticket}_v\} \quad KC,tgs \)

Client/Server Authentication Exchange: To obtain the service
(5) \( C \rightarrow V: \quad \text{Ticket}_v \mid \mid \text{Authenticator}_c \)
(6) \( V \rightarrow C: \quad \{TS_5 +1\} \quad KC,v \)
First step: C to AS

Kerberos authentication server

$K_C$, $ID_c || ID_{tgs} || TS_1$, $K_C$, $K_{TGS}$
Second step: AS to C

Ticket for TGS encrypted with key $K_{TGS}$

Encrypted with key $K_C$

$\|ID_{TGS}\|TS_2\|\text{Lifetime}_2\|\text{Ticket}_{TGS}$
Third step: C to TGS

Remember $\text{Ticket}_{TGS}$ is:

$$||\text{ID}_c||\text{AD}_C||\text{ID}_{TGS}||\text{TS}_2||\text{Lifetime}_2$$

**Diagram:**
- $K_C$
- $K_{C,TGS}$
- $K_{TGS}$
- $K_V$

**Flow:**
1. $ID_c||ID_V||\text{Ticket}_{TGS}||\text{Authenticator}_{C,TGS}$
2. Ticket granting server (TGS)

**Encryption:**
- Authenticator$_{C,TGS}$ encrypted with key $K_{C,TGS}$
Fourth step: TGS to C

Encrypted with key $K_{C,TGS}$

$K_{C,V} \parallel ID_C \parallel AD_C \parallel ID_V \parallel TS_4 \parallel \text{Lifetime}_4$

Encrypted with key $K_V$: Ticket$_V$ for computational server
Fifth step: C to V

Remember Ticket_v:

\[ K_{C,v} \| ID_C \| AD_C \| ID_v \| TS_4 \| \text{Lifetime}_4 \]

Authentication \( C_v \) encrypted with key \( K_{C,v} \)

ID_C || Ticket_v || Authenticator_{C,v}

ID_C || AD_C || TS_5

Computational server

V

K_v

K_C

K_{C,TGS}

K_{C,v}
Sixth step: V to C

Encrypted with key $K_{C,V}$

$TS_5 + 1$
Difference between version 4 and 5

a) Encryption system dependence (v.4 DES with non standard PCBC, v.5 you can choose the encryption algorithm and use CBC)

b) Internet protocol dependence (v.4 only IP; v.5 any type)

c) Message byte ordering (v.4 arbitrary; v.5 defined by ASN1 Standard)

d) Ticket lifetime (v.4 21 h max; v.5 arbitrary)

e) Authentication forwarding to other hosts (v.4 no; v.5 yes). Example: A client issues a request to a print server that then accesses the client’s file from a file server, using the client’s credentials for access.

f) Inter-realm authentication: v.4 N^2 (!) realm to realm relationships (v.5 simpler)
Currently we have two Kerberos versions:
4 : better restricted to a single realm
5 : allows inter-realm authentication with less overhead than v. 4.

Kerberos v5 is an Internet standard specified in RFC1510, and used by many utilities.

To use Kerberos:
a) you need a KDC on your network;
b) you should have “kerberised” applications running on all participating systems.
Ticket lifetime problem

Lifetime associated with the ticket-granting ticket:

a) If too short → the user is repeatedly asked for the password
b) If too long → a greater opportunity to replay exists.

The threat is that a cracker will steal the ticket and use it before it expires.
Kerberos supports cross-realm authentications (federations).
- Allows clients in one realm to access servers in another realm.
- Requires pre-agreement between relevant AS/TGS pairs.

Mechanism used is *forwardable* tickets.
- Client in realm A requests TGT from normal TGS for use in another realm B.
- TGS in realm A grants TGT for realm B.
  • So TGS needs to know key $K_{AS,TGS}$ that is valid in realm B.
  • TGS in realm A sets a forwardable flag in the issued TGT.
- Client from realm A can now present TGT in realm B.
Request for service in another realm V.5
Kerberos issues (1)

1. Lack of revocation: ticket granting tickets valid until they expire, typically 10 hours. What if compromised?

2. Key management within realms: long-term keys need to be established between AS and TGS, TGS and Servers and AS and clients.

3. Synchronous clocks are needed, protected against timing attacks. Caches of recent messages to protect against replay within clock skew.

4. Availability: need for on-line AS and TGS, that are trusted by clients and are hard to eavesdrop.
Kerberos issues (2)

5. Key storage: short-term keys and ticket granting tickets located on largely unprotected client hosts.

6. Passwords: in most deployments, the Client-AS long-term key $K_{AS,C}$ is usually based on a password entered by user at the start of a session
   a) This key is used to encrypt messages with predictable formats.
   b) So Kerberos vulnerable to dictionary attacks if the password is not well chosen.
   c) Details in paper by Thomas Wu at: http://citeseer.ist.psu.edu/wu99realworld.html
   d) Ultimately, then, security is dependent on users and the quality of the passwords they can be persuaded to remember.

7. Code vulnerabilities: many found over the years: http://web.mit.edu/kerberos/advisories/
Microsoft have adopted and extended Kerberos for network authentication since Windows 2000.

First extension:
- Support for public-key encryption to protect client/AS messages (rather than password-based long-term key).
- Allows use of authentication based on client smart cards.

Second extension:
- Use of Kerberos data authorization field (normally empty)
- Transports Win2K access privileges in the form of SIDs, derived from Active Directory
- these are compared to ACLs of remote objects to make access decisions.

Message formats are published, but proprietary to Microsoft. Non-standard extension to Kerberos makes it difficult to interoperate Microsoft and non-Microsoft implementations.
Kerberos is an example of a Single Sign On (SSO) system. User enters a single password, and obtains seamless access to multiple network services or applications.

Microsoft Passport: an example of a web-based SSO solution, aimed at e-commerce consumers.

*Shibboleth*: an open, standards-based effort at achieving *federated network identity*, a concept related to SSO.

Many vendors currently offer similar SSO/password management products.
Lessons learned?

Designing protocols is easy. But designing **secure** protocols is **hard**
- there are many infamous failures in the literature.

Some good protocols are already standardised (e.g. ISO 9798, ITU-T X.509, …)

Security weaknesses arise from many sources:
- errors in specification and implementation,
- side-channels,
- lack of user training,
- host insecurities,
- poor random number generation,…

The problem of verifying security gets harder as the protocols get more complex.
Lessons learned?

"On the Internet, nobody knows you’re a dog."
Appendices
Practical Kerberos
To use Kerberos, you must first establish a Kerberos principal. A Kerberos principal is like a regular account on a machine. The name of the principal looks like `your_name@YOUR.REALM`. The part before the `@` is a string that you choose. Usually, it's the same as your regular account name. The part after the `@` is the name of the realm.

Associated with each principal is a name, a password, and some other information. This information is stored in the Kerberos database, and is encrypted using a Kerberos master key.

For the user, Kerberos is nearly transparent. There are a number of services which require Kerberos authentication. An example is rlogin. To use one of them, you need to obtain a TGT first. The command for this is `kinit`:

```
% kinit
Password for your_name@YOUR.REALM:
```

`kinit` runs locally on the machine you are logged in. (No transmission of password on insecure lines).
How to use Kerberos (2)?

When you enter in your password, the `kinit` program submits a request to the AS for a TGT. The password is used to compute a key, which in turn is used to decrypt part of the response from the AS. (This is the part that contains the confirmation of the request, as well as the session key.) If you enter the correct password, you get a TGT. You can verify this by using the command `klist`:

```
% klist
Ticket cache: /var/tmp/krb5cc_1234
Default principal: your_name@YOUR.REALM
```

```
Valid starting       Expires       Service principal
24-Sep-01 12:58:02   24-Sep-01 20:58:15  krbtgt/YOUR.REALM@YOUR.REALM
```

The ticket cache field tells you which file contains your credentials cache. The default principal is the principal that the TGT is for (you). The remainder of the output is a list of your existing tickets. Since you've just started, there's only one. The service principal (`krbtgt`, etc) shows that this ticket is a TGT. Note that it's good for a short time: approximately 8 hours.
How to use Kerberos (3)?

If you now use the Kerberos version of rlogin, this program will use the TGT in your credentials cache to request a ticket for the rlogin daemon on the machine you're logging into. This happens automatically, so all you see is the following:

```
% rlogin newhost.domain
Last login: Fri Jul 21 12:04:40 from etc etc
```

You'll notice something new if you read the contents of your cache:

```
% klist
Ticket cache: /var/tmp/krb5cc_1234
Default principal: your_name@YOUR.REALM
Valid starting       Expires       Service principal
24-Sep-01 12:58:02  24-Sep-01 20:58:15  krbtgt/YOUR.REALM@YOUR.REALM
24-Sep-01 13:03:33  24-Sep-01 20:58:15  host/newhost.domain@YOUR.REALM
```
Service principal: The first component (the part before the /) is the base principal name. The second component (between the / and @) is called the instance. For services, this is usually the hostname that the service is running on, although in the case of Kerberos services, it's the realm name. For users, it's usually null (in which case, there's no slash, either), or when the user is accessing some privileged item, some tag to indicate this (such as your_name/admin or your_name/secure). The last component (after @) is the realm name, as before.

The default action for rlogin is to leave any tickets that it obtains in the cache. This represents a security problem if someone can hijack the terminal/station that you're currently logged into.

Nevertheless, you may wish not to leave credentials lying around in your cache, in which case you can perform a kdestroy:

```
% kdestroy
% klist

klist: No credentials cache file found while setting cache flags
(ticket cache /var/tmp/krb5cc_1234)

kdestroy removes all tickets (including the TGT) from your cache.
```
A commercial Kerberos light
RSA SecureID authentication

Must be sent via SSL, SSH or IP_{SEC}: it is not at all clear if all RSA apps are so programmed!

Password, Actual number from SecurID

Computational server with ACE Server SW: login, ftp, telnet. User’s applications must use an API to be RSA SecurID compliant.
How can the server predict whether the actual SecurID number is the correct one for a certain user?

The magic number is stored in a secret file on the server whose access is determined by the token’s S/N associated with the user’s name.
What follows, shows that the writing of crypto programs is better left to the professionals and that security by obfuscation is very dangerous.

I.C. Wiener analysis of SecurID (2005)
64-bit time computation

```c
INT64 time64; INT32 time; UCHAR byte;
// Seconds since 01/01/86, 00:00
time = gettimeofday();  // precision: 64 bit --> 32 bit

// Round down time
time = time / 30;
time = time / 4;
time = time * 4;  // Result always even: LS-Nibble: 0x0,0x4,0x8,0xC.
    // Precision: 32 bit - 2 bit = 30 bit

// Expand time into 64-bit. Duplicate least significant bytes.
byte = time & 0xFF;
time = time << 8;
time = time | byte;
time64 = time;  // 1st duplicate
time64 = time64 << 32;
time64 = time64 | time;  // 2nd duplicate of LSB
    // Precision: 30 bit - 8 bit = 22 bit
```

Total number of possible times: \(2^{22} = 4192304\)
ACE-Server DB:
1) The magic number for token X is stored in a text file and can be accessed through the X-token’s serial number.

2) This special text-file is shipped with the tokens to the customers.

3) The ACE/Server’s administrator links the username with a specified token via token’s serial number.

4) The ACE-Server must be synchronized with the token clock. The lifetime of the token is limited (2, 4, 5 years).
RSA SecureID output conversion (1)

Displayed token code nibble (red) and its possible pre-convert values

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>D</td>
<td>C</td>
<td>D</td>
<td>C</td>
<td>D</td>
<td>C</td>
<td>D</td>
<td>C</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>F</td>
<td>E</td>
<td>F</td>
<td>E</td>
<td>F</td>
<td>E</td>
<td>F</td>
<td>E</td>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>

4 possible nibbles for each digit.
From the table we easily see that:

i. 0xA, 0xC and 0xE can only be mapped to 0, 2, 4, 6, 8

ii. 0xB, 0xD and 0xF can only be mapped to 1, 3, 5, 7, 9

Rightly 0xA, …, 0xF should map with uniform probability into {0,1,2,3,4,5,6,7,8,9}

6 digit token code $\rightarrow 4^6 = 4096$ possible pre-convert values!