Kerberos: http://web.mit.edu/kerberos/

- Kerberos is a TTP-based authentication protocol developed from Needham-Schroeder.
- There is (free) software implementing that protocol
- Kerberos was originally devised as part of Project Athena at MIT.
  - designed to provide a means for workstation users (clients) and servers (and vice versa) to authenticate one another.
- See also RFC 1510 – Kerberos and DASS (Distributed Authentication Security Service), RFC 1507 (asymmetric Version)
- A version of Kerberos is integral to Windows since Win2K.
- Kerberos is integrated into many versions of Unix and used by "Kerberized" applications.

Kerberos Principals

Authentication Server **(AS)**

- Authenticated by *client* at login based on *long-term key*,
- AS gives client *ticket granting ticket* and *short-term key.*
- AS provides an authentication service.

Ticket Granting Server **(TGS)**

- Authentication with client based on short-term key and ticket granting ticket.
- TGS then issues *tickets* to client which give client access to further *servers*.
- TGS provides an access control/authorization service.

Advantages of Kerberos

Logical separation of authentication and authorisation/access control.

- **But: AS and TGS are often implemented on same physical platform.**

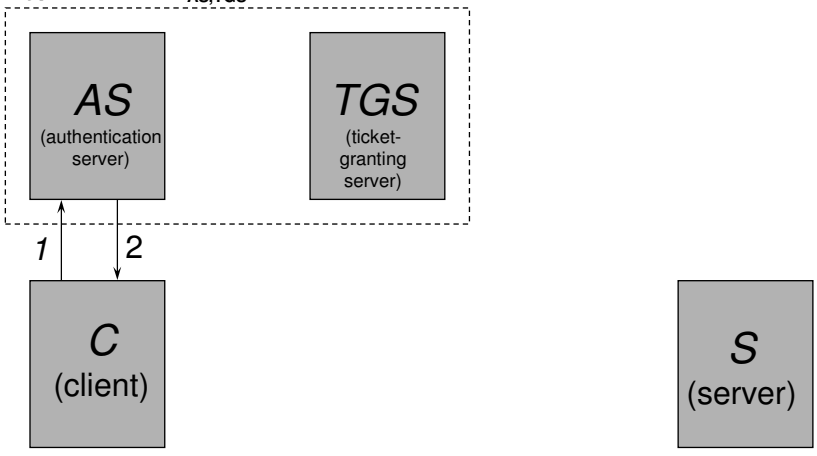
Differentiated control over lifetime of ticket granting tickets (typically 10 hours) and session tickets for actual access to services (typically 5 minutes).

A user only needs to use his long-term secret key once per 10 hour session, to establish short-term key and ticket granting ticket.

- **Convenient for users.**
- **Reduces possibility of exposure of long-term key.**

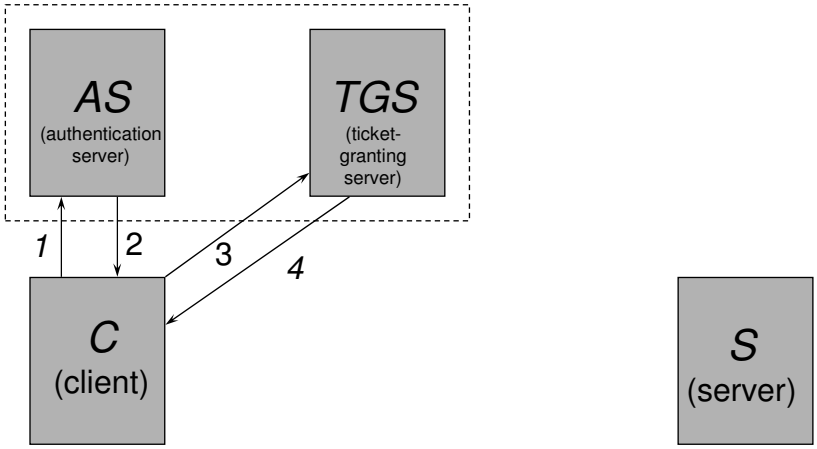
Kerberos Protocol

Messages 1 and 2 are exchanged between the client and the AS. This typically happens only once per 'log in'. A short-term key is provided by the AS. Message 2 contains the ticket granting ticket and a version of that ticket encrypted under $K_{AS,TGS}$ for the client to forward to the TGS.



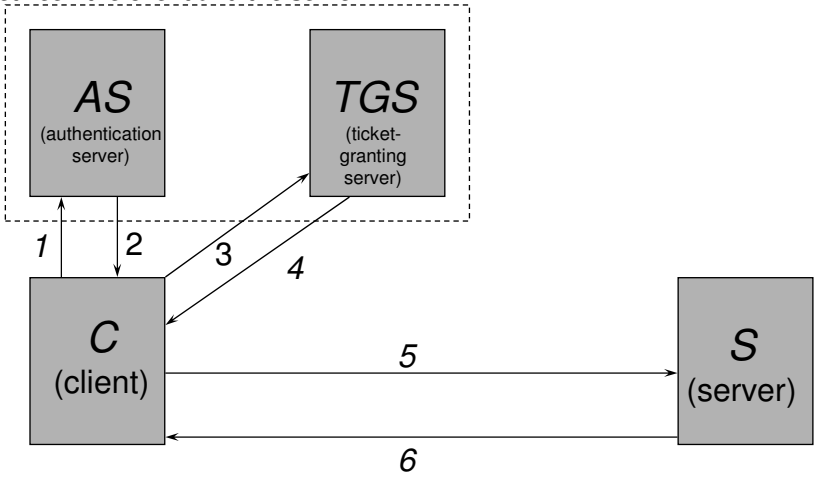
Kerberos Protocol

Messages 3 and 4 are exchanged between the client and the TGS (using the short-term key provided by the AS). Message 3 and 4 can be repeated a number of times without repeating messages 1 and 2. 3,4 messages exchange happens whenever the client wants to communicate with a new server.



Kerberos Protocol

Messages 5 and 6 are exchanged between the client and server (using a key provided by the TGS). Message 5 and 6 can be repeated a number of times without repeating messages 3 and 4, during the lifetime of the key set up between the client and the server.



Kerberos Phases

Phase 1: **In messages** 1 and 2**, C and AS use long-term key to authenticate. AS gives C short-term key and ticket granting ticket (TGT).**

Phase 2: **In messages** 3 and 4**, C and TGS use short-term key and ticket granting ticket to authenticate. TGS gives C session key and ticket.**

Phase 3: **In messaged** 5 and 6**, C and S use session key and ticket to authenticate and set up secure session.**

Phases 2 and 3 will usually be repeated many times for each execution of Phase 1.

Kerberos – Message Formats (Simplified)

1. $C \rightarrow AS: TGS||from||to||N_C$
2. $AS \rightarrow C: \{K_{C,TGS}||C||from||to\}_{K_{AS,TGS}}|| \{K_{C,TGS}||N_C||from||to||TGS\}_{K_{AS,C}}$

   (the first part of message 2 is the ticket granting ticket for the TGS).

3. $C \rightarrow TGS: S||from||to||N'_C || \{K_{C,TGS}||C||from||to\}_{K_{AS,TGS}} || \{C||T_1\}_{K_{C,TGS}}$
4. $TGS \rightarrow C: \{K_{C,S}||C||from||to\}_{K_{TGS,S}} || \{K_{C,S}|| N'_C||from||to||S\}_{K_{C,TGS}}$

   (the first part in message 4 is the ticket for the server S; T is a timestamp).

5. $C \rightarrow S: \{K_{C,S}||C||from||to\}_{K_{TGS,S}} || \{C||T_2\}_{K_{C,S}}$
6. $S \rightarrow C: \{T_2\}_{K_{C,S}}$ (optional)

('from' and 'to': time interval to limit the key validity)

Keys Used in Kerberos

$K_{AS,TGS}$ is a long-term key shared by AS and TGS.

$K_{AS,C}$ is a long-term key shared by AS and C.

$K_{TGS,S}$ is a long-term key shared by TGS and S.

- These keys need to be established in advance.

$K_{C,TGS}$ is a short-term key shared by C and TGS (established by messages 1 and 2).

- This key is transported securely from C to TGS in the ticket granting ticket.

$K_{C,S}$ is a session key shared by C and S (established by messages 3 and 4).

- This key is transported securely from C to S in the ticket.

Tickets in Kerberos

$\{K_{C,TGS}||C||from||to\}_{K_{AS,TGS}}$

- Is the ***ticket granting ticket***.
- Received by C in message 2 and forwarded to TGS in message 3.
- Only TGS can decrypt it to obtain short-term key $K_{C,TGS}$ and validity period from||to. These parameters determine ticket given to C in message 4.

$\{K_{C,S}||C||from||to\}_{K_{TGS,S}}$

- Is the ***ticket***.
- Received by C in message 4 and forwarded to S in message 5.
- Only S can decrypt it to obtain session key $K_{C,S}$ and validity period from||to. These parameters determine access given to C in subsequent session with server S.

These tickets are similar to message 3 in Needham-Schroeder: $\{K \,||\, A\}_{K_{B,T}}$

- Now extended with validity periods for keys.

Entity Authentication in Kerberos

Entity authentications are achieved using a mixture of nonces and timestamps.

Methods are similar to the protocols discussed earlier (and in particular the Needham-Schroeder protocol).

For example: AS is authenticated to C using challenge/response protocol based on encryption, shared key $K_{AS,C}$ and nonce N_C in messages 1 and 2.

C is not authenticated to AS explicitly, but C can only decrypt message 2 if it has the correct key $K_{AS,C}$.

Other authentications: C and TGS; C and S.

Use of Cryptography in Kerberos

Kerberos uses symmetric encryption and MACs.

Specifically, Version 5 (as in RFC 1510) uses DES combined with one of MD4, MD5, or a CRC (not recommended).

Releases 1.2 and higher of Kerberos Version 5 allow triple DES (3DES) in CBC-mode.

Extensions supporting AES included since Kerberos Version 5, release 1.3.2.

Kerberos Issues – 1

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

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

Scalability**: authentication across realms is complicated.**

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

Availability**: need for on-line AS and TGS, trusted by clients not to eavesdrop.**

Kerberos Issues – 2

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

Denial of Service**: potential for DoS attacks on clock service or on AS/TGS?**

Passwords**: in most deployments, the Client-AS long-term key $K_{AS,C}$ is usually based on password entered by user at start of session**

- **Kerberos vulnerable to dictionary attacks – paper by Wu at:** `http://citeseer.nj.nec.com/wu99realworld.html`
- **Ultimately, then, security is dependent on users and the quality of the passwords they can be persuaded to remember.**

Code Vulnerabilities**: many found over the years.**

see http://web.mit.edu/kerberos/www/advisories/

Kerberos Security Advisories

MITKRB5-SA-2007-006
kadmind RPC library buffer overflow, uninitialized pointer
MITKRB5-SA-2007-005
kadmind vulnerable to buffer overflow
MITKRB5-SA-2007-004
kadmind affected by multiple RPC library vulnerabilities
MITKRB5-SA-2007-003
double-free vulnerability in kadmind (via GSS-API library)
MITKRB5-SA-2007-002
KDC, kadmind stack overflow in krb5_klog_syslog
MITKRB5-SA-2007-001
telnetd allows login as arbitrary user
MITKRB5-SA-2006-003
kadmind (via GSS-API mechglue) frees uninitialized pointers
MITKRB5-SA-2006-002
kadmind (via RPC library) calls uninitialized function pointer
MITKRB5-SA-2006-001
multiple local privilege escalation vulnerabilities
MITKRB5-SA-2005-003
double-free in krb5_recvauth
MITKRB5-SA-2005-002
buffer overflow, heap corruption in KDC
MITKRB5-SA-2005-001
Buffer overflows in telnet client
MITKRB5-SA-2004-004
Heap buffer overflow in libkadm5srv
MITKRB5-SA-2004-003
ASN.1 decoder denial-of-service
MITKRB5-SA-2004-002
Double-free vulnerabilities in KDC and libraries
MITKRB5-SA-2004-001
Buffer overrun in srcrpc_to_localname

Windows 2000 Network Authentication

Microsoft have adopted and extended Kerberos for network authentication in Windows 2000.

Supersedes Windows NTLM (unilateral authentication) in NT4.

One 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.**

www.microsoft.com/windows2000/techinfo/howitworks/security/kerberos.asp

Windows 2000 Network Authentication

Second extension:

- use of Kerberos data authorization field (normally empty)
- transports Win2K access privileges in the form of SIDs (Security IDentifiers) derived from Active Directory these are compared to ACLs of remote objects to make access decisions.

Message formats published, but proprietary to Microsoft.

Non-standard extension to Kerberos makes it difficult to interoperate Microsoft and non-Microsoft implementations.

Single Sign On

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.

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

Many vendors currently offer similar SSO/password management products.

Protocols: More Technicalities ☺

Interlock Protocol

Secret Splitting

SKEY

Key Exchange with Public Keys

Full Version of „Authenticated Key Establishment – 3 “:

$A \rightarrow B: \quad PK_A$
$B \rightarrow A: \quad PK_B$
$A \rightarrow B: \quad \{SK\}_{PK_B}$
$B \rightarrow A: \quad \{SK\}_{PK_A}$

If Mallory can modify 1 and 2, he can put himself in the middle by distributing PK_M to Alice and Bob

-> M must be able to intercept and modify traffic; how hard this is depends on the network (cf. Internet, GSM, Broadcast media,...)

Interlock Protocol (Rivest & Shamir)

Let's make it harder for Mallory :

$A \rightarrow B$:	PK_A
$B \rightarrow A$:	PK_B
$A \rightarrow B$:	$\lvert\{M_A\}_{PK_B}\rvert_{1..(n/2)}$
$B \rightarrow A$:	$\lvert\{M_B\}_{PK_A}\rvert_{1..(n/2)}$
$A \rightarrow B$:	$\lvert\{M_A\}_{PK_B}\rvert_{(n/2)+1..n}$
$B \rightarrow A$:	$\lvert\{M_B\}_{PK_A}\rvert_{(n/2)+1..n}$

Interlock protocol:

- A and B want to send messages to each other.
- A sends first half to B.
- B sends first half to A.
- A sends second half to B.
- B sends second half to A.

Since the man-in-the-middle cannot decrypt half of a message, it must pass something on.

- Secure if the attacker cannot intelligibly mimic A or B.

Trent

Trent can make Mallory's life still harder by signing PK_A and PK_B

- M cannot insert his public key, because it is signed by T as belonging to M
- Assume M compromises T:
  - M can only sign new keys
  - M cannot intercept traffic unless he inserts his own faked key

But:

- Trent can be a bottleneck
- Mechanisms like key revocation are needed

Real life: Hybrid Systems

Often used in communication systems based on public keys:

$$A \rightarrow B: \quad S_A\{\{M\}_{SK} || \{SK\}_{PK_B}\}$$

Hybrid system

- can be combined with time stamps, etc.
- can be extended to multiple receipients

Simple Authentication using SKEY

Let f be a one-way (trap door) function and R a random number

- Bob computes and (securely) transmits f(R), $f^2(R)$, ... $f^{n-i}(R)$ to Alice
- Bob remembers only one value: current := $f^n(R)$
- Alice remembers all $f^i(R)$, 1 < i < n-1

Alice's i^{th} authentication:

A → B: $f^{n-i}(R)$

- Bob authenticates A by checking

  $f(f^{n-i}(R))$ = current

  and sets current := $f^{n-i}(R)$

→ Knowing what Bob knows cannot compromise Alice!

Securely Remembering Data: Secret Splitting

Goal: Confidentiality of a message M

Trent generates a random number R, where |R| = |M|

T → A: $M \oplus R$
T → B: R

A and B must cooperate to retrieve M:

$M = M \oplus R \oplus R$

Can be extended to n principals:

T → A: $M \oplus R_1 \oplus R_2 \oplus R_3 \oplus ... \oplus R_n$
T → B: R_1
T → C: R_2
...

Disadvantage: you need all pieces to reconstruct M

More complex solution to this: (n,m) threshold schemes

Zero Knowledge Protocols

"I can't tell you my secret, but I can prove to you that I know the secret."

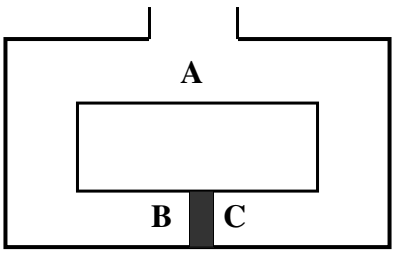
Ali Baba's Cave

Ali Baba had discovered the secret of this strange cave. A password will vanish the secret wall between point B and point C, creating a loop.

To prove his great discovery, Ali Baba invites a television team. He wished not to share his secret password, however.

He would go to either point B or C, and a reporter will randomly request Ali Baba to go to point A via either the left or the right passage.

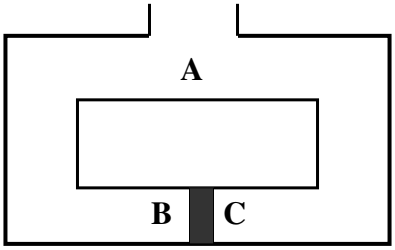
Knowing the secret of the cave, Ali Baba can pass the reporter's test:
He can prove that he knows the password without having to reveal it.



Ali Baba's Cave

However, a fake version of the documentary had been made. It involved an Ali-look-alike performing the same experiment. But without the knowledge of the secret, the actor can only succeed 50% of the time.

However, after editing the film, no one in the world can tell the different between the real and the fake version.



Guillou-Quisquater's Analogy

- By performing a series of verification experiment, it is possible to prove that you know a certain secret without sharing it with anyone.
- Zero-Knowledge Protocols help prevent leaks of any secret information by not directly requesting the secret itself during verification.
- Zero-Knowledge Protocols won't care if you actually know the password or not, as long as you can prove that you know it.
- Faking the proof of knowing the secret is possible, but it has a low probability of success.

Further reading:

J.J. Quisquater and L. Guillou: *How to explain zero-knowledge protocols to your children*", Springer LNCS, 435 (1990), 628-631.

The Bizcard Example

Bob: “Let me in! I have access to this area!”

Alice: “Oh really? What is the secret password?”

Bob: “I can’t tell you my password; it’s a secret.”

Alice: “That’s too bad. Because you cannot get in without telling me your secret password.”

The Bizcard Example (slighty misleading)

The Zero-Knowledge Protocol:

- The password is a positive integer.
- Equipment: A deck of cards

1. While Alice is looking away, Bob counts from the top of the deck until he reaches the card that corresponds to the password. Bob then make an unique mark on one side of that card and turn over all the cards in the deck (without changing their order) and hand the deck to Alice.
2. Now Bob is looking away. Alice also counts from the top of the deck until she reaches the card that corresponds to the password. Alice then make an unique mark on the other side of that card. To conceal the secret, Alice shuffles the deck.
3. If the shuffled deck contains one card having distinct marks on both its sides, then it is possible that both Bob and Alice knows the password. Therefore, Bob is able to prove his knowledge of the password without revealing it to Alice.

The Bizcard Example

The Zero-Knowledge Protocol Phase II:

Alice is not convinced that Bob actually knows the password because the protocol is not perfect: Bob might have guessed the password!

Since the password, *s*, is a positive integer, it has to be limited by a range, *z*, such that: $1 \leq s \leq z$. If Bob doesn't actually know the password, he could have guessed it with probability $1/z$.

The Solution: Alice can request Bob to perform the exact same experiment *k* times so that the probability of Bob correctly guessing the password every time is reduced to $(1/z)^k$.
(If we use a secret that varies with each try!)

When $(1/z)^k$ is small enough, that is, when the probability of Bob actually knowing the password is high enough, Alice may grant Bob access to his account without worrying that he might be an imposter.

(Dis)Advantages

Advantages of Zero-Knowledge Protocols:

- Not requiring the revelation of one's secret.
- Does not involve complex encryption methods.

Disadvantages of Zero-Knowledge Protocols:

- Limited:
  Secret must be numerical, otherwise a translation is needed.
- Lengthy:
  Each computation requires a certain amount of running time.
- Imperfect:
  Mallory can still intercept the transmission (i.e. messages to the Verifier or the Prover might be modified or destroyed).

Properties of ZKPs

Completeness:

- The Verifier will always accept a proof from the Prover, given that they both follows the correct protocol.

Soundness:

- The Verifier will not accept any "incorrect" proof from the Prover, given that the Verifier follows the correct protocol.

Zero-Knowledge:

- During the whole "proving" process, the Verifier will learn nothing about the Prover's secret, nor will she be able to prove that secret to any other party.

Zero Knowledge Proofs of Identity

The Chess Grandmaster Problem

- Anyone can defeat or beat a grandmaster in Chess:
  - Choose a second grandmaster and act as a man-in-the-middle

Problem with ZKP:

- Mallory can act as a man-in-the-middle and pass Alice's answers to Bob
- Can be „fixed" with timestamps, where each answer must be given at an exact time: no time left to pass messages
  - ... often of little value in practice

Probabilistic Proofs

Proofs based on interactive protocols are probabilistic.

- There is generally a chance that the Verifier will reject some valid proofs or accept invalid ones.

We can define a probabilistic proof system for L as an interactive protocol P such that:

- For all x in the assertion language P(x) halts in polynomial time.
  - The Efficiency property.
- If x is in L, then P(x) accepts with probability at least α.
  - The Completeness property.
- If y is not in L, then P(x) accepts with probability at most β.
  - The Soundness property
- Where $1 >= \alpha > \beta >= 0$
  - We can repeat such a proof multiple times to make the chance of false positive or negative negligible.

Lessons Learned?

Designing protocols is easy.

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, …)

– use these rather than rolling your own!

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

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

Further Reading

Ross Anderson & Roger Needham: Programming Satan's Computer, http://www.cl.cam.ac.uk/~rja14/#Protocols

http://www.conceptlabs.co.uk/alicebob.html

- Now there are hundreds of papers written about Alice and Bob. Over the years Alice and Bob have tried to defraud insurance companies, they have played poker for high stakes by mail, and they have exchanged secret messages over tapped telephones.

  ...