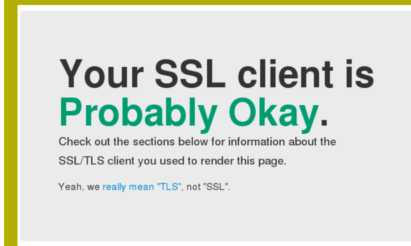


GnuPG

UCL
Université catholique de Louvain

Certificates | SSL/TLS | PGP

INGI2347: COMPUTER SYSTEM SECURITY (Spring 2015)

Marco Canini | Guest lecturer: Xavier Carpent

Lecture slides adapted from UCL INGI2347 by Gildas Avoine, Upenn CSE331 by Steve Zdancewic.
Reproduced with permission

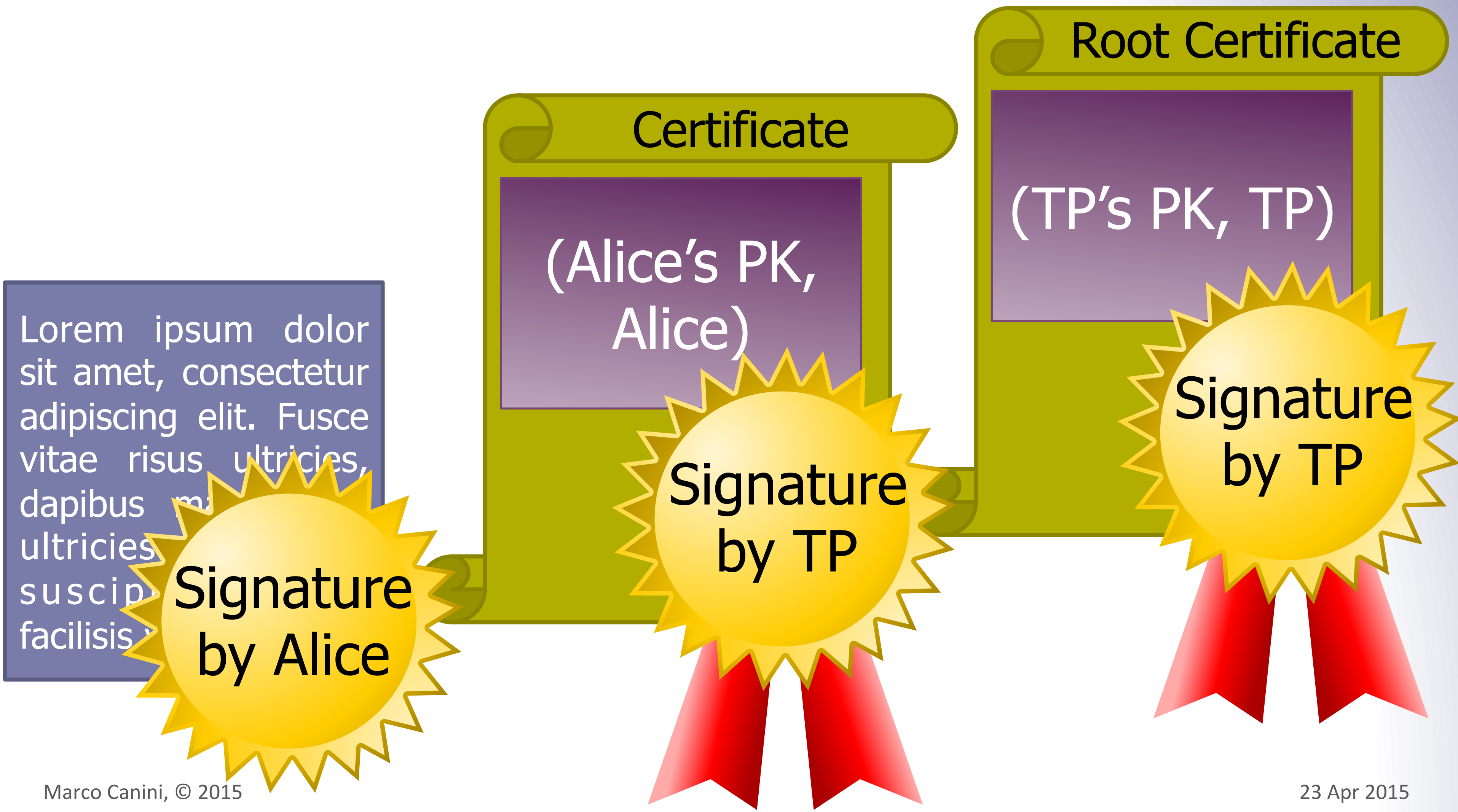
Certificates

What is a certificate?

Certificate's goal is to **link** a public key (PK) with its owner

- The pair (PK, owner) is signed by a trusted party (TP)
- The TP is named **Certification Authority** (CA)
- To check the signature, the CA's PK is needed
  - **Root certificate**: the pair (CA's PK, CA) is self-signed
  - The authenticity of the root certificate is fundamental (included in browsers)

Illustration



X.509 Certificates

X.509

- Standard from International Telecommunication Union (ITU), 1988
- Also IETF RFC-2459 (and updates)

Three required fields:

- TBS Certificate (TBS = "To Be Signed")
  - The useful payload of the certificate
- CA signature algorithm
  - Identifier for the crypto algorithm used by the CA to sign this certificate
- CA signature value
  - Signature of the certificate by the CA

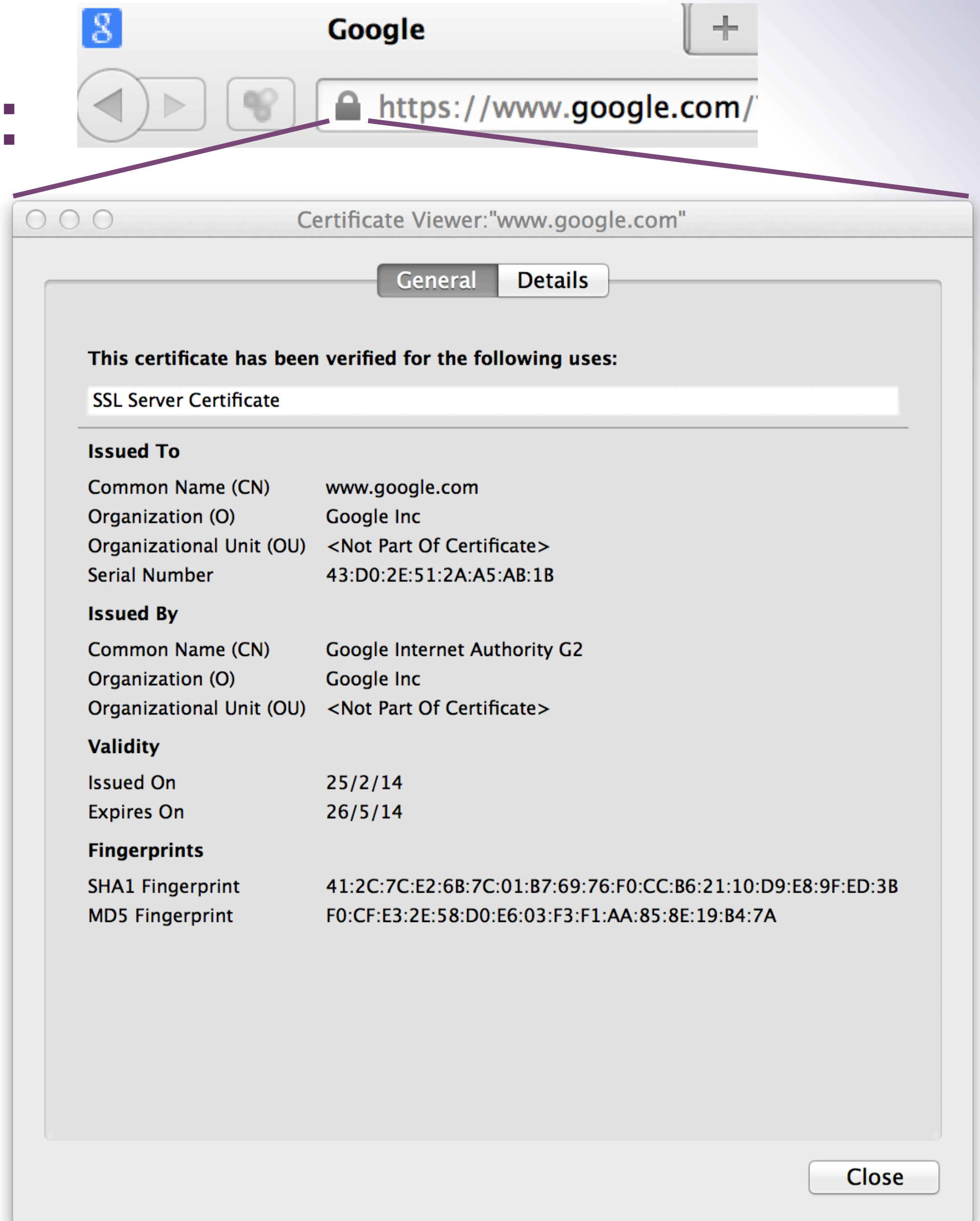
X.509 TBS

- Serial number
  - Unique number assigned by the CA to the certificate
- Issuer field
  - Identifies the entity who has signed and issued the certificate
- Subject
  - Identifies the entity associated with the public key
    - O: organization, C: country, OU: organization unit, CN: common name, ST: state, L: city, etc. no IP address

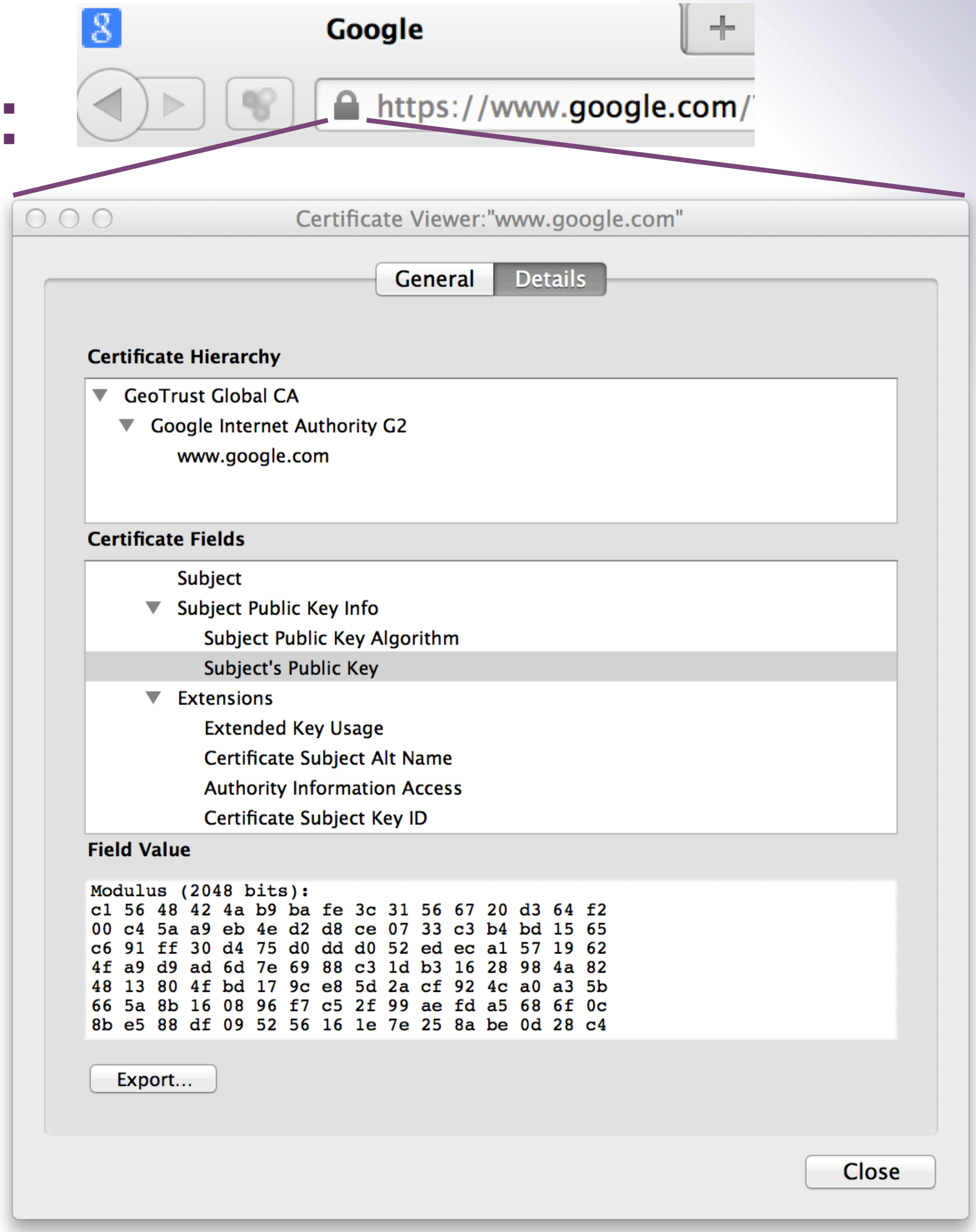
X.509 TBS (Continued)

- Validity
  - Not before
  - Not after
- Subject Public Key Info
  - Public key
  - Identifies the algorithm with which the key is used
    - e.g., RSA, DSA, or DH
- Etc.

Example:


Google
https://www.google.com/
Certificate Viewer:"www.google.com"
General
Details
This certificate has been verified for the following uses:
SSL Server Certificate
Issued To
Common Name (CN) www.google.com
Organization (O) Google Inc
Organizational Unit (OU) <Not Part Of Certificate>
Serial Number 43:D0:2E:51:2A:A5:AB:1B
Issued By
Common Name (CN) Google Internet Authority G2
Organization (O) Google Inc
Organizational Unit (OU) <Not Part Of Certificate>
Validity
Issued On 25/2/14
Expires On 26/5/14
Fingerprints
SHA1 Fingerprint 41:2C:7C:E2:6B:7C:01:B7:69:76:F0:CC:B6:21:10:D9:E8:9F:ED:3B
MD5 Fingerprint F0:CF:E3:2E:58:D0:E6:03:F3:F1:AA:85:8E:19:B4:7A
Close

+ Example:


Google
https://www.google.com/
Certificate Viewer:"www.google.com"
General
Details
Certificate Hierarchy
GeoTrust Global CA
Google Internet Authority G2
www.google.com
Certificate Fields
Subject
Subject Public Key Info
Subject Public Key Algorithm
Subject's Public Key
Extensions
Extended Key Usage
Certificate Subject Alt Name
Authority Information Access
Certificate Subject Key ID
Field Value
Modulus (2048 bits):
c1 56 48 42 4a b9 ba fe 3c 31 56 67 20 d3 64 f2
00 c4 5a a9 eb 4e d2 d8 ce 07 33 c3 b4 bd 15 65
c6 91 ff 30 d4 75 d0 dd d0 52 ed ec a1 57 19 62
4f a9 d9 ad 6d 7e 69 88 c3 1d b3 16 28 98 4a 82
48 13 80 4f bd 17 9c e8 5d 2a cf 92 4c a0 a3 5b
66 5a 8b 16 08 96 f7 c5 2f 99 ae fd a5 68 6f 0c
8b e5 88 df 09 52 56 16 1e 7e 25 8a be 0d 28 c4
Export...
Close

Working with Certificates

Certificate Authorities

- Issuers of certificates found on web servers

CA	Count [%]
GeoTrust	25.19
GoDaddy.com	13.65
Verisign	13.09
Thawte	9.79
Comodo Limited	7.12
Unknown	2.40
DigiCert	2.39
Network Solutions LLC	2.09
Comodo CA Limited	1.77
GlobalSign	1.64

NOTE: GeoTrust, Verisign, and Thawte are the same group

Source: https://secure1.securityspace.com/es/s survey/data/man.201002/casurvey.html (Feb 2010)

How to obtain a certificate

- Applicant registers with a CA
- CA (physically) authenticates the applicant
- CA asks applicant to generate public/private keys
- CA creates a certificate with the applicant's identity, PK, expiration date, etc., and the CA's signature
- CA provides a copy of its own PK to applicant

Registration Authority (RA)

- CA can delegate the registration of an applicant to the **registration authority** (RA)
- RA does not have CA's private key
- CA trusts the RA to authenticate the applicants
- After applicant is authenticated, applicant generates a pair of keys and sends the public key to the CA to create the certificate
- Technically RA sends a signed Certificate Signing Request (CSR) to the CA

CSR in practice

- Generate a 1024-RSA key-pair
  - openssl genrsa 1024 > mykey.key
- Generate a CSR
  - openssl req -new -key mykey.key -out myreq.csr
- Verify a CSR
  - openssl req [-text] [-noout] -verify -in myreq.csr
- Online checkers
  - http://support.ecenica.com/ssl-certificates/csr-checker/
  - https://ssl-tools.verisign.com/checker/

+

Certificate without CA

- Everyone can self-sign a certificate
- Distribute the certificate through an authenticated channel
- Makes sense in enterprise intranet
- Not really for public-facing services
- Rather get a free certificate…

Certificates in practice

- Generate a certificate
  - openssl x509 -req -in myreq.csr -signkey mykey.key -out mycert.crt
- View a certificate
  - openssl x509 -text -in mycert.crt
- Verify a certificate
  - openssl verify mycert.crt

Key escrowing

Keys are held in escrow so that, under certain circumstances, an authorized third party may gain access to those keys

Example:

- A company can provide two key pairs and certificates to each of is employees
  - One for signing | One for encrypting
- CA escrows a copy of the private encryption key
- Only employees can sign, but company can decrypt

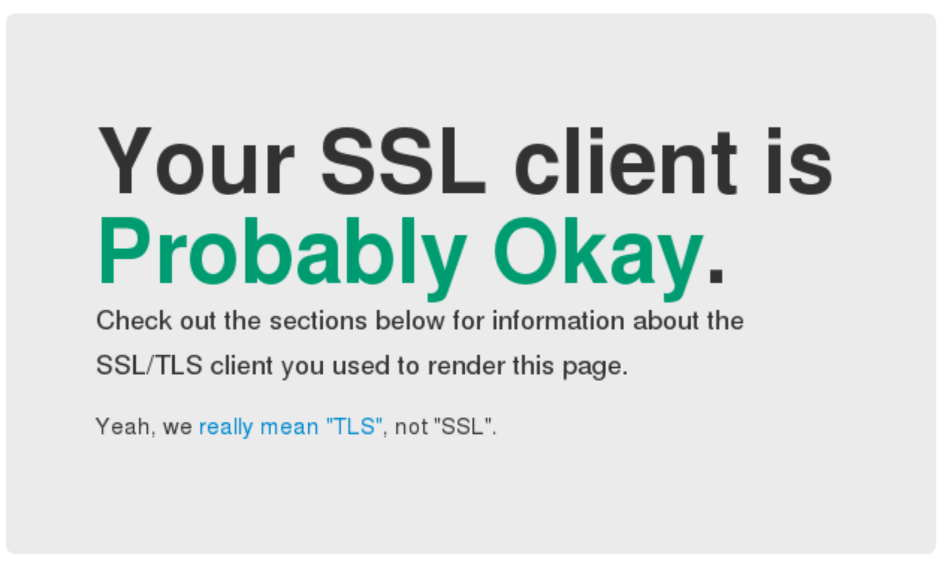
Verifying a certificate

1. Verify the **certification path**
   - Performed locally
   - Delegated to a server: SCVP (Server-based Certificate Validation Protocol)
2. Verify the **validity period**
3. Verify that the certificate is **not revoked**
   - Performed locally: CRL (certificate revocation lists)
   - Delegated to a server: OCSP (Online Certificate Status Protocol)
   - Supported by all major browsers … but not implemented consistently
     - What is the risk?

Issues w/ compromised certificates

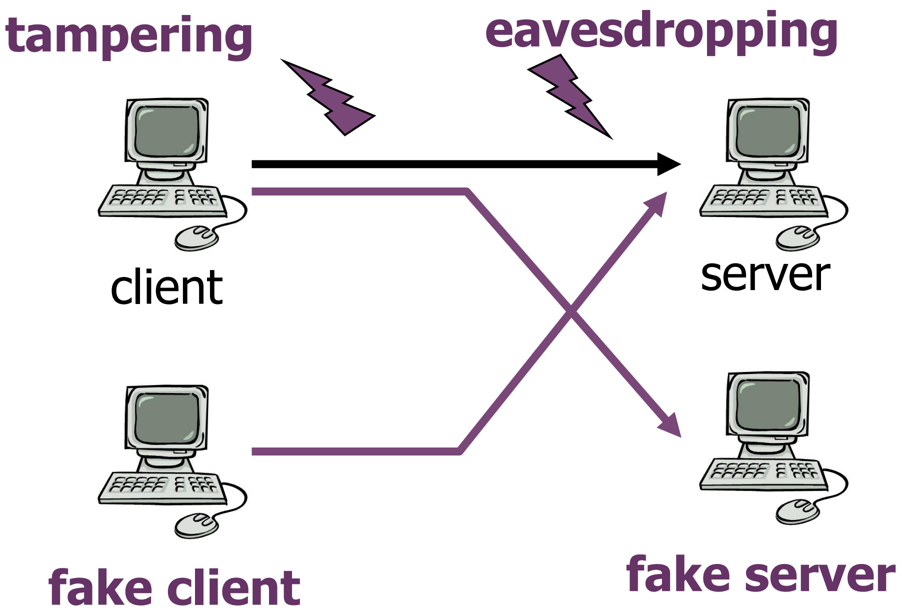
- A certificate might become compromised
  - For example, see Heartbleed bug: http://heartbleed.com/
- Even if the compromised certificate is revoked and replaced with a new one, a secure site could still be vulnerable
- Problem: browsers support revocation checking in different, inconsistent ways
  - e.g., when OCSP is not available only IE and Opera will check a URL pointing to a CRL in individual certificates
  - By default OCSP is disabled in Chrome (in part due to privacy concerns)
    - Chrome uses its own updating mechanism and is intended for most important certificates only
- Finally, CRLs are retrieved less frequently by browsers

Read more: http://news.netcraft.com/archives/2014/04/24/certificate-revocation-why-browsers-remain-affected-by-heartbleed.html


Your SSL client is Probably Okay.
Check out the sections below for information about the SSL/TLS client you used to render this page.
Yeah, we really mean "TLS", not "SSL".

SSL / TLS

SSL Primer



- Authentication of server based on public key
- Trusted third party: certification authority (CA)

Secure Sockets Layer (SSL)

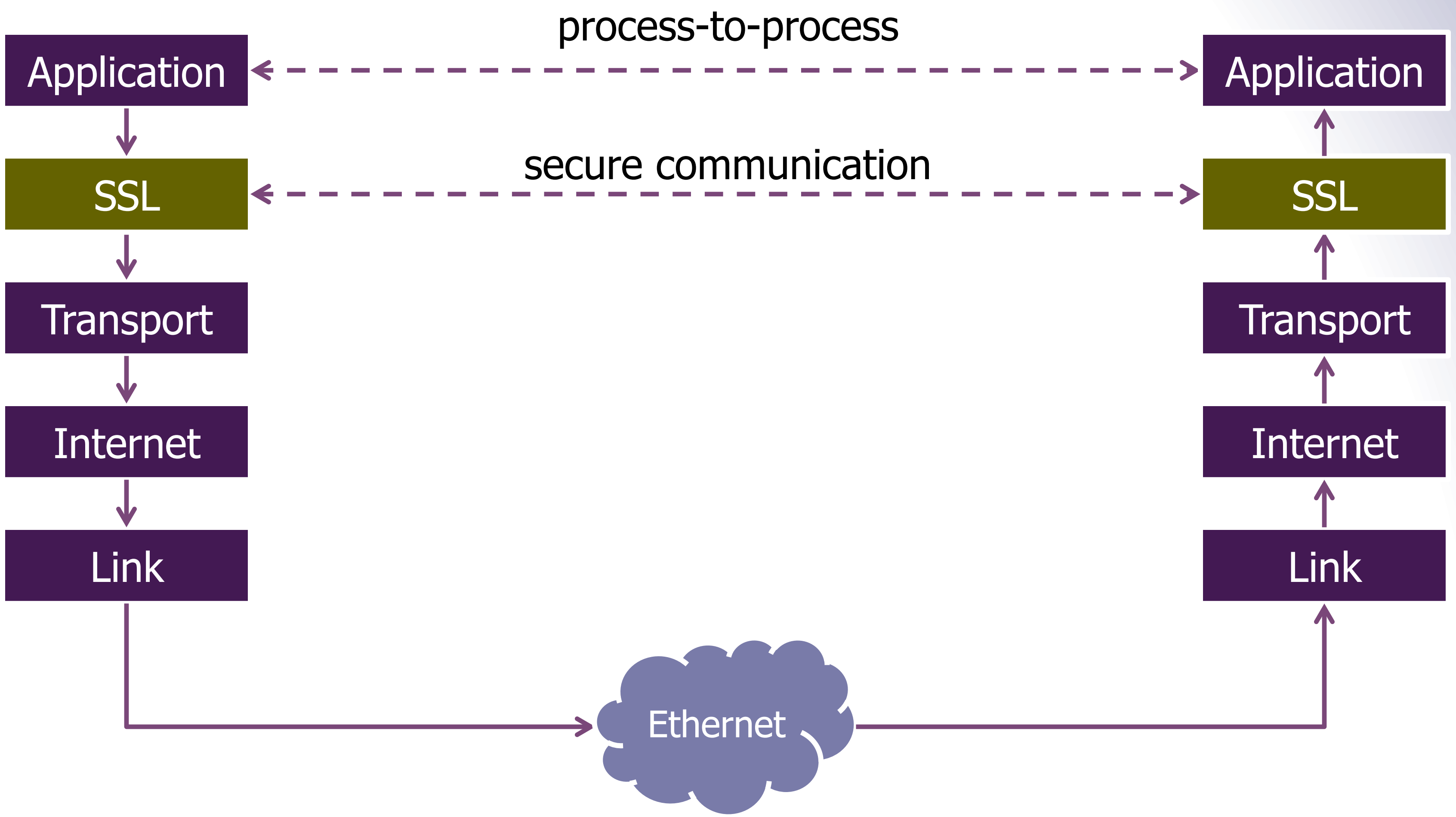
- Most widely deployed security protocol in the world
- SSL was developed by Netscape to offer secure access to web servers (HTTPS)
- History
  - SSL v1.0 never publicly released
  - SSL v2.0 released in 1994 (flawed)
  - SSL v3.0 released in 1996, leads to TLS 1.0 in 1999

Transport Layer Security (TLS)

- TLS is an IETF standard based on SSL v3.0
  - Slight modifications compared to SSL v3.0
  - TLS v1.0 and SSL v3.0 do not interoperate
  - TLS v1.0 sometimes called SSL v3.1
  - TLS v1.0 defined in RFC 2246
  - TLS v1.2 updated in RFC 5246 (August 2008)

- Current version (March 2011)
  - TLS v1.2 (prohibits SSL v2.0)
  - RFC 6176

SSL in the layered model



Approaches

Create a new protocol from an existing protocol

- Examples:
  HTTP (80) / HTTPS (443), FTP (21) / FTPS (990), SMTP (25) / SMTPS (995), POP3 (110) / POP3S (995), IMAP (143) / IMAPS (993)
- Disadvantage: only clients supporting TLS can connect
- Advantage: we are sure that communications are secure

Extend a protocol to negotiate SSL/TLS

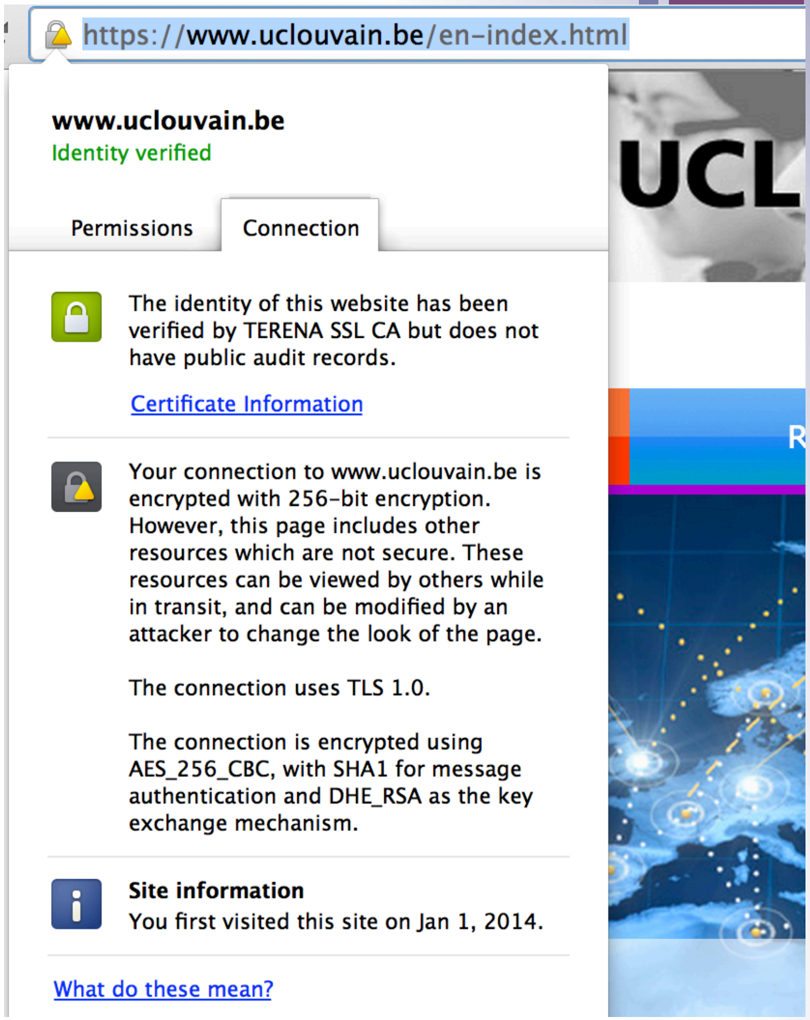
- Examples: (E)SMTP, POP3, IMAP, with the help of the STARTTLS command the client can ask to use TLS
- Advantage: the client is not required to support TLS to use the service

OpenSSL


```
bash$ openssl s_client -connect www.uclouvain.be:443
CONNECTED(00000003)
[...]
Certificate chain
 0 s:/C=BE/L=Louvain-la-Neuve/O=Universit\xC3\xA9 Catholique de Louvain/OU=Portail UCL/CN=www.uclouvain.be
   i:/C=NL/O=TERENA/CN=TERENA SSL CA
[...]
Server certificate
-----BEGIN CERTIFICATE-----
MIIErDCCA5SgAwIBAgIRAOjy08jirG7k+6k8Ln7bZxQwDQYJKoZIhvcNAQEFBQAw
[...]
-----END CERTIFICATE-----
[...]
SSL handshake has read 5258 bytes and written 328 bytes
---
New, TLSv1/SSLv3, Cipher is DHE-RSA-AES256-SHA
Server public key is 2048 bit
[...]
SSL-Session:
    Protocol  : TLSv1
    Cipher    : DHE-RSA-AES256-SHA
    Session-ID: C0FE449DC7345355B4119A095C27DA72691326880FE52271FB2CB3B0DCF29FE0
    Master-Key: 7A8DE9425505930A2F11AFC241F9236ABA61DAC7BFC0A9709C6F887D819BAA42C5F1B7A9E01CC26945A[...]
```

Example: HTTPS

- TLS guarantees data confidentiality and authenticity (server, possibly client)
  - The server must have a certificate
  - The client can have one
    - e.g., e-banking, Belgian SPF Finances

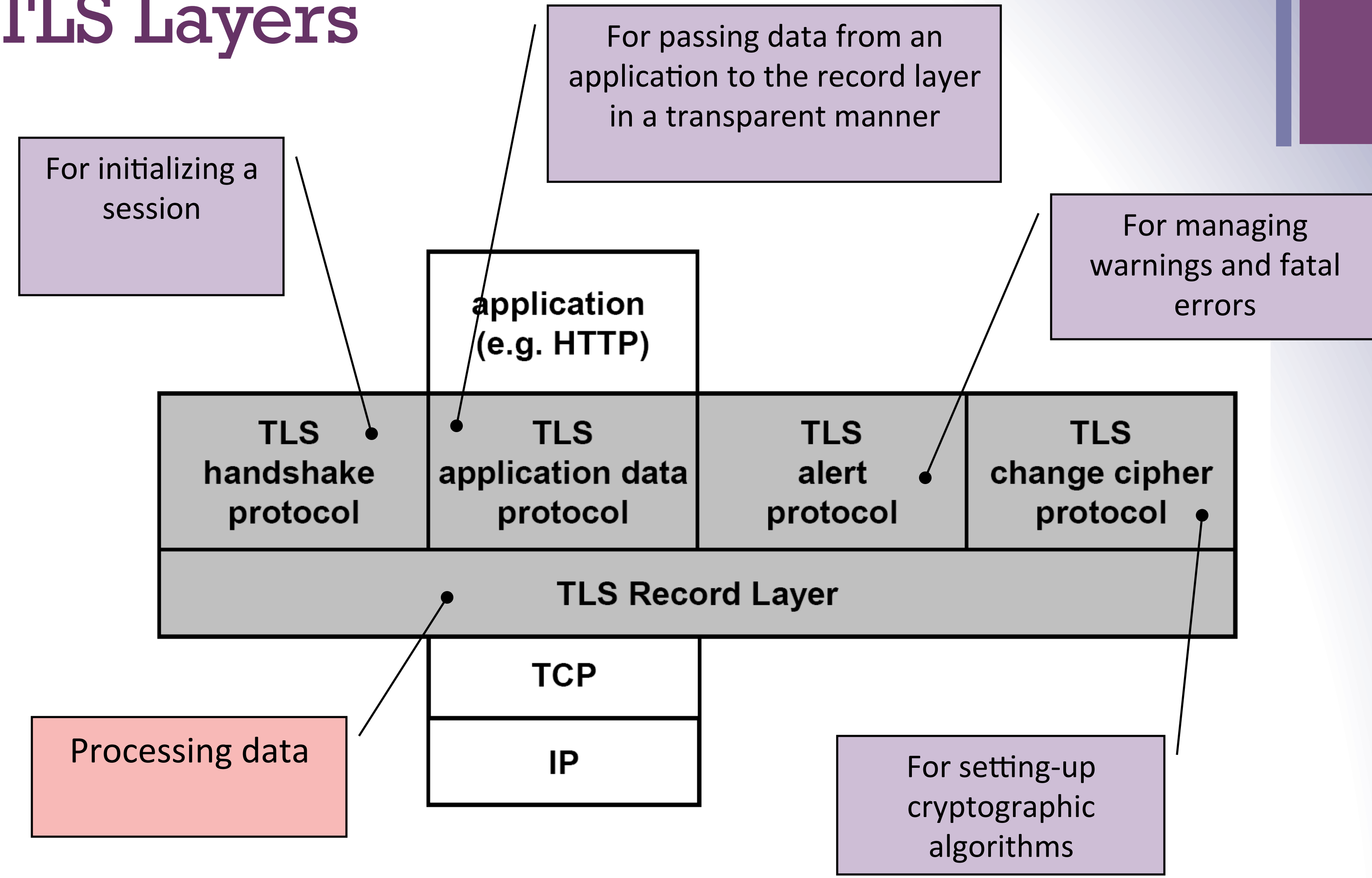


Example: Mail

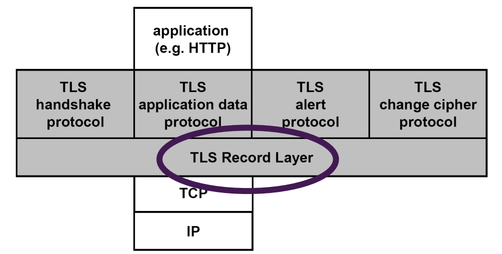
- ESMTP (sending mail), POP3 (mailbox access), IMAP (better mailbox access)
  - TLS is implemented as a protocol extension
  - The use of TLS is optional (needs to be configured)
- By default these protocols send cleartext passwords
- TLS protects passwords and email contents

TLS Protocol

TLS Layers

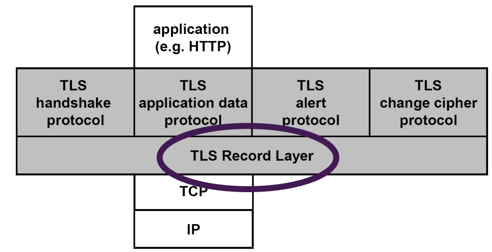

For passing data from an application to the record layer in a transparent manner
For initializing a session
For managing warnings and fatal errors
application (e.g. HTTP)
TLS handshake protocol
TLS application data protocol
TLS alert protocol
TLS change cipher protocol
TLS Record Layer
TCP
IP
Processing data
For setting-up cryptographic algorithms

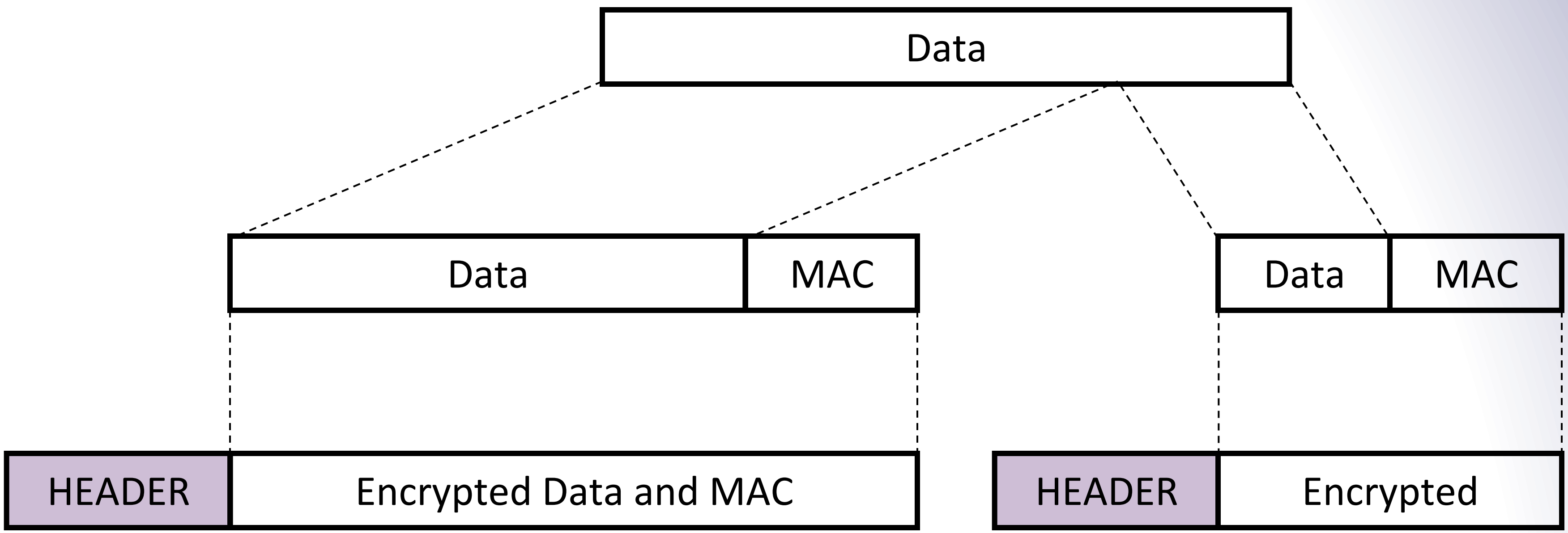
TLS Record Layer



- Processing of data
  - Fragmentation
  - Compression (optional)
  - Authentication
  - Encryption
- It delivers processed fragments to the transport layer (TCP)
- At the receiving end, the inverse operations are carried out

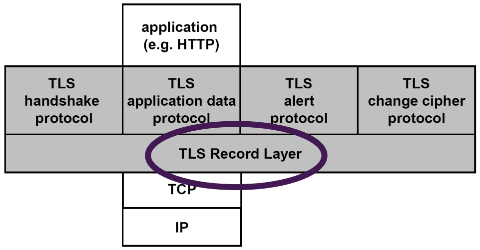
Record Layer Summary





+

MAC Computation

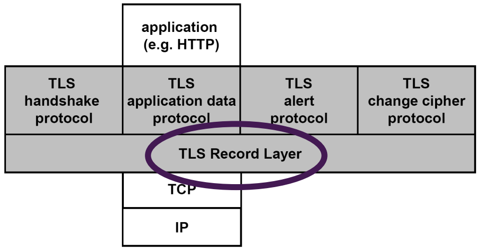


MAC = **Hash** (MAC_key || Pad2 ||

Hash (MAC_key || Pad1 || Seq || Length || Content))

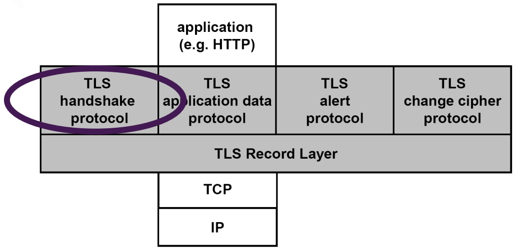
- MAC_key: secret shared by client and server
- Pad1, Pad2: pre-defined constants
- Seq: sequence number of this message
- **Hash**: Either HMAC-MD5 or HMAC-SHA1
- Length: Length in bytes of the compressed record
- Content: Compressed record

Encryption



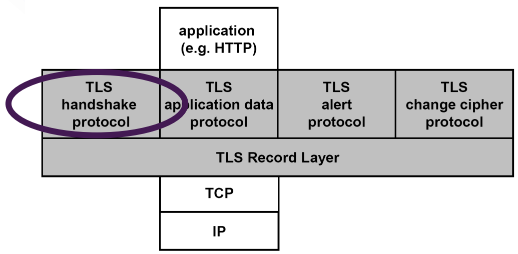
- Encryption is performed on compressed and authenticated records
- Block ciphers:
  - DES (40 bits or 56 bits), 3DES, IDEA, RC2 (40 bits)
  - AES (128 bits or 256 bits) in TLS v1.1
- Stream ciphers:
  - NULL, RC4 (40 bits or 128 bits)
- The client should refuse 40-bit keys if such a cipher is suggested by the server (warning enforced in TLS 1.1)

Handshake in Brief



- Negotiation of:
  - Protocol version (SSL 3.0, TLS 1.0, TLS 1.1)
  - Algorithms:
    - Key exchange (RSA, Diffie-Hellman)
    - Encryption (DES, 3DES, IDEA, RC4, RC2, AES)
    - MAC (HMAC-MD5, HMAC-SHA)
    - The client proposes the desired algorithms in order of preference, the server chooses
- Optional authentication of the partner using a certificate
- Messages are not encrypted
- Last messages authenticate the exchange

Handshake Exchanges



Client_Hello (crypto, random) →

← Server_Hello (crypto, random)

← Server Certificate

← Server_Hello_Done

Client_Key_Exchange →

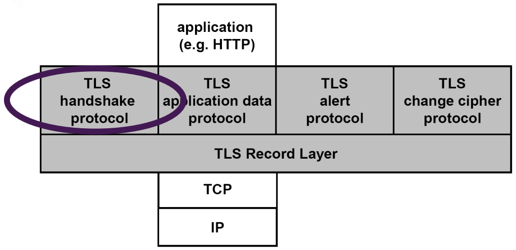
Change_Cipher_Spec →

Handshake_Finished →

← Change_Cipher_Spec

← Handshake_Finished

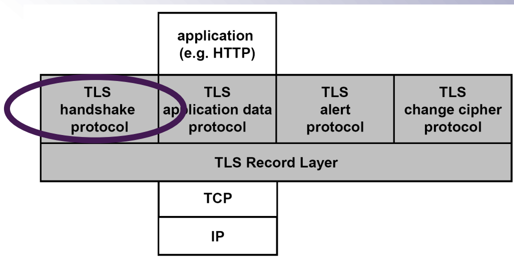
Client_Hello Content



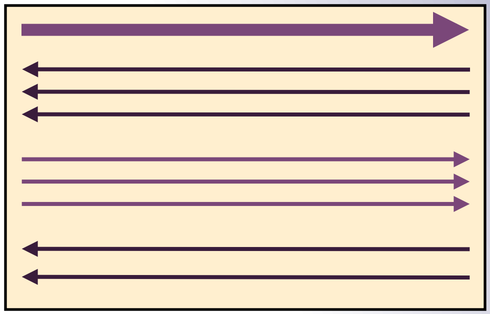
- Goal
  - Used by the client to initiate SSL session
  - Sent in clear without signature

- Content
  - Protocol Version
  - 32 bytes long random number
  - Composed of two parts:
    - 4 bytes Unix timestamp (number of seconds since 01/01/1970)
    - 28 bytes random number
  - Optional Session Identifier
    - Each SSL session has an identifier which can be used later to restart a session
  - List of supported Ciphers
  - List of supported Compression Methods

Client_Hello Crypto

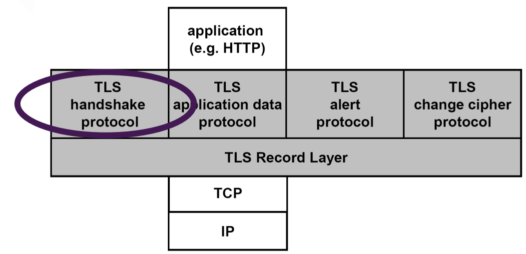


- List of supported cryptographic algorithms
  - Authentication + key exchange + cipher + hash



- Authentication
  - RSA or DSS
- Key Exchange
  - RSA, Diffie Hellman
- Encryption
- Hash

Cipher Suite Examples

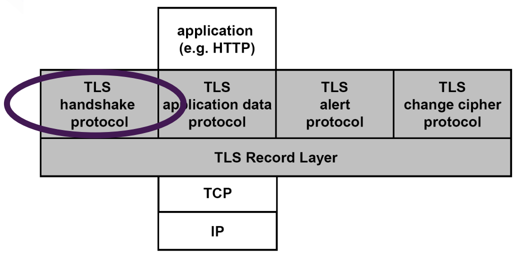



```
CipherSuite TLS_DH_DSS_WITH_DES_CBC_SHA = { 0x00,0x0C };

CipherSuite TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA = { 0x00,0x0D };

CipherSuite TLS_DH_RSA_WITH_DES_CBC_SHA = { 0x00,0x0F };

CipherSuite TLS                                 10};

CipherSuite TLS                                 ;

CipherSuite TLS                                 x13};

CipherSuite TLS                                 ;

CipherSuite TLS                                 x16};
```

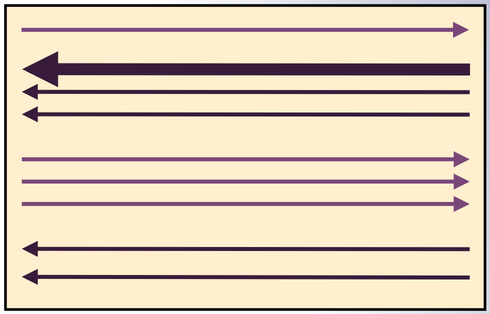
DH: Diffie-Hellman
DSS: Digital Signature Standard
DES: Data Encryption Standard
CBC: Cipher Block Chaining
SHA: Secure Hash Algorithm
EDE: Encrypt-Decrypt-Encrypt

Source: RFC4346.

Server_Hello Content

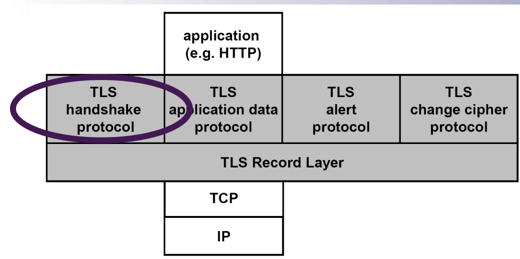


- Goal
  - Used by the server to reply to Client_Hello
  - Sent in the clear without signature



- Content
  - Protocol version: highest version of the protocol supported by both client and server
  - Random number
  - Optional Session Identifier, if it allows sessions to be resumed
  - Cipher Suite: One of the cipher suites proposed by client
  - Compression Method

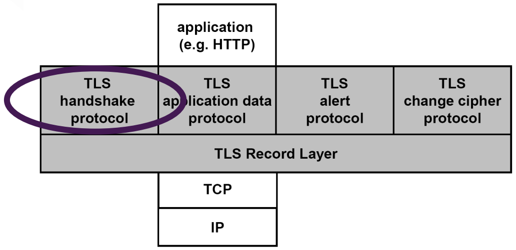
Server Certificate



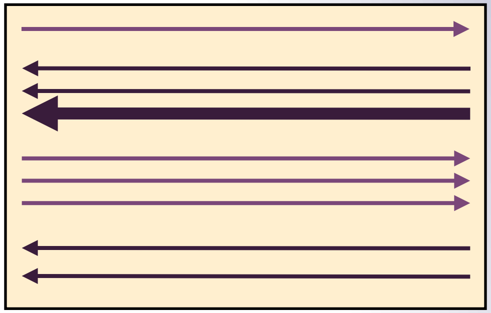
- Goal
  - Sent by the server to authenticate itself
  - A server may have several certificates from different certification authorities
- Content
  - A list of X.509 certificates:
    - Server certificate
    - Certificates of certification authorities

- Certificate can also be sent by the client when client authentication is requested by the server with Certificate_Request

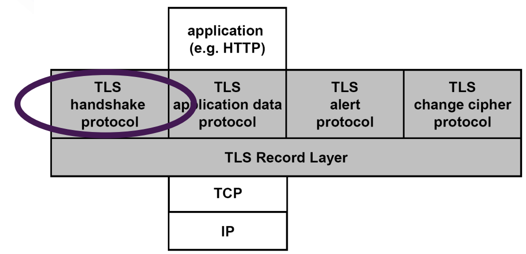
Server_Hello_Done



- Goal
  - Indicates that server has finished its handshake first phase
  - Sent unencrypted

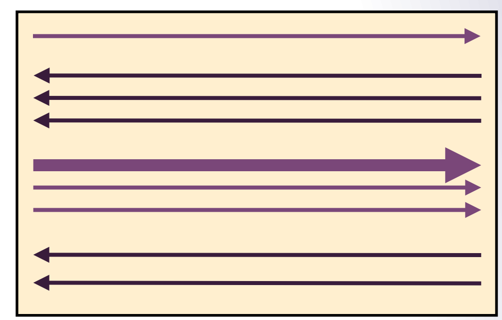


Client_Key_Exchange

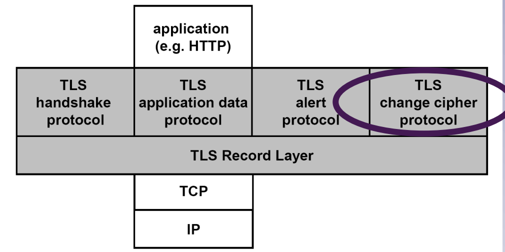


- Goal
  - Used by the client to send the PreMasterSecret, which is used to derive session keys
  - Encrypted with the server's public key

- Content
  - Encrypted PreMasterSecret with the public key of the server

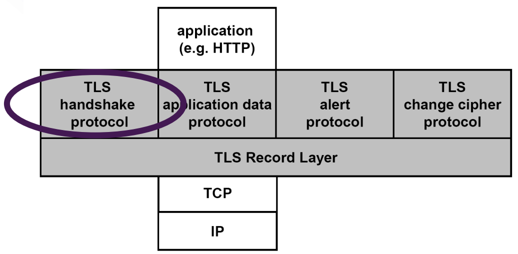


Change_Cipher_Spec



- Goal
  - Used by client and server to indicate that they start using a (new) key
  - During handshake, indicates that next message will be encrypted with the appropriate key

Handshake_Finished



- Goal
  - Sent by both client and server to confirm the establishment of the secure SSL session
    - Session is established only if client received expected Finished message from server and vice-versa
  - Allows to detect man in the middle attacks on Client_Hello and Server_Hello messages
    - Example: Attacker changes cipher list to propose weaker ciphers
  - First encrypted message on each direction

- Contents
  - Keyed hash (MD5 or SHA-1) of all the handshake messages and the MasterSecret

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

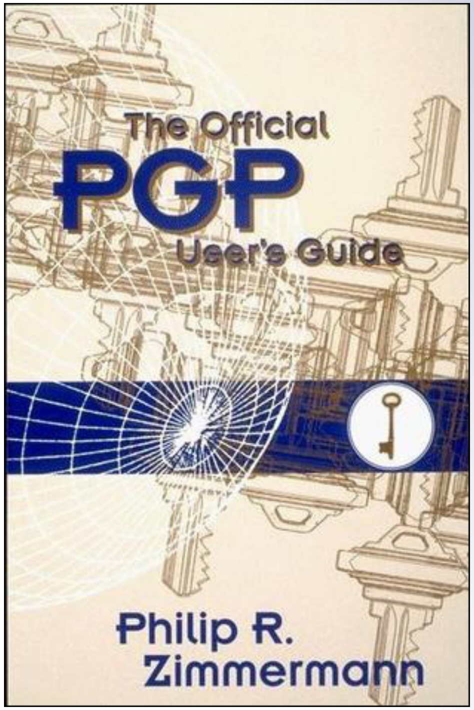
2010:

- 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

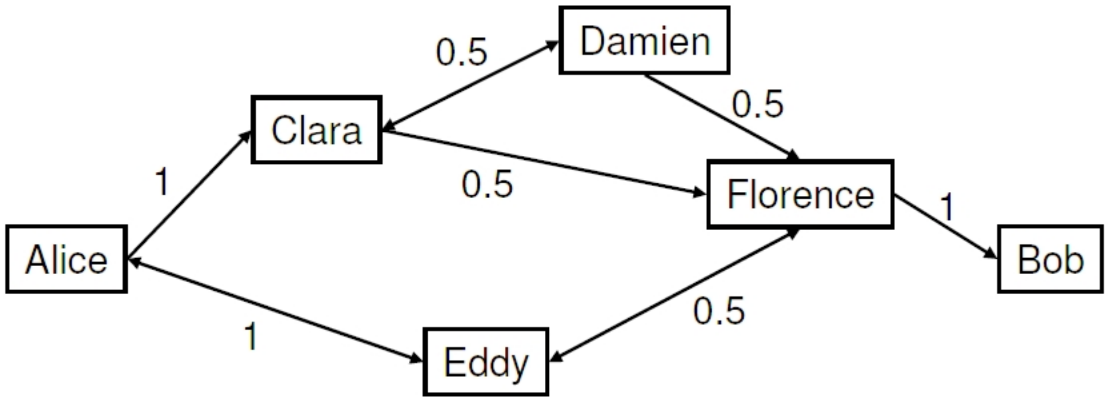
Public-key Distribution

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

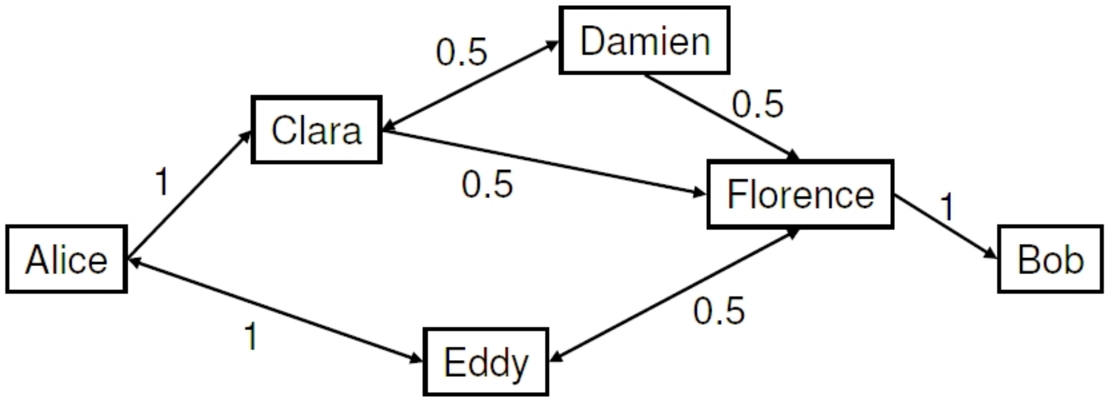
Validity and Trust in PGP

- 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

Any questions?

Stay tuned

Next time you will learn about

Passwords | Time-memory trade-offs