+

Kerberos | PGP

INGI2347: COMPUTER SYSTEM SECURITY (Spring 2014)

Marco Canini

Lecture slides adapted from UCL INGI2347 by Gildas Avoine
Reproduced with permission

+

Announcements

- Final exam on 11 Jun at 8:30 in BARB91
- Lecture 12 (Cloud computing security) will not be part of the exam

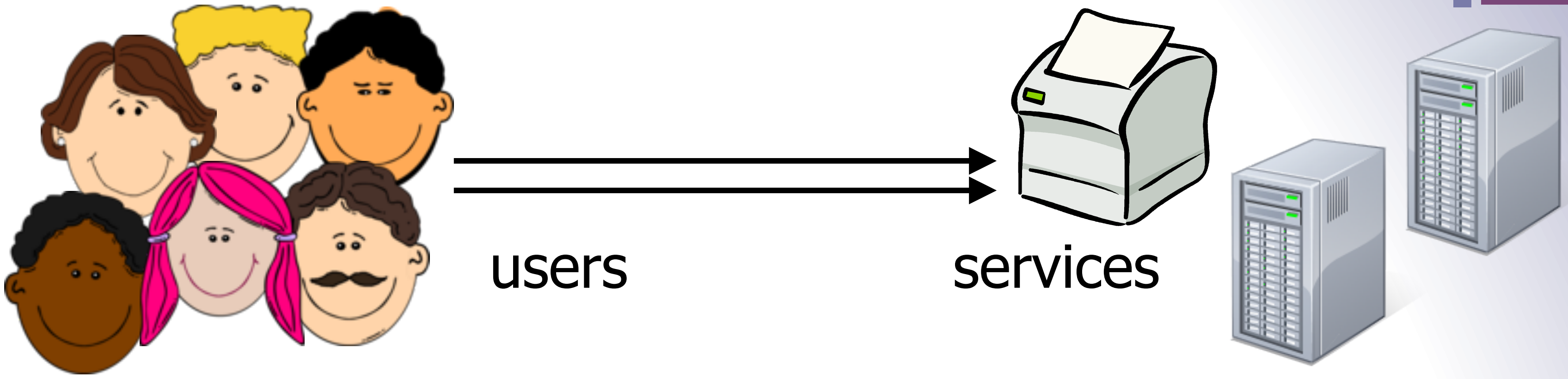
Plan for today

Lecture 13

- Kerberos
- Pretty Good Privacy (PGP)

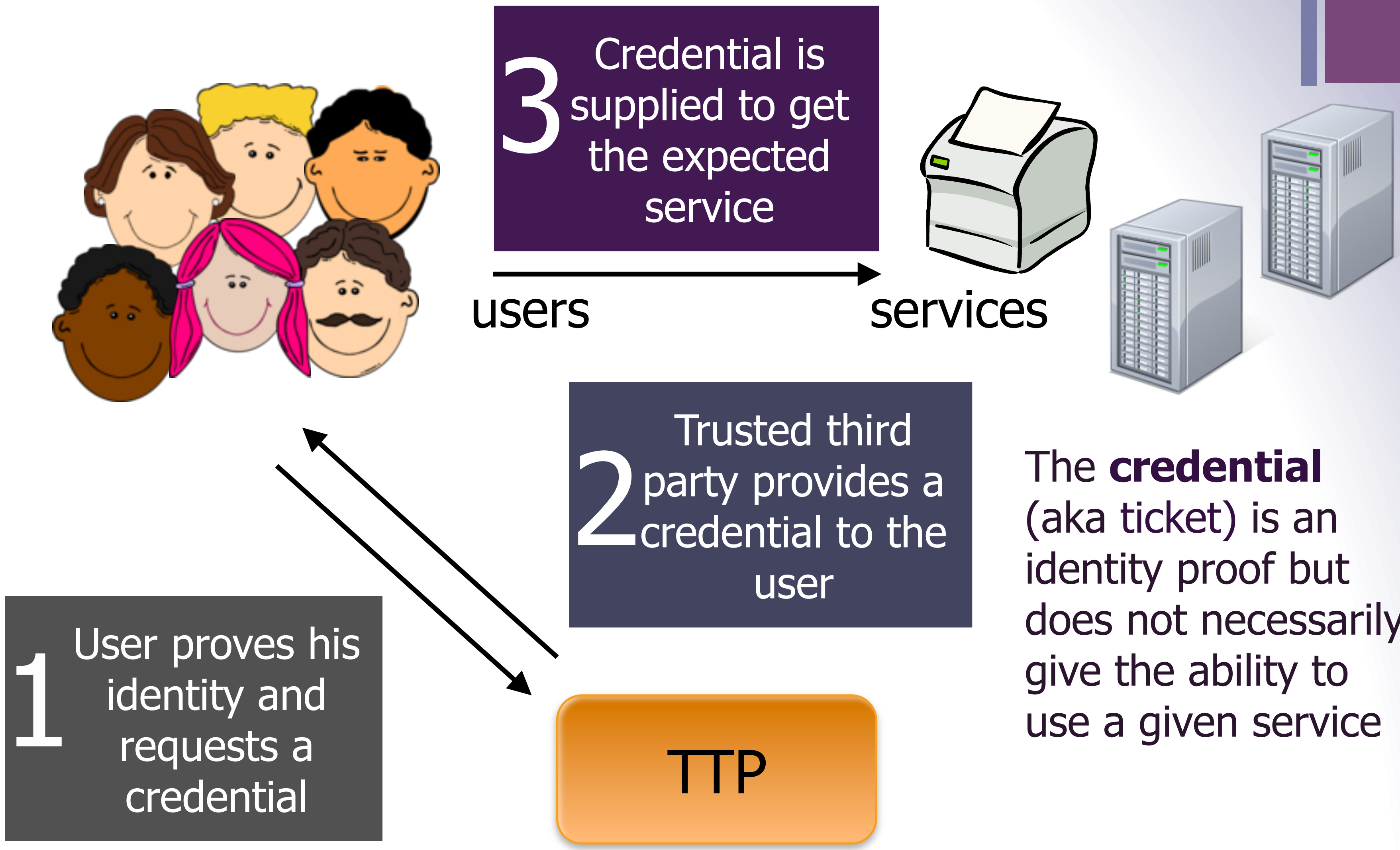
NEXT

Many-to-many authentication



- How do users prove their identities when requesting services from servers on the network?
- Solution: every server knows every user's password
  - **Insecure**: break into one server may compromise all users
  - **Inefficient**: passwords must be changed on every servers
  - **Inconvenient**: passwords must be typed for each request

Server-aided authentication



1 User proves his identity and requests a credential

2 Trusted third party provides a credential to the user

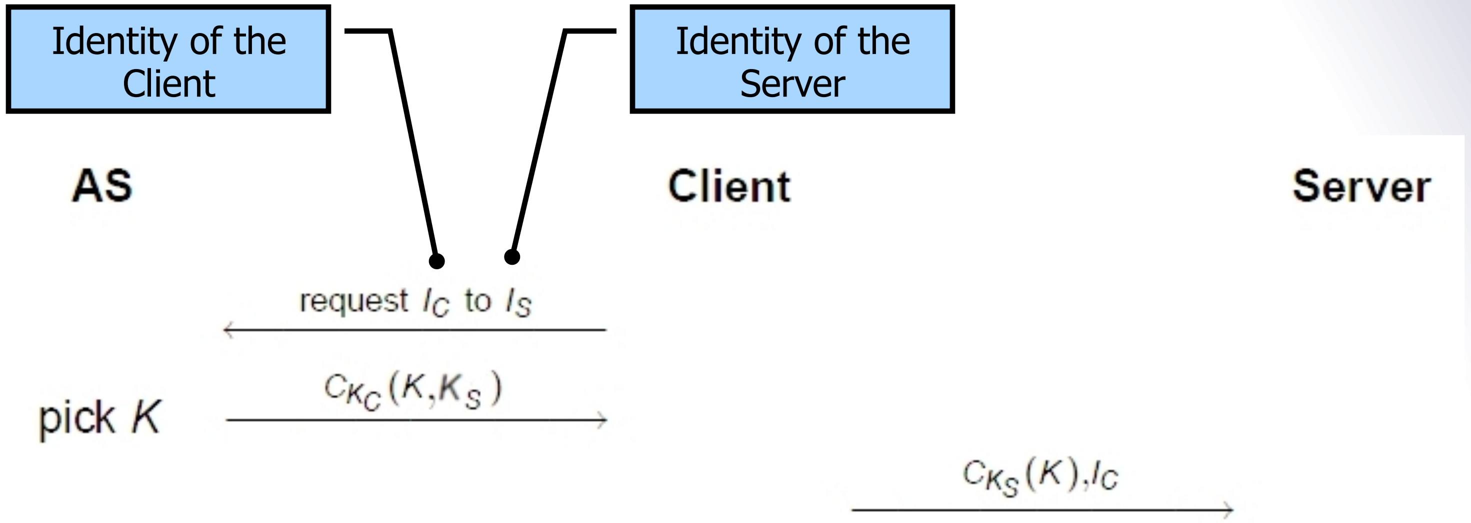
3 Credential is supplied to get the expected service

The **credential** (aka ticket) is an identity proof but does not necessarily give the ability to use a given service

Server-aided authentication

- Hypotheses:
  - There is an online trusted authentication server (AS)
  - AS shares K_C with client C
  - AS shared K_S with server S
- Goal:
  - To help C and S share a session key **K**

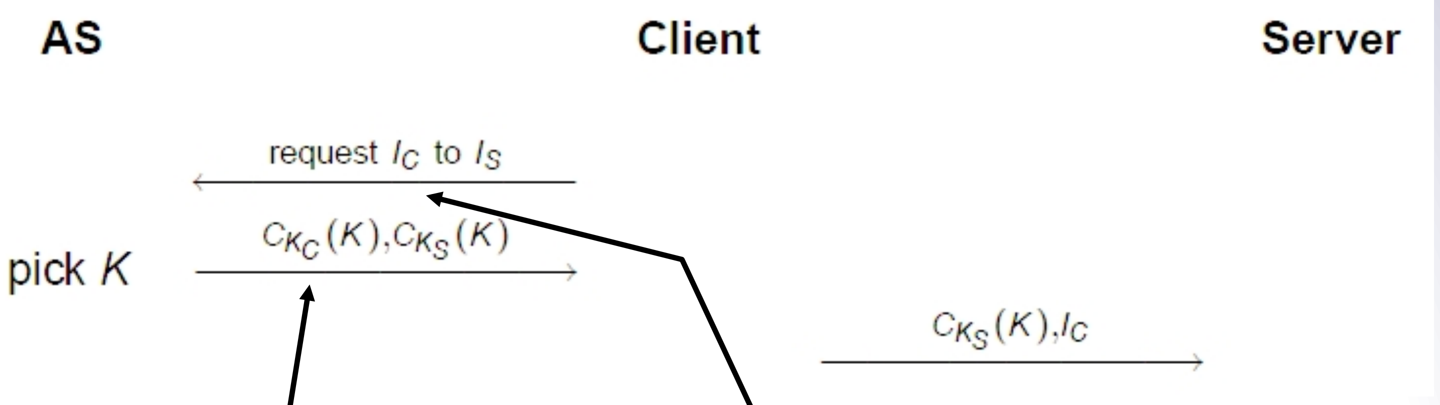
Very Weak Example



The client can give the server's key to other clients

Weak Example

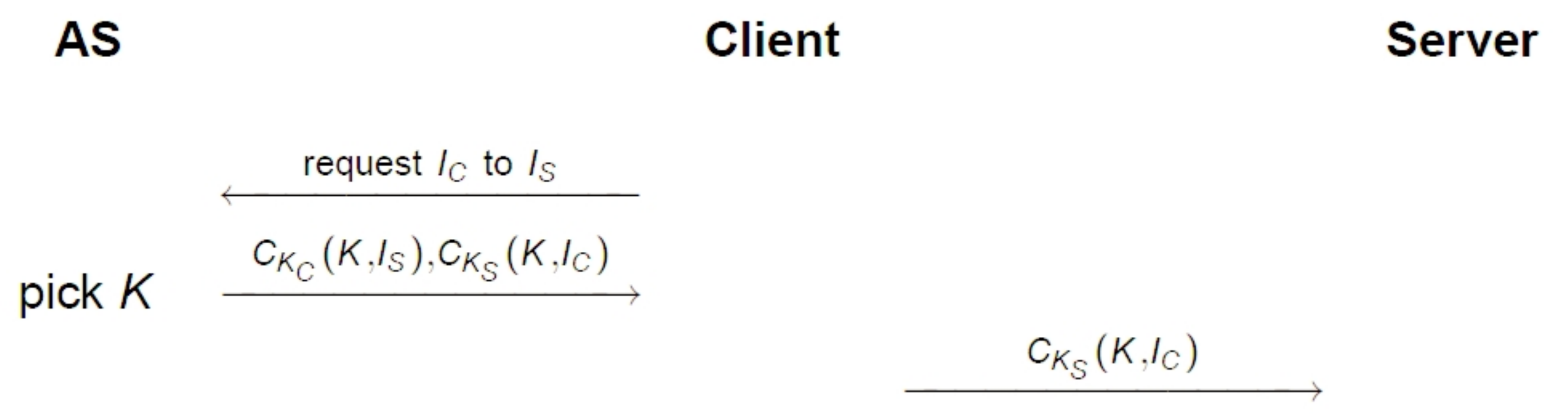
A solution consists in not revealing the server's key: AS encrypts itself the session key K with the server's key. "sealed envelop"



AS uses K_A instead of K_C

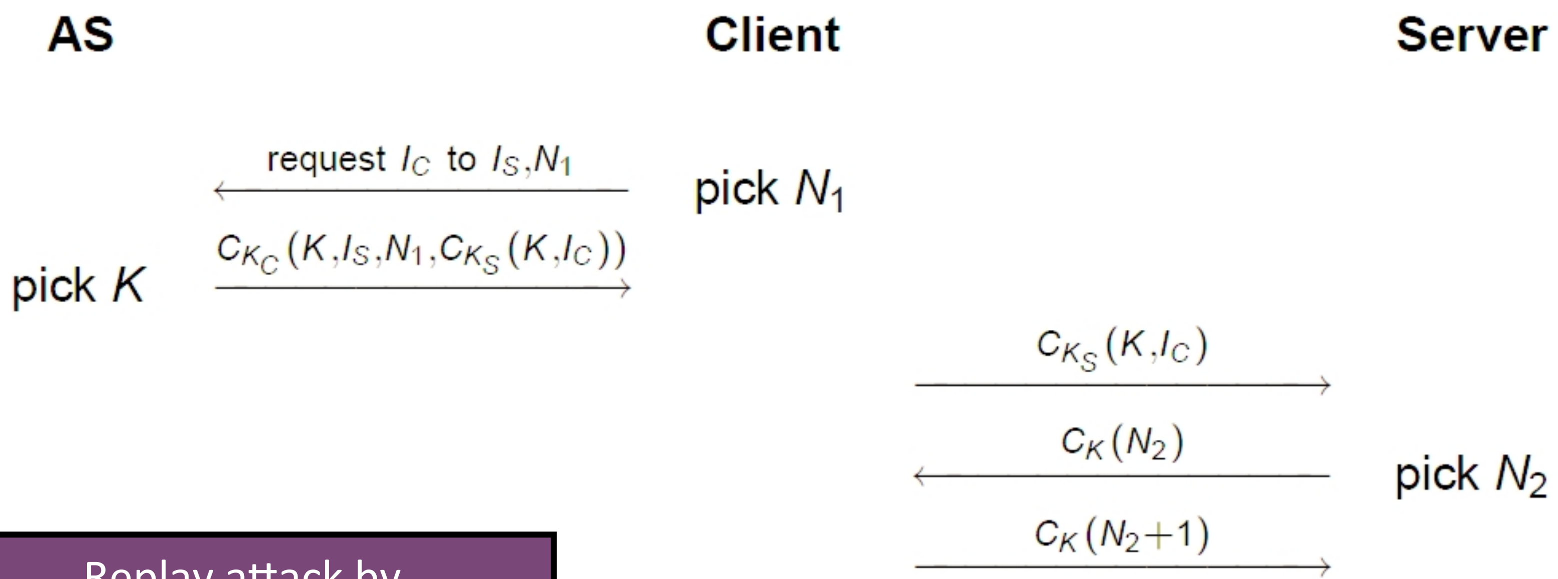
An attacker can replace I_C by I_A

Still Weak Example



Replay attack by impersonating **AS** if K is compromised, due to careless users: no means to be sure that K is fresh

Needham Schroeder (1978)



Replay attack by impersonating **C** if K is compromised, due to careless users: no means to be sure that K is fresh

N1 is a nonce, a random value used only once

+

Kerberos

Kerberos V

- The name Kerberos comes from Greek mythology
  - It is the three-headed dog that guarded Hades' entrance
- Created at the MIT, free of charge
  - Kerberos 4 (1988), obsolete
  - Kerberos 5 (1993), RFC 1510, then RFC 4120 (2005)
- Deployed:
  - Initially on Unix systems
  - Used in many commercial products, e.g., Windows since 2K
  - Based on symmetric-key cryptography

Kerberos V

- Once logged into a system, you can access remote resources without inputing username and password anymore

- Kerberos software on the workstation will finish the authentication automatically on your behalf

Kerberos Elements

- C = Client | S = Server
- AS = Authentication server
  - a.k.a. KDC = Key Distribution Center
- TGS = Ticket Granting Server

1- Request a Ticket Granting Ticket

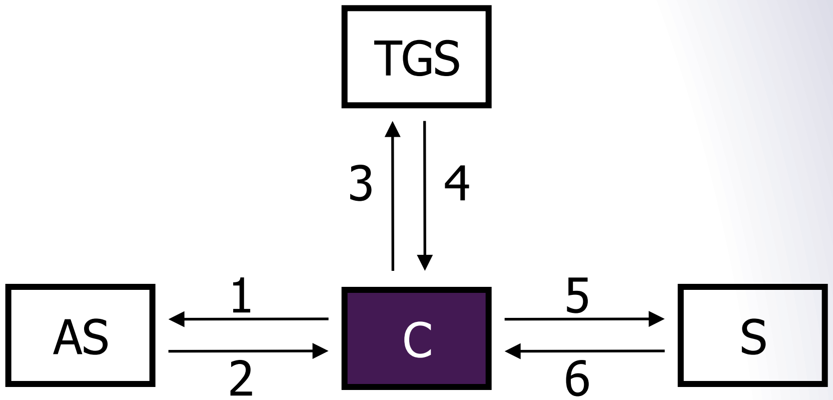
2- Provide a Ticket Granting Ticket

3- Request a Ticket for a given service

4- Provide a Ticket for a given service

5- Forward the Ticket

6- Provide a service



Tickets and Authenticator

- To access a service, the client must have a ticket for that service
- Client can get this ticket from the TGS
- To access the TGS, the client must have a Ticket Granting Ticket
- Client can get this ticket from the Authentication Server
- The client shows a ticket + an authenticator

Tickets, Authenticator

- The ticket contains:
  - $\mathbf{I_C}$: the client's identity
  - **v**: validity period
  - $\mathbf{K_{C,S}}$: symmetric session key to be used between client and server
  - Others: Flags, IP address, times, etc.

- It is encrypted with the key of the server $\mathbf{K_S}$

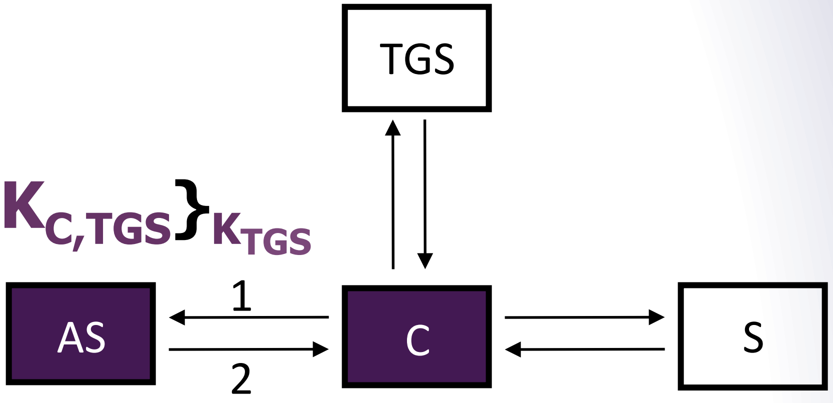
- The authenticator is just the client's identity and a timestamp encrypted with the session key

Between C and AS

- Firstly, C must be authenticated by AS to have access to TGS
- C sends his identity and the identity of the TGS he wants to access to
- AS replies with a Ticket Granting Ticket (TGT) encrypted with TGS's key and a session key encrypted with C's key

(1) I_C, I_{TGS}, N

(2) $\{I_{TGS}, N, K_{C,TGS}\}_{K_C}, \{I_C, v, K_{C,TGS}\}_{K_{TGS}}$



+

User & Service Authentication

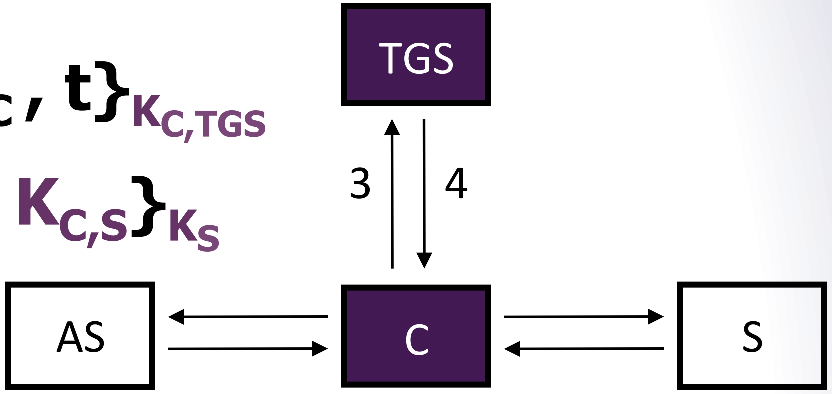
- The user types his username and password on his machine
- The client applies a one-way function (in practice a hash function) on the password in order to get the cryptographic key $\mathbf{K_c}$
- Server's keys are random bit-strings

Between C and TGS

- Client sends the TGT as well as an authenticator to the TGS
  - Recall that the ticket contains the session key $\mathbf{K_{C,TGS}}$
- TGS uses the session key to verify the authenticator
- TGS knows whether C is authorized to access the server S
- TGS delivers a ticket to access the service

(3) I_S, N', $\{I_C, v, K_{C,TGS}\}_{K_{TGS}}$ $\{I_C, t\}_{K_{C,TGS}}$

(4) $\{I_S, N', K_{C,S}\}_{K_{C,TGS}}$, $\{I_C, v, K_{C,S}\}_{K_S}$

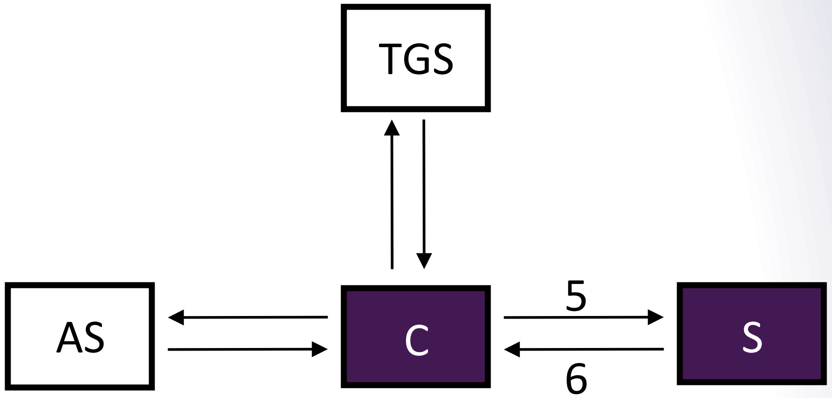


Between C and S

- Service ticket again contains the client's identity, a validity period and the session key to be used between client and server
- Client has also received a copy of the session key, encrypted with the previous session key
- It sends an authenticator and the ticket to the server

(5) $\{I_C, v, K_{C,S}\}_{K_S}, \{I_C, t\}_{K_{C,S}}$

(6) $\{t\}_{K_{C,S}}$



Discussion

- It is the client's responsibility to store its credential (the tickets); the servers are stateless
- The authentication server is accessed only once during the ticket validity (typically 8 hours)
- Clients can access services with their tickets even if the authentication server is down
- Once a client is authenticated, its ticket cannot be revoked

Ski Pass Analogy

- You get a three-day ski pass (TGT) from your travel agency against a proof of identity (and money…)
- The three-day ski pass (TGT) can be used at four different resorts
- You show the pass at whichever resort you decide to go (until it expires), and you receive a lift ticket (ST) for that resort
- Once you have the lift ticket (ST), you can ski all you want at that resort (until it expires)
- If you go to another resort later, you once again show the three-day ski pass (TGT), and you get another lift ticket (ST) for the new resort

+

Pretty Good Privacy (PGP)

PGP History

- PGP = Pretty Good Privacy
- Several flavors: PGP, PGPi, GPG

PGP

- Published by Philip Zimmermann in 1991
- Portable software initially containing classical algorithms MD5, IDEA, RSA
- First software allowing anybody to completely protect their documents and messages
- 3 years of enquiry and harassment by the American government
  - Patented algorithms (RSA patented in the US until 2000)
  - Suspicion of violating export regulations

PGP History

1996-97:

- Selling of PGP Inc. to McAffee (Network Associates)
  - Code no longer public
- During the 39th IETF meeting at Munich, Zimmermann and Callas requested the IETF to setup a working group on the standardization of PGP (OpenPGP [RFC1991, Aug 96], [RFC2440, Nov 98], [RFC4880, Nov 07])
- Richard Stallman at the Individual-Network Betriebstagung at Aachen requested the European hackers to implement public key software (US citizens were not allowed to do so outside us)

2001:

- Zimmermann leaves Network Associates
- Network Associates abandons PGP

PGP History

2002:

- PGP Corporation is created, buys back PGP rights www.pgp.com
- Code is again public
- Free trial version
- Basic functionalities remain available after 30 days, but not the additional functionalities, e.g., disk encryption
- Complete system compliant with OpenPGP

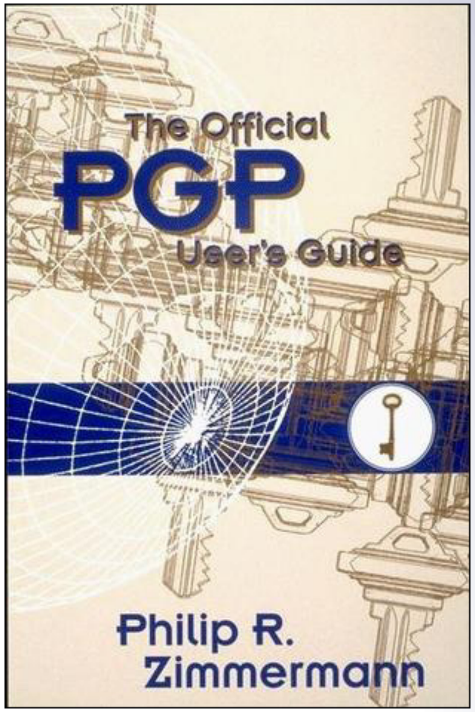
2010:

- Symantec acquired PGP

PGP History

PGPi

- Developed by Ståle S. Ytteborg (Norway) to counter the US export regulations
- Maintained from 1997 to 2000
- Obtained from the printed source code of PGP
- MIT Press thus published a book with the PGP source code
- www.pgpi.org



PGP History

GPG = GnuPG = GNU Privacy Guard

- GnuPG is the GNU GPL version of PGP www.gnupg.org
- Initially, used ElGamal and Blowfish instead of RSA and IDEA
- Follow the Open PGP Standard
- Version 0.0.0 released in December 1997

- GUI Frontends:
  - http://www.gnupg.org/related_software/frontends.en.html

Basics

PGP Features

- Signature

- Encryption
  - Hybrid crypto: combine symmetric and public-key crypto
  - Session key is symmetric; encrypt session key with public-key of recipient

- Key management
  - What is called a PGP key is actually a PGP certificate
  - Web of trust

Example

```
This is an example of signed message

-----BEGIN PGP SIGNATURE-----
```

```
iQIcBAEBCgAGBQJTcWJaAAoJEChyd2euJIo/aYYP/0Vl/+u5zNkFw9lgvCd4UYdu
88aTImx+KmP8loFnu0Q6EC8UCuYCd8q/CHNPVq9k+pBE3Szolt6L3EIO6hDwRjJn
lnODZVoAWBgy5S5+BEgTA6OI3ixsmySacjkfYKbSprgLCKRklgesVl9Lo+5/ZTXJ
gQRhqePkYEmsfMKnTmLi9jiS/TqfXBcKOiuZ2Y/ihhNULIP4mnIDKw7k2AI8d27/
rAV2uMEi2XKDwxn9ziJ31yAM6IUhKvEKFwAjHf63rETZM3QrlgHaG/U128S5pqzS
JCkXFMhXnyCVRXmVDaoq9drzWXJ7EU8YHYDZnw6cuuYXPkGQC83T8XM+ZDIXFeQz
o0uFXcKUPyO+Ns6D2HrPKv+yxi8PbmBTOZs8nKIj843BzWFr3etnR19N1f/+zV+X
VMaNRW/i67Of8uD4dJlkA8PYDgBmglBn8oRiU0L5bqOWoXJFJKXQiYz62lZvtPwS
PBDAfM2NfGkdBV4ypOoqydTzwhd8ZO26PICKAKFhW+AfEeQu7a7tOD0+m/3L74Mf
ljbTTalyctgTY/slDiP/bHS8NCgIIhvjsJYdfrMCuc+t29bh5FwMnyemU07Ynqa2
vo4L/Jq1qJ3Cy2h+kyW4MZlh6ADauacbHH1pVLKvHOnH5mT4FsP0rsI/F73oZSN2
RQZwQdrjHsIihP02ERCX
=FyhH
```

```
-----END PGP SIGNATURE-----
```

Symmetric Encryption [RFC4880]

All of them seem to be secure.

- TDES [Mandatory]
  - Slow. Considered to be secure
- IDEA
  - Patented until 2010. Seem to be secure, resisted to all cryptanalysis for 17 years…
- CAST5 (128 bit-key) [should impl. CAST5]
  - Less studied than the other algorithms
- Blowfish (128 bit-key)
  - Less studied than the other algorithms
- Twofish (256 bit-key) (AES contest top-5 finalists)
  - Rather new
- AES (128/192/256 bit-key) [should impl. AES128]
  - The standard since 2000

Public-Key Encryption [RFC4880]

- RSA
- ElGamal [Mandatory]

(Public-Key) Signature [RFC4880]

- RSA
- DSA [Mandatory]
- ElGamal no longer recommended for signature
  - Attack by Phong Nguyen (2003) when ElGamal keys used for both encryption and signature.
  - *"[…] We show that as soon as one (GPG-generated) ElGamal signature of an arbitrary message is released, one can recover the signer's private key in less than a second on a PC. As a consequence, ElGamal signatures and the so-called ElGamal sign+encrypt keys have recently been removed from GPG"* (Nguyen, 2003)
  - The flaw was exploitable during 4 years…

Hash Functions [RFC4880]

- MD5
  - Deprecated
- SHA-1 [Mandatory]
  - Should be avoided
- SHA-224/256/384/512
  - Seem Ok
- RIPEMD-160
  - Seem Ok
- Tiger
  - Seem Ok

Protection of the Private Key

- The private key cannot be memorized by the user
- **How can we protect the private key?**
- Stored on the hard drive
  - Encrypted with a password (no means to access it without the user's collaboration)
  - Once decrypted, it is in the computer's memory (dangerous)
- Stored on a smart card
  - Access to the card is protected by a password
  - The key never leaves the card, it's the data that transits through the card to get encrypted, decrypted or signed
- The passphrase must be as strong as the key (i.e., same entropy at least)

Key Size [Lenstra, Verheul, 01]

sym. key (bits)	public key (bits)
71	1024
80	1536
87	2048
99	3072

Help choosing an appropriate key size:
http://www.keylength.com/en/1/

+

Public-key Validity

Getting the Recipient's Key

- How to be sure that the key we use to encrypt a message is the correct one?

- Directory
  - Who put the key into the directory?
  - Fake identity associated to the key?
  - Is the directory a legitimate one?

- Face to face, check the ID, check the hash of the key, sign the key

- Certificates

Certificates

- Peer-to-peer
- Users trust some other users
- One or several signatures on each certificate

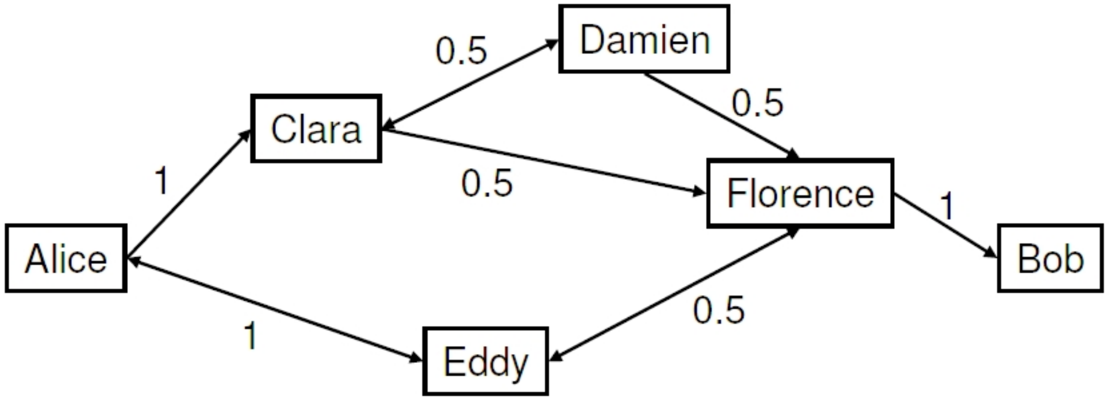
Public-key Distribution

Validity and Trust in PGP

- Two important notions in PGP
- Validity: I know that this key belongs to Bob
- Trust: I know that Bob does not sign keys arbitrarily
- When we sign a key, we declare its validity

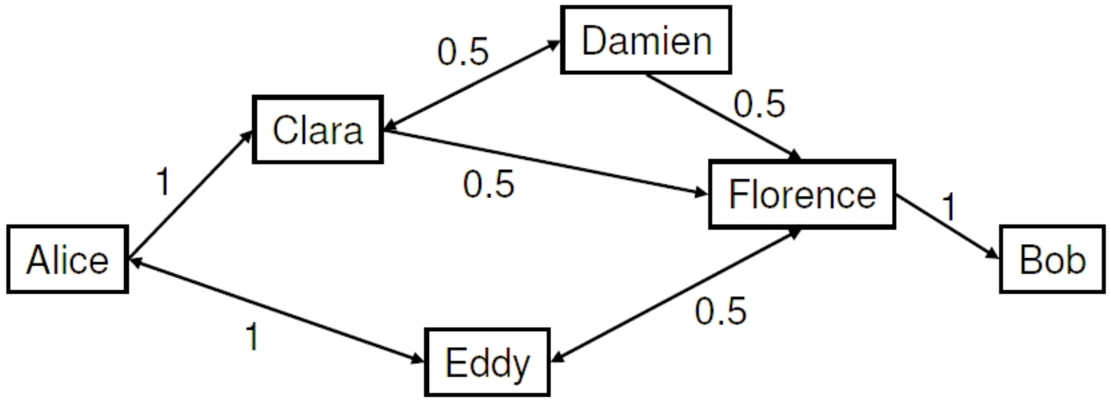
Validity and Trust in PGP

- We can also declare a full or partial trust
- A key is valid if the sum of the partial trusts of its valid signatures is at least 1



The Web of Trust

- Clara and Eddy are valid since Alice has signed them
- Alice has full trust in Clara and Eddy:
  - Damien, Florence, and Eddy are valid
- Clara and Eddy each have a partial trust in Florence:
  - Alice trusts Florence and Bob is valid



Key Signing Party

- Each participant’s public key is published in advance and downloaded by everybody
- Each participant identifies himself (with passport) and reads aloud his key fingerprint
- Everybody signs that key and uploads it on a key servers

Key Publication

- Several PGP key servers exist across the world
  - http://pgp.mit.edu/
- They contain keys of all PGP users that want to publish their key
- If Alice is sure that the key associated to Clara belongs to Clara, she can sign Clara's key and re-submit it to the servers
- If Eddy trusts Alice, he can accept Clara's key

Public-key Revocation

Key Revocation

- How can we revoke a key published on a server?
- Servers are replicated: withdrawing a key is useless because another server will duplicate it again
- How can we prove that we are allowed to revoke a key if we lost it?
- We generate a key revocation certificate when we generate the key
- We put a validity deadline to the key when we generate it

Hope you enjoyed INGI2347!