Internet-Sicherheit A
Transport Layer Security

Prof. Dr. Christian Dietrich
<dietrich@internet-sicherheit.de>

This chapter

- Transport Layer Security (TLS), Secure Sockets Layer (SSL)
- TLS Handshake
- Key exchange mechanisms
- Authentication and X.509 certificates
- TLS extensions
- Server Name Indication
- Attacks against TLS
    - CRIME: Avoid compression
    - Heartbleed: OpenSSL implementation issue
- Signing Oracle

Threat model

- The Internet is an open network of interconnected systems with varying legislations
- Data transmission happens over potentially insecure/eavesdropping nodes
- Many applications require the transmission of sensitive data such as credentials (passwords, secrets, credit card information, private information)

- Threat model
  - An attacker controls the whole network communication
  - Eavesdrop, modify and inject communication packets

- Imagine the worst case situation
  An attacker who controls the local network, all Internet routes, and DNS resolution

Goals of Transport Layer Security (TLS)

- For two communicating peers, Transport Layer Security (TLS) provides
    1. Privacy and
    2. Data integrity

- Security is provided to any protocol that is spoken on top of a TLS channel

Network protocol layers

Application	HTTP, FTP, SMTP, IMAP, SSH
Transport	TCP, UDP
Network/Internet	IP, ARP, ICMP, IGMP, IPSEC
Data link	Ethernet

Network protocol layers and TLS

Application	HTTP, FTP, SMTP, IMAP, SSH
Security	Transport Layer Security
Transport	TCP, UDP
Network/Internet	IP, ARP, ICMP, IGMP, IPSEC
Data link	Ethernet

TLS: Key Concepts

- Symmetric cryptography
  - Used to encrypt the data transmitted
  - The connection is private
- Asymmetric cryptography
  - Used to authenticate the identity of the communicating parties
  - The connecting parties “know” and trust each other
- Message authentication code
  - Used to ensure integrity
  - Nobody can modify the transmitted messages without being noticed

What is TLS?

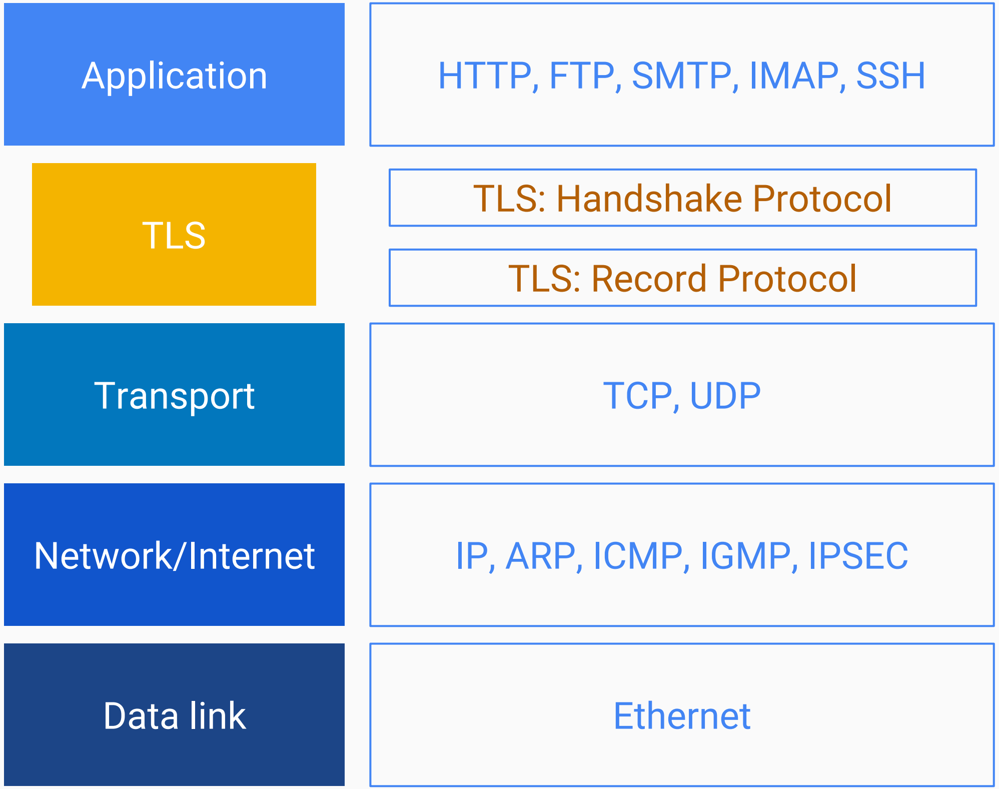
- TLS is the de facto standard for secure communication in TCP/IP networks
- Predecessor was named Secure Sockets Layer (SSL)
- TLS can be used with many protocols, often in one of two ways
  - A separate TCP port is used to offer a service where the application layer protocol (e.g. IMAP) is spoken on top of a TLS layer
    - TCP port 143 for plaintext IMAP
    - TCP port 993 for IMAP over TLS (IMAPS)
  - TLS is started during an existing TCP connection of the application layer protocol
    - Often called STARTTLS (SMTP, POP3, IMAP and other protocols)
- Most recent version (TLS 1.3) is in draft state

The history of TLS

- November 1993: Release of Mosaic, the first wide-spread web browser
- SSL 1.0 – Internal Netscape design, 1994
  - “This version circulated only internally (i.e., inside Netscape Communications), since it had several shortcomings and flaws.”
  - Several weaknesses
- SSL 2.0 – Netscape, Nov 1994
  - Several weaknesses
- SSL 3.0 – Netscape and Paul Kocher, Nov 1996
- TLS 1.0 – Internet standard, Jan 1999
  - Based on SSL 3.0, but not interoperable (different cryptographic algorithms)
  - Announces itself as SSL “3.1”
- TLS 1.1 – Apr 2006
- TLS 1.2 – Aug 2008
- TLS 1.3 – maybe 2018, possibly later

TLS Basics

- TLS consists of two (ordered) protocols
- Handshake protocol
  - Uses public-key cryptography to establish several shared secret keys between the client and the server
- Record protocol
  - Uses the secret keys (handshake protocol) to protect confidentiality, integrity, and authenticity of data exchanges between client and server



Application	HTTP, FTP, SMTP, IMAP, SSH
TLS	TLS: Handshake Protocol TLS: Record Protocol
Transport	TCP, UDP
Network/Internet	IP, ARP, ICMP, IGMP, IPSEC
Data link	Ethernet

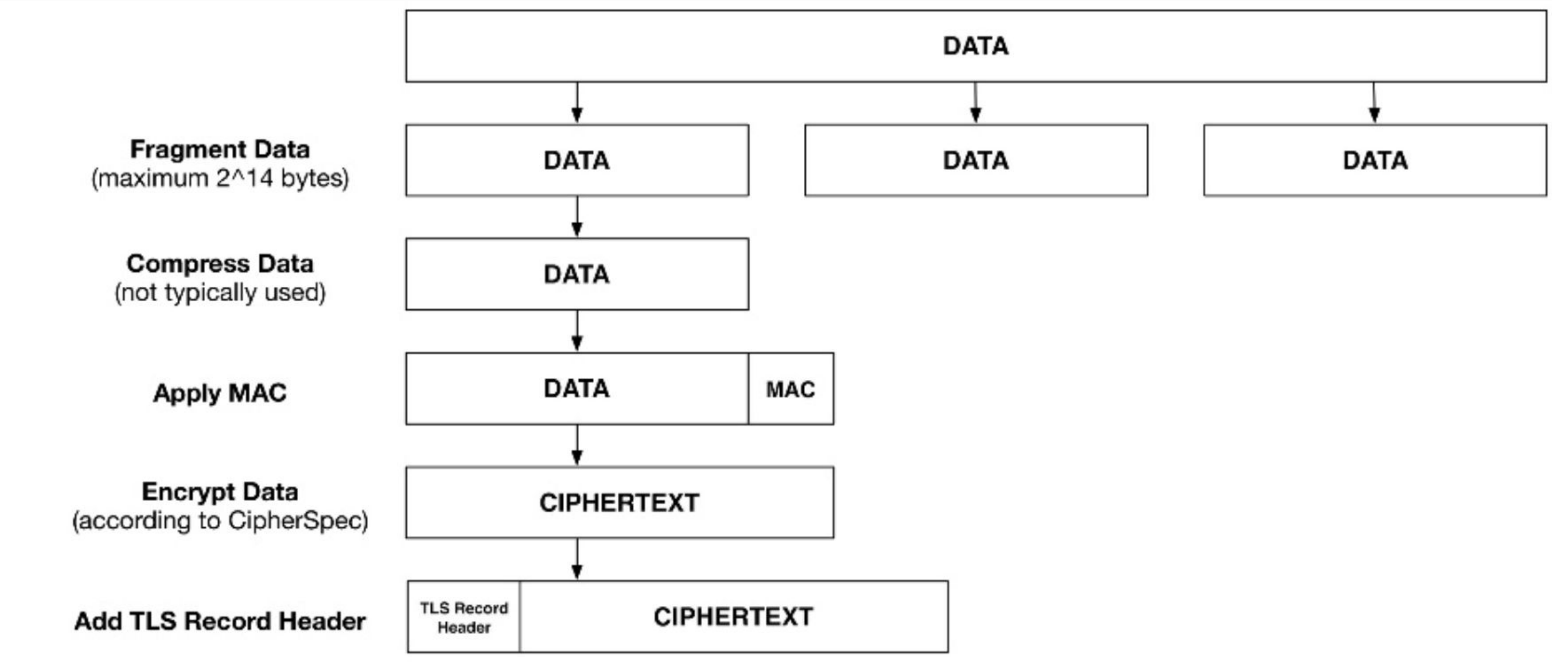
TLS Record Protocol

- The handshake is used to derive a shared secret

- The TLS record protocol takes care of
  - Confidentiality: Symmetric encryption with a first secret key (agreed on during the handshake)
  - Integrity: HMAC with a second secret key (agreed upon during the handshake)

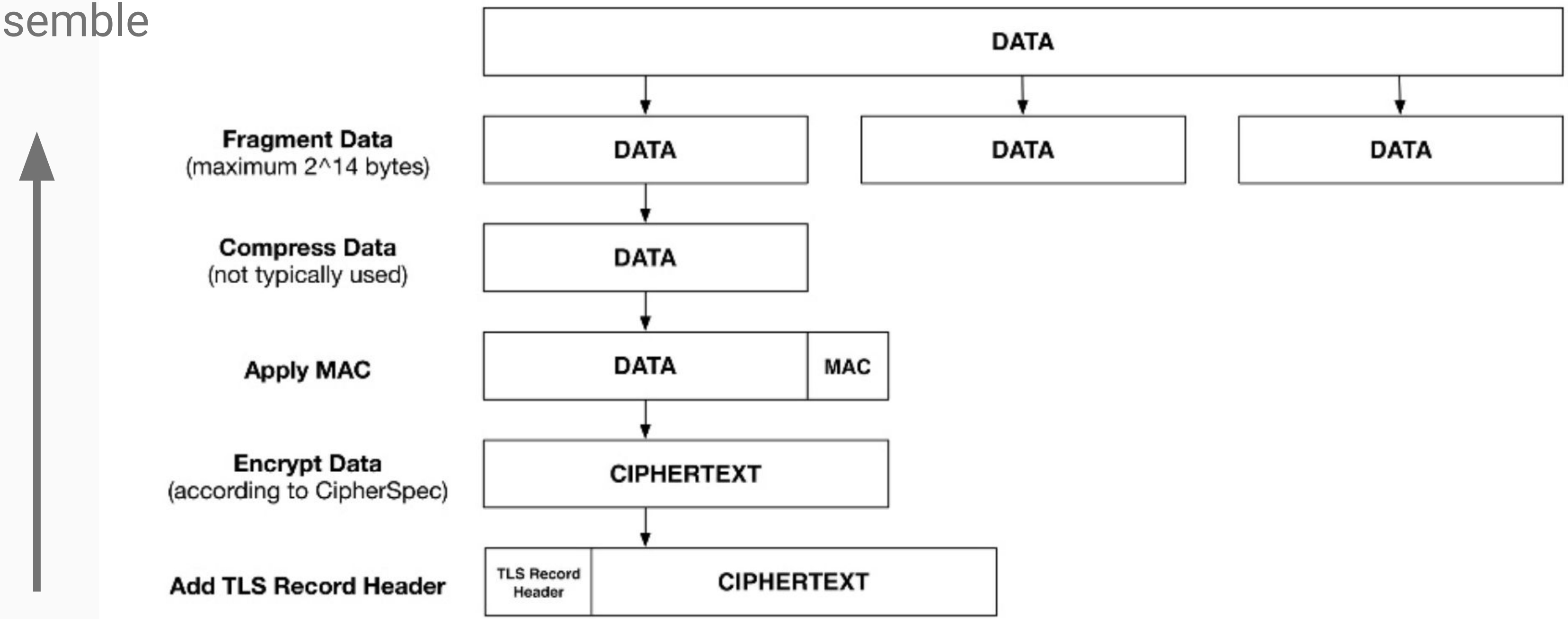
TLS Record Protocol: Sending side

- Fragment the data into manageable blocks
- Optionally compresses the data
- Apply a MAC
- Encrypt



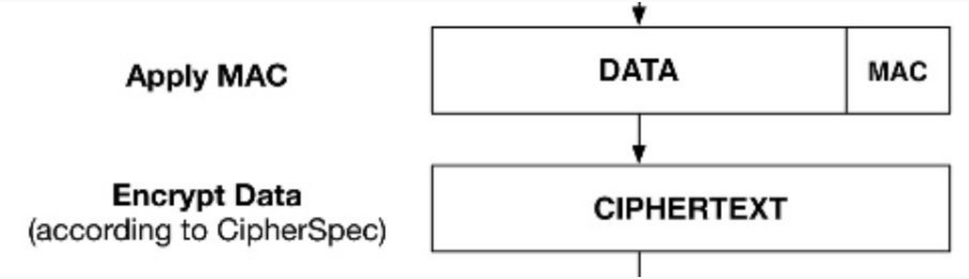
TLS Record Protocol: Receiving side

- Decrypt
- Verify (HMAC)
- Decompress
- Reassemble

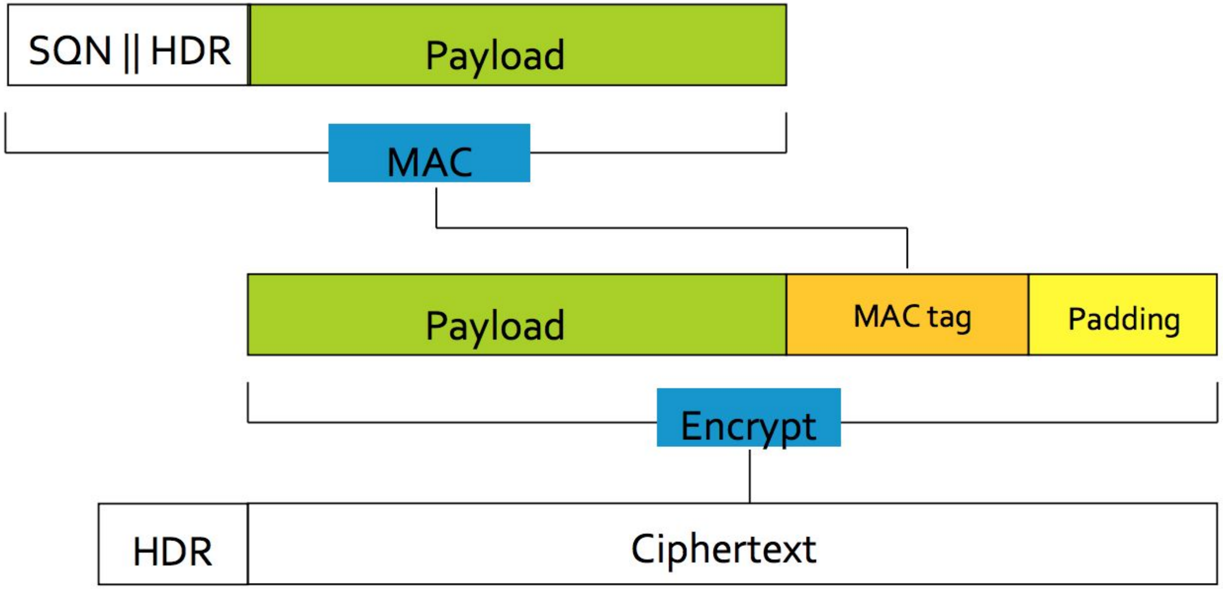


TLS 1.2 Record Protocol

- Order of integrity protection and encryption
- MAC-then-encrypt (used in TLS)
  - Guarantees integrity of the plaintext
  - The MAC is encrypted
  - Needs to decrypt on the receiver side before MAC verification
  - Susceptible to attacks (POODLE)
- Encrypt-then-MAC
  - Provides integrity over plaintext and ciphertext
  - Verify the MAC and only then decrypt (does not feed spoofed ciphertext into the decryption step)
- TLS 1.3: Upcoming versions will likely use a different scheme: Authenticated Encryption (simultaneously provides confidentiality, integrity, and authenticity of the data)



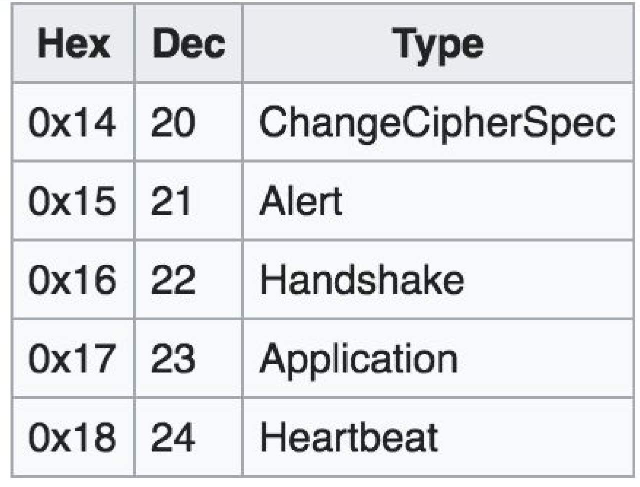
TLS Record Protocol: MAC-then-encrypt



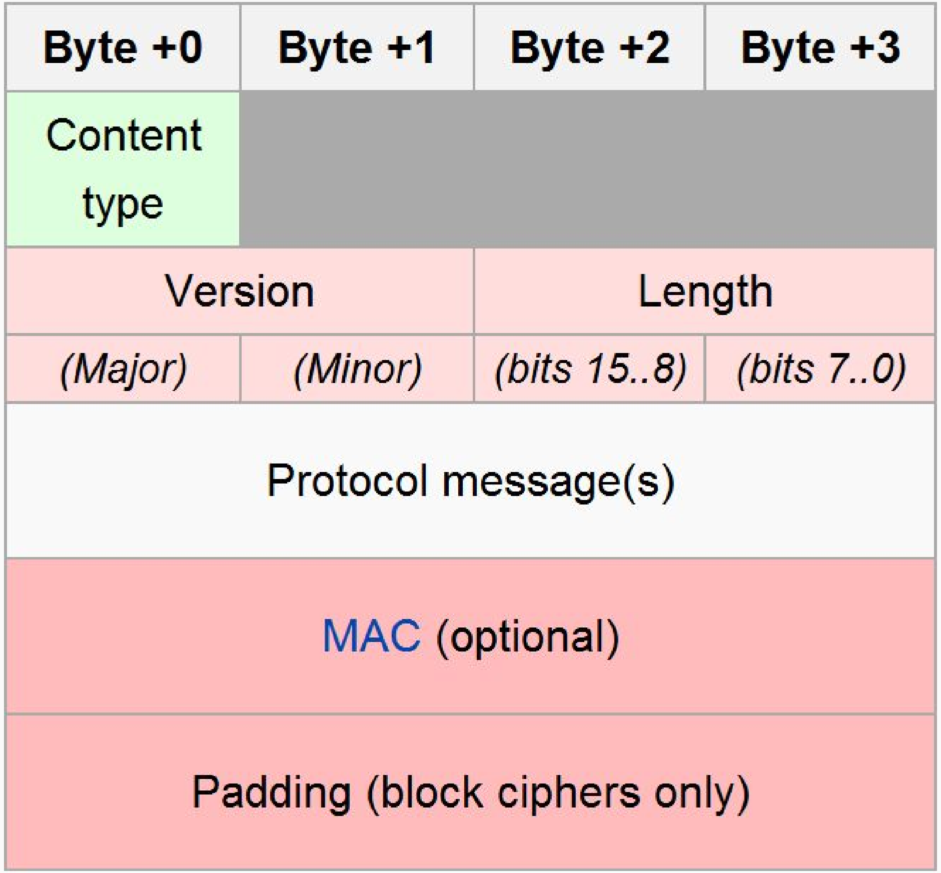
MAC	HMAC-MD5, HMAC-SHA1, HMAC-SHA256
Encrypt	CBC-AES128, CBC-AES256, CBC-3DES, RC4-128

TLS Record Format

- First byte is the content type and identifies the record layer protocol type



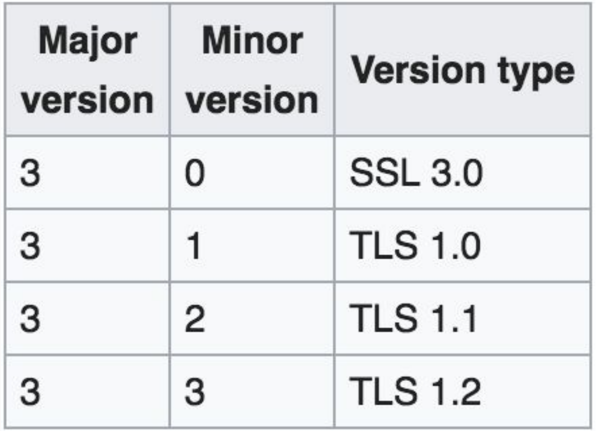
Hex	Dec	Type
0x14	20	ChangeCipherSpec
0x15	21	Alert
0x16	22	Handshake
0x17	23	Application
0x18	24	Heartbeat



Byte +0	Byte +1	Byte +2	Byte +3
Content type			
Version		Length	
(Major)	*(Minor)*	*(bits 15..8)*	*(bits 7..0)*
Protocol message(s)			
MAC (optional)			
Padding (block ciphers only)			

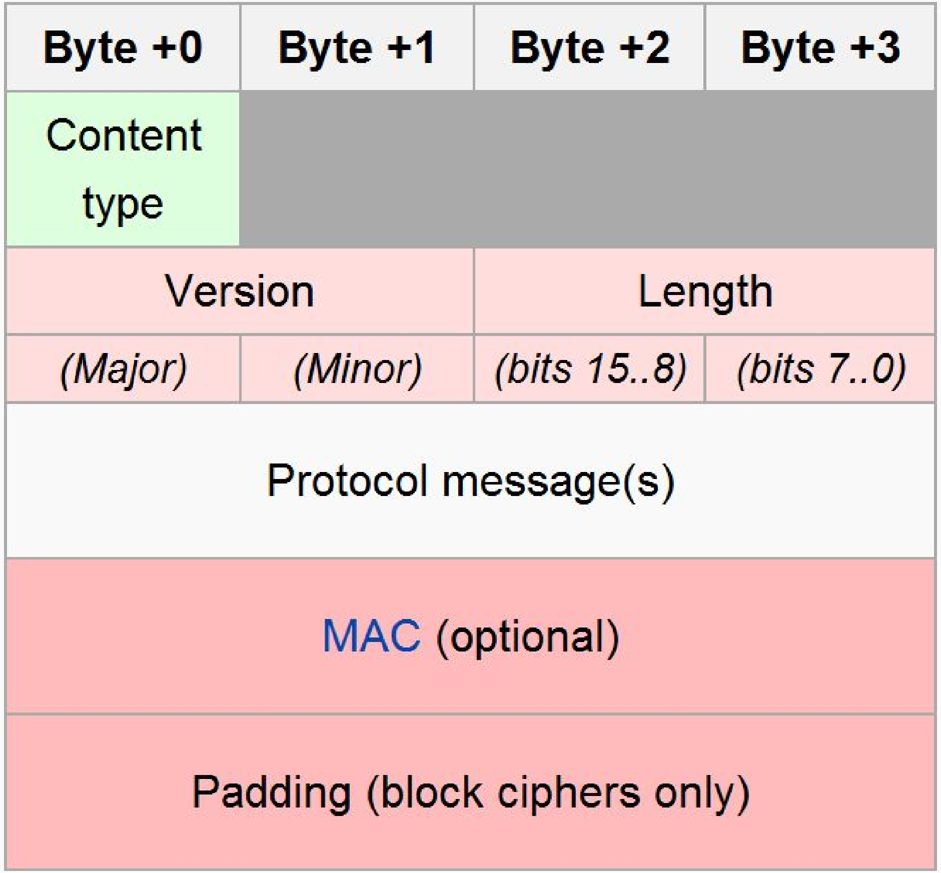
TLS Record Format

- Two bytes of version numbers
  - Major and minor version



Major version	Minor version	Version type
3	0	SSL 3.0
3	1	TLS 1.0
3	2	TLS 1.1
3	3	TLS 1.2

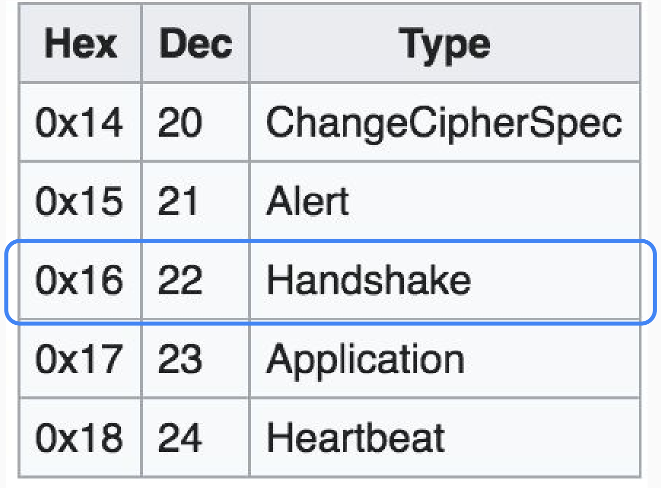
- Two bytes of total length information (Protocol message + MAC + Padding)



Byte +0	Byte +1	Byte +2	Byte +3
Content type			
Version		Length	
(Major)	*(Minor)*	*(bits 15..8)*	*(bits 7..0)*
Protocol message(s)			
MAC (optional)			
Padding (block ciphers only)			

TLS Handshake Protocol

- Operates on top of TLS records
- Establish the cryptographic parameters of the session
  - Agree on a protocol version
  - Select cryptographic algorithms
  - Authenticate the communicating peers (optional, but very often used)
  - Use public-key encryption techniques to generate shared secrets
- Each party may implement a specific subset of cryptographic operations
- The goal is to agree on the *strongest* possible crypto parameters that both parties support



Hex	Dec	Type
0x14	20	ChangeCipherSpec
0x15	21	Alert
0x16	22	Handshake
0x17	23	Application
0x18	24	Heartbeat

TLS Handshake Protocol

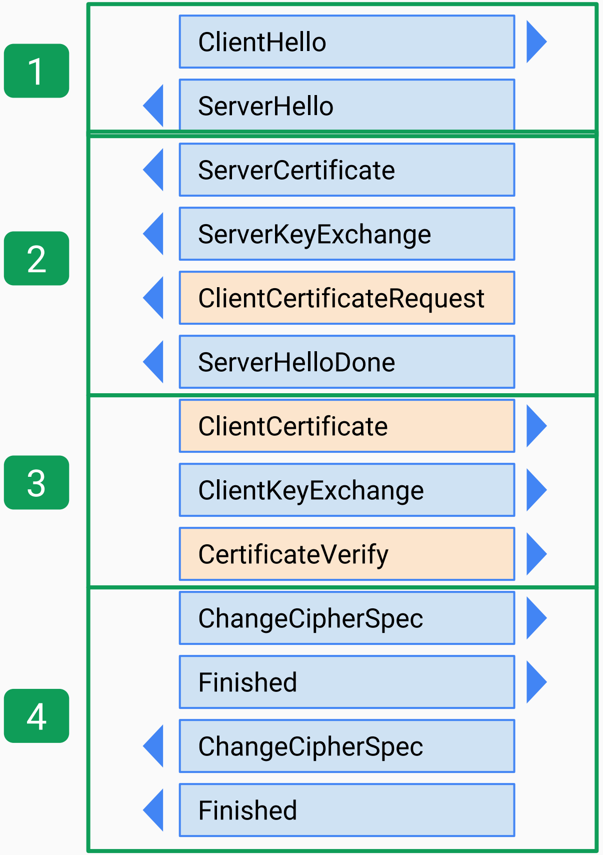
- Exchange hello messages to agree on algorithms, exchange random values, and check for session resumption
- Exchange the necessary cryptographic parameters to allow the client and server to agree on a premaster secret
- Exchange certificates and cryptographic information to allow the client and server to authenticate themselves
- Generate a master secret from the premaster secret and exchanged random values
- Provide security parameters to the record layer
- Allow the client and server to verify that their peer has calculated the same security parameters and that the handshake occurred without tampering by an attacker

TLS Handshake: Premaster and master secret

- Premaster secret
  - Input that helps to derive the master secret
  - Detach the master secret from contents that is transmitted between the client and the server
- Master secret
  - 48-byte secret byte sequence
  - Used on both sides to derive keying material for the record layer
    - the key for symmetric encryption
    - the initialization vectors (IV) for symmetric encryption
    - the HMAC for message integrity
  - Computed by a function of the premaster secret and random nonces (depends on the key exchange algorithm)
  - RSA: Permutation of the client-chosen premaster secret (the premaster secret is sent encrypted with the server’s public key)
  - DH: Client and server derive the master secret from the DH key exchange result

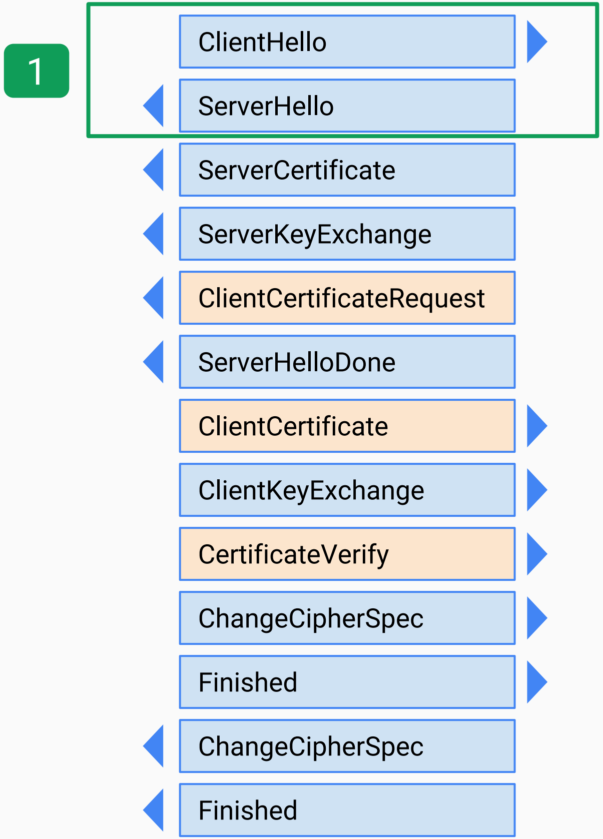
TLS Handshake

- Consists of up to 13 messages, some of which are optional (marked orange)
- Divided into 4 phases
  1. Establish security capabilities
  2. Server authentication and key exchange
  3. Client authentication and key exchange
  4. Finish



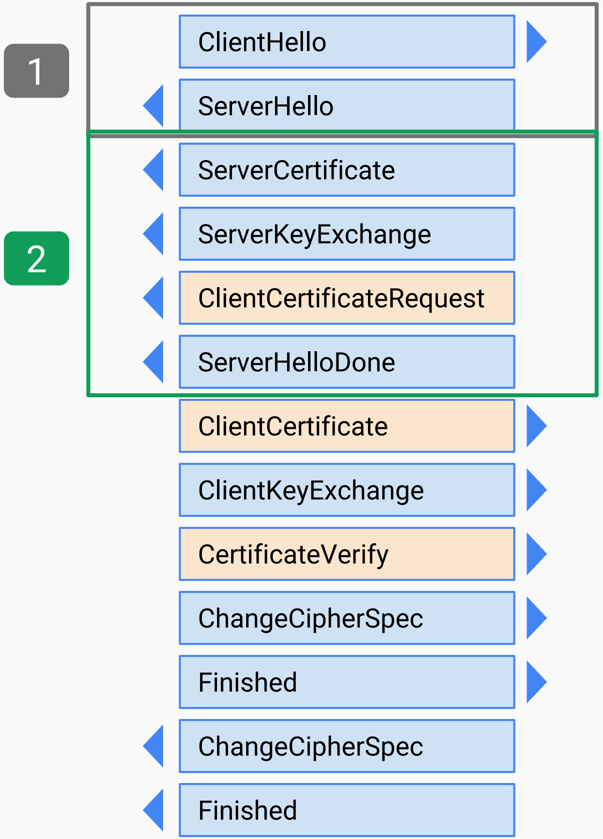
TLS Handshake: Phase 1

- Establish security capabilities
  - Client and server generate nonces
  - Protocol Version, Session ID, Cipher Suite, and Compression Method
- `ClientHello`
  - Highest supported TLS version
  - Client-generated nonce
  - Session ID (if continuation) or zero
- `ServerHello`
  - Chosen TLS version
  - Server-generated nonce
  - Session ID (copied from client) or zero



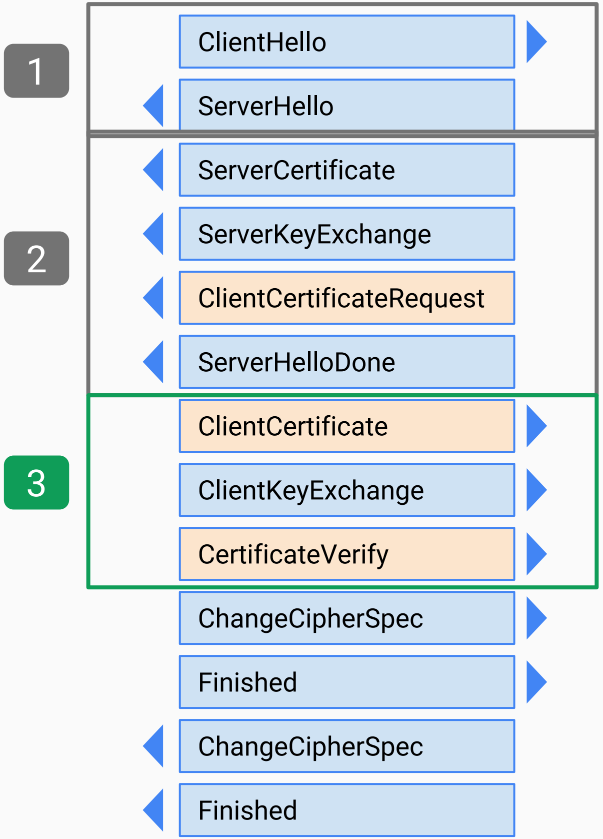
TLS Handshake: Phase 2

- Server authentication and key exchange
- `ServerCertificate`
  - Authenticate to the client using an X509v3 certificate
  - Possibly send intermediate certificates
- `ServerKeyExchange`
  - Diffie-Hellman: Server parameters
  - RSA: no ServerKeyExchange parameters required
  - PSK: identity hint
- `ClientCertificateRequest`
  - If present, ask the client to present its client certificate ⇒ mutual authentication (both parties)
- `ServerHelloDone`
  - "The hello-message phase of the handshake is complete"



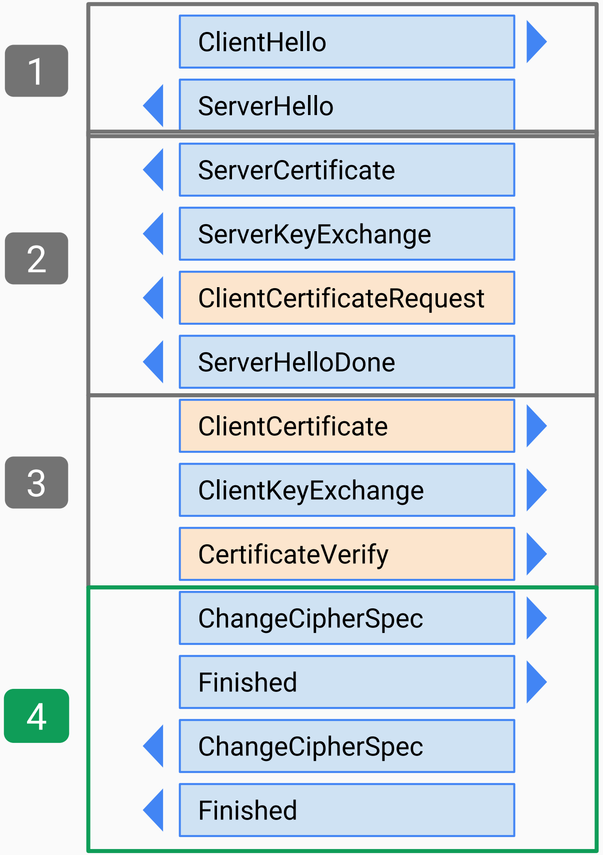
TLS Handshake: Phase 3

- Client authentication and key exchange
- `ClientCertificate`
  - Send an X509v3 certificate, if the server asked for it with a ClientCertificateRequest in phase 2
- `ClientKeyExchange`
  - Second part of the key exchange
  - Diffie-Hellman: Client parameters
  - RSA: 48-byte premaster secret, encrypted with the server's public key
  - Both parties now have enough info to generate the master secret (and the session keys)
- `CertificateVerify`
  - Client proves access to the ClientCertificate's private key by signing the handshake messages



TLS Handshake: Phase 4

- Handshake completed
- ChangeCipherSpec (sent by the client)
  - Switch to the agreed cipher
  - From now on, the messages from the client will be encrypted and authenticated
- Finished (sent by the client)
  - Contains a hash and MAC over the previous handshake messages + "client finished"
- ChangeCipherSpec (sent by the server)
  - Switch to the agree cipher
  - From now on, the messages from the server will also be encrypted and authenticated
- Finished (sent by the server)
  - Contains a hash and MAC over the previous handshake messages + "server finished"



TLS: Cipher suites

- `ClientHello` and `ServerHello` announce which cipher suites the implementation supports and which is chosen for the session
- A cipher suite comprises of
  - a key exchange algorithm
  - an authentication algorithm
  - a symmetric encryption algorithm
  - and a Message Authentication Code (MAC) algorithm
- Naming scheme
  - TLS_RSA_WITH_3DES_EDE_CBC_SHA
    - RSA specifies the key exchange algorithm and the authentication algorithm
    - 3DES_EDE_CBC specifies the block cipher to encrypt messages
    - SHA indicates the message authentication algorithm

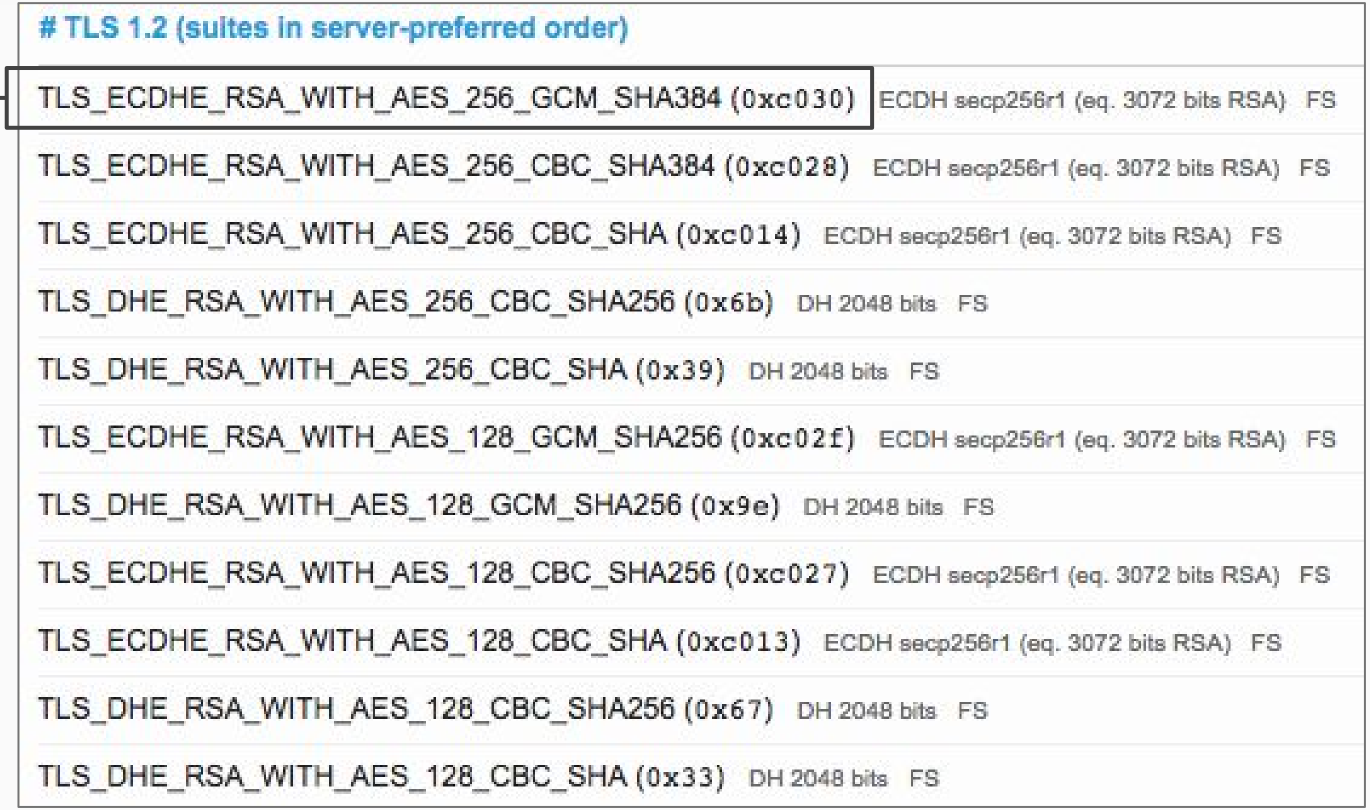
TLS: Cipher suites (www.internet-sicherheit.de)

ECDHE: Elliptic Curve Diffie-Hellman Key Exchange

RSA: RSA authentication

AES256 with GCM: Symmetric encryption with authentication

SHA384: PRF/hash function



TLS 1.2 (suites in server-preferred order)

TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)	ECDH secp256r1 (eq. 3072 bits RSA)	FS
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xc028)	ECDH secp256r1 (eq. 3072 bits RSA)	FS
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)	ECDH secp256r1 (eq. 3072 bits RSA)	FS
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 (0x6b)	DH 2048 bits	FS
TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x39)	DH 2048 bits	FS
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)	ECDH secp256r1 (eq. 3072 bits RSA)	FS
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 (0x9e)	DH 2048 bits	FS
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xc027)	ECDH secp256r1 (eq. 3072 bits RSA)	FS
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)	ECDH secp256r1 (eq. 3072 bits RSA)	FS
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 (0x67)	DH 2048 bits	FS
TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x33)	DH 2048 bits	FS

TLS: Considerations for key exchange algorithms

- Various key exchange algorithms can be chosen from
  - Static RSA key exchange
  - Transient RSA key exchange
  - Diffie-Hellman key exchange

  - Pre shared keys (PSK) (not really a key *exchange*)

- Static RSA key exchange may not provide perfect forward secrecy (PFS)
- DH key exchange however does provide PFS
- Transient RSA key exchange is supported in TLS, but rarely used

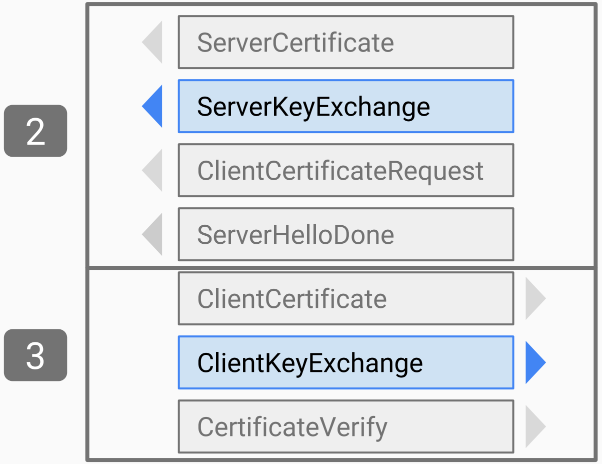
TLS: Considerations for pre-shared keys

- Usually, TLS uses public key certificates
- But: Various forms of pre-shared key protocols exist in TLS (RFC4279)
  - Plain PSK
  - DHE-PSK
    - Protect against dictionary attacks by passive eavesdroppers (but not active attackers) and also provide Perfect Forward Secrecy (PFS)
  - RSA-PSK
    - Combine public-key-based authentication of the server (using RSA and certificates) with mutual authentication using a PSK
    - TLS-RSA-PSK is used in the web authentication of the German electronic ID

- Advantages
  - PSK can avoid the need for public key operations
  - Bind the TLS connection to a previously established session (over a different protocol)

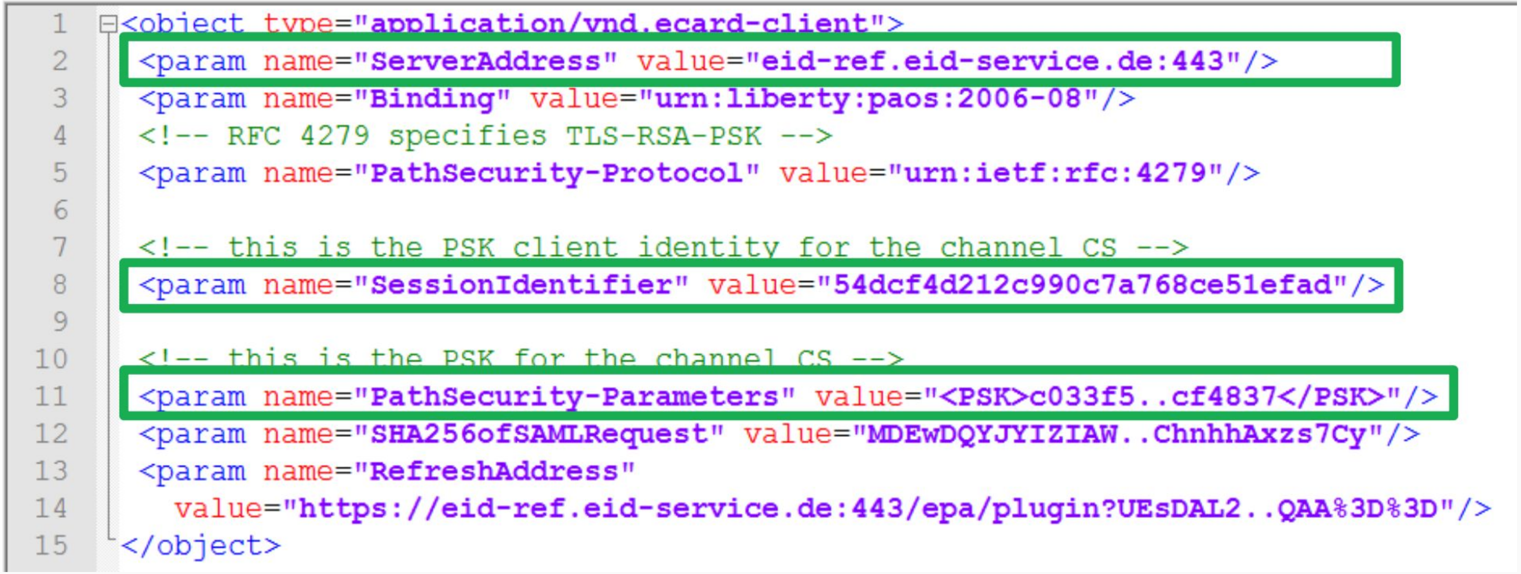
TLS-RSA-PSK: German electronic ID

- One application of TLS-RSA-PSK is the web authentication protocols for the German electronic ID (BSI-TR-03124 and BSI-TR-03112)
- The TLS session is bound to a web authentication session using the PSK
  - `ServerKeyExchange`
    - Transmits the session identifier as the psk_identity hint field
  - `ClientKeyExchange`
    - Transmits the RSA-encrypted PSK (the premaster secret)



TLS-RSA-PSK: German electronic ID

- The PSK and the identity hint are exchanged beforehand
  - Identity hint: `SessionIdentifier`
  - PSK: `PathSecurity-Parameters <PSK>...</PSK>`

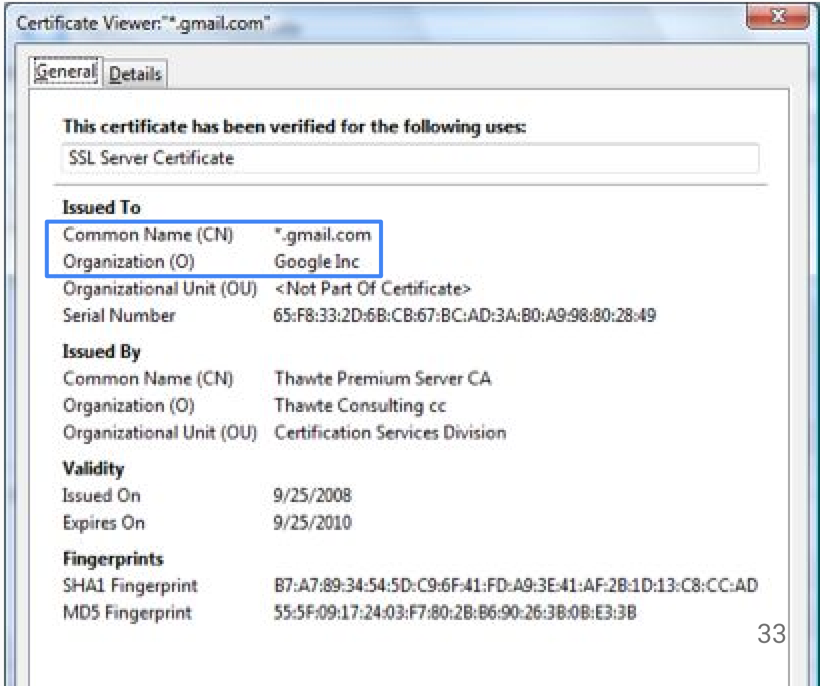


```
<object type="application/vnd.ecard-client">
  <param name="ServerAddress" value="eid-ref.eid-service.de:443"/>
  <param name="Binding" value="urn:liberty:paos:2006-08"/>
  <!-- RFC 4279 specifies TLS-RSA-PSK -->
  <param name="PathSecurity-Protocol" value="urn:ietf:rfc:4279"/>

  <!-- this is the PSK client identity for the channel CS -->
  <param name="SessionIdentifier" value="54dcf4d212c990c7a768ce51efad"/>

  <!-- this is the PSK for the channel CS -->
  <param name="PathSecurity-Parameters" value="<PSK>c033f5..cf4837</PSK>"/>
  <param name="SHA256ofSAMLRequest" value="MDEwDQYJYIZIAW..ChnhhAxzs7Cy"/>
  <param name="RefreshAddress"
    value="https://eid-ref.eid-service.de:443/epa/plugin?UEsDAL2..QAA%3D%3D"/>
</object>
```

Authentication in TLS

TLS authentication

- TLS serves two goals
  - Privacy (discussed in detail until now)
  - Authentication of the peers
- For authentication, X509 certificates and a public key infrastructure is used
- Two requirements
  - Authenticate a server (and optionally a client)
  - Obtain the server's (and optionally the client's) public key



This certificate has been verified for the following uses:

SSL Server Certificate

Issued To

Common Name (CN)	*.gmail.com
Organization (O)	Google Inc
Organizational Unit (OU)	<Not Part Of Certificate>
Serial Number	65:F8:33:2D:6B:CB:67:BC:AD:3A:B0:A9:98:80:28:49

Issued By

Common Name (CN)	Thawte Premium Server CA
Organization (O)	Thawte Consulting cc
Organizational Unit (OU)	Certification Services Division

Validity

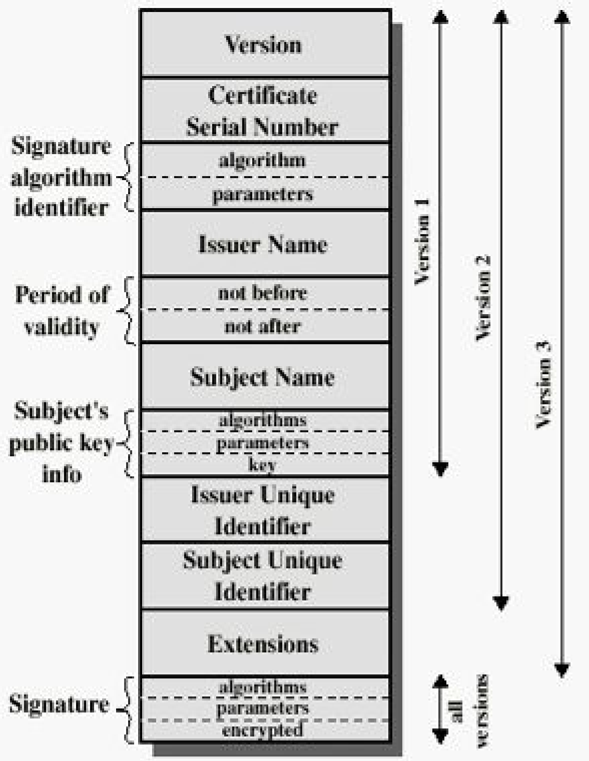
Issued On	9/25/2008
Expires On	9/25/2010

Fingerprints

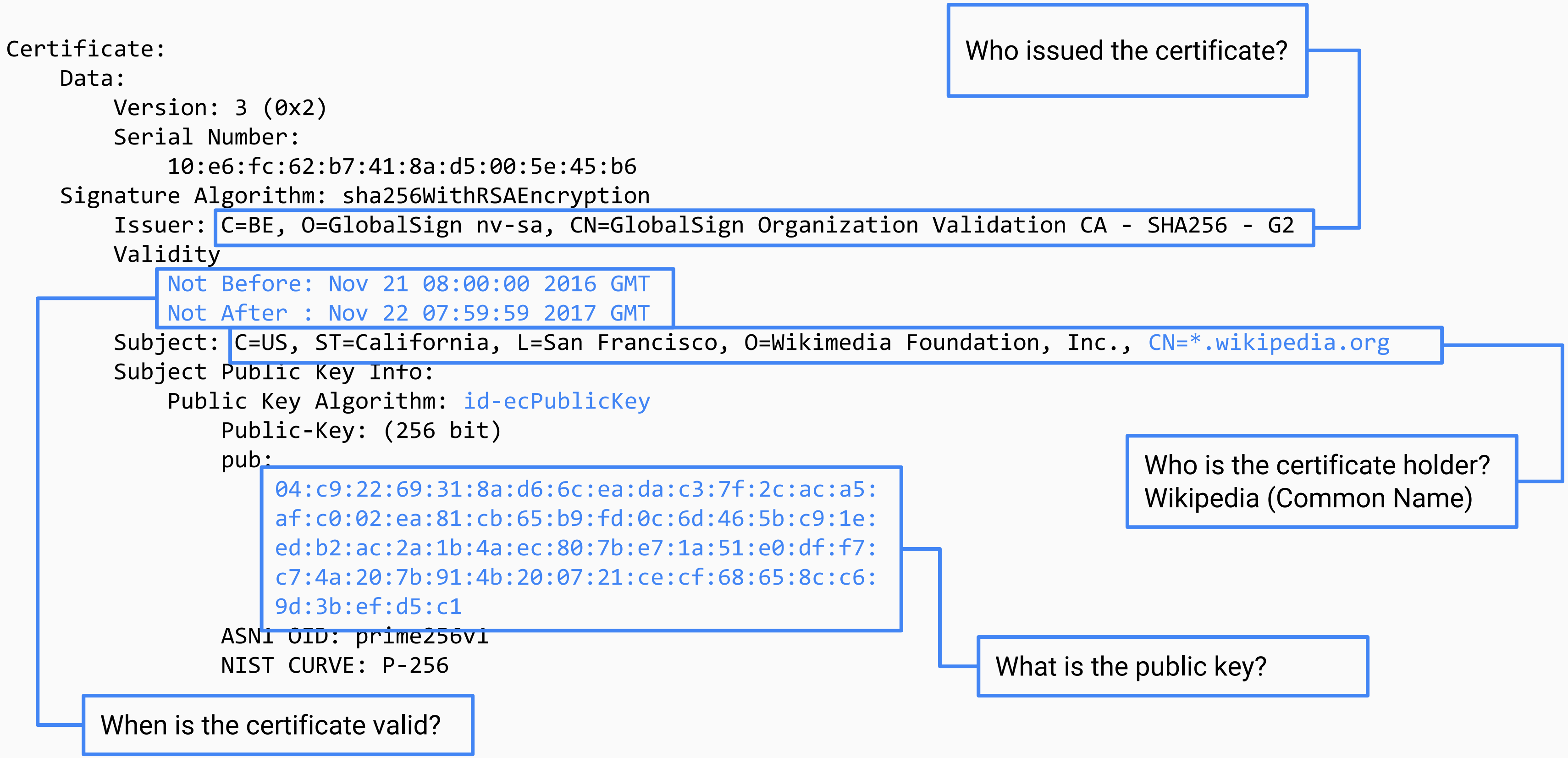
SHA1 Fingerprint	B7:A7:89:34:54:5D:C9:6F:41:FD:A9:3E:41:AF:2B:1D:13:C8:CC:AD
MD5 Fingerprint	55:5F:09:17:24:03:F7:80:2B:B6:90:26:3B:0B:E3:3B

X.509 Authentication Service and Certificate

- Internet standard that defines
  - the format for public key certificates
  - revocation lists
  - certification path validation algorithm
- X.509 is used in
  - IPSec
  - TLS
  - electronic signatures
- Standardized by ITU-T
- Based on the ASN-1 encoding scheme



X.509 certificate example (1/2)



```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            10:e6:fc:62:b7:41:8a:d5:00:5e:45:b6
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=BE, O=GlobalSign nv-sa, CN=GlobalSign Organization Validation CA - SHA256 - G2
        Validity
            Not Before: Nov 21 08:00:00 2016 GMT
            Not After : Nov 22 07:59:59 2017 GMT
        Subject: C=US, ST=California, L=San Francisco, O=Wikimedia Foundation, Inc., CN=*.wikipedia.org
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                   04:c9:22:69:31:8a:d6:6c:ea:da:c3:7f:2c:ac:a5:
                   af:c0:02:ea:81:cb:65:b9:fd:0c:6d:46:5b:c9:1e:
                   ed:b2:ac:2a:1b:4a:ec:80:7b:e7:1a:51:e0:df:f7:
                   c7:4a:20:7b:91:4b:20:07:21:ce:cf:68:65:8c:c6:
                   9d:3b:ef:d5:c1
                ASN1 OID: prime256v1
                NIST CURVE: P-256
```

Who issued the certificate?

Who is the certificate holder?
Wikipedia (Common Name)

What is the public key?

When is the certificate valid?

X.509 certificate example (2/2)

```
    X509v3 extensions:
        X509v3 Key Usage: critical
            Digital Signature, Key Agreement
        Authority Information Access:
            CA Issuers - URI:http://secure.globalsign.com/cacert/gsorganizationvalsha2g2r1.crt
            OCSP - URI:http://ocsp2.globalsign.com/gsorganizationvalsha2g2
        X509v3 Certificate Policies:
            Policy: 1.3.6.1.4.1.4146.1.20
              CPS: https://www.globalsign.com/repository/
            Policy: 2.23.140.1.2.2
        X509v3 Basic Constraints:
            CA:FALSE
        X509v3 CRL Distribution Points:
            Full Name:
              URI:http://crl.globalsign.com/gs/gsorganizationvalsha2g2.crl
        X509v3 Subject Alternative Name:
            DNS:*.wikipedia.org, DNS:*.m.mediawiki.org, DNS:*.m.wikibooks.org, …,  DNS:wikipedia.org
        X509v3 Extended Key Usage:
            TLS Web Server Authentication, TLS Web Client Authentication
        X509v3 Subject Key Identifier:
            28:2A:26:2A:57:8B:3B:CE:B4:D6:AB:54:EF:D7:38:21:2C:49:5C:36
        X509v3 Authority Key Identifier:
            keyid:96:DE:61:F1:BD:1C:16:29:53:1C:C0:CC:7D:3B:83:00:40:E6:1A:7C
Signature Algorithm: sha256WithRSAEncryption
     8b:c3:ed:d1:9d:39:6f:af:40:72:bd:1e:18:5e:30:54:23:35:
     ...
```

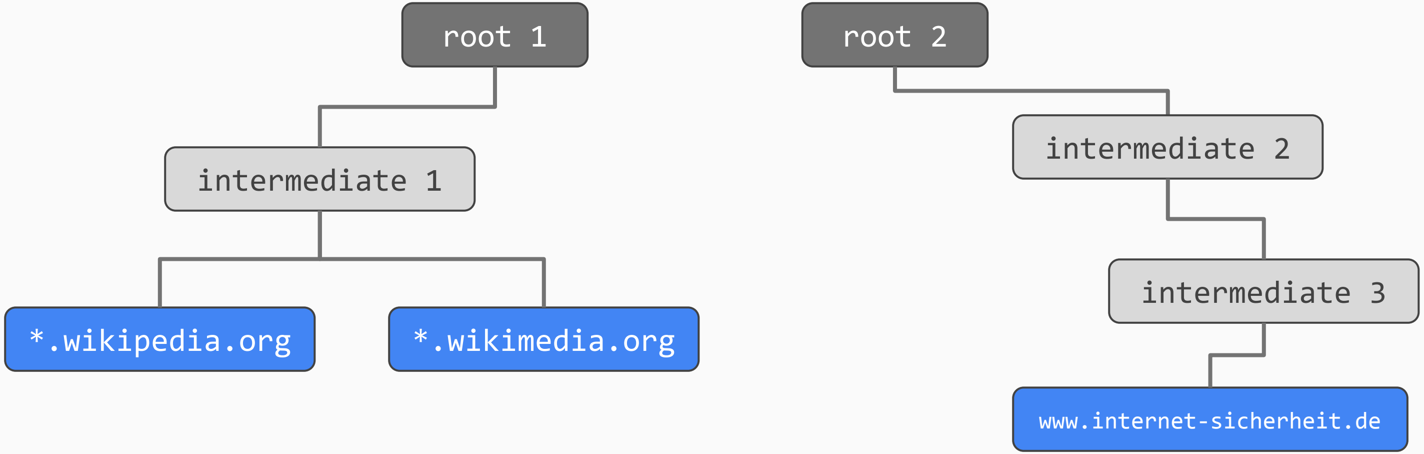
X.509 Common Name

- The Common Name field is important to identify the authenticated peer
- Two variants
  - Specific host name: `www.google.com` Only www.google.com, nobody else
  - Wildcard pattern: `*.google.com` Any host whose name ends in .google.com
- Browsers interpret the star (‘*’) differently
  - Firefox 3: ‘*’ matches any character, including the dot ‘.’
    - www.subdomain.example.com matches *.example.com
  - MS Internet Explorer 7: ‘*’ does not match the dot
    - www.subdomain.example.com will not match *.example.com

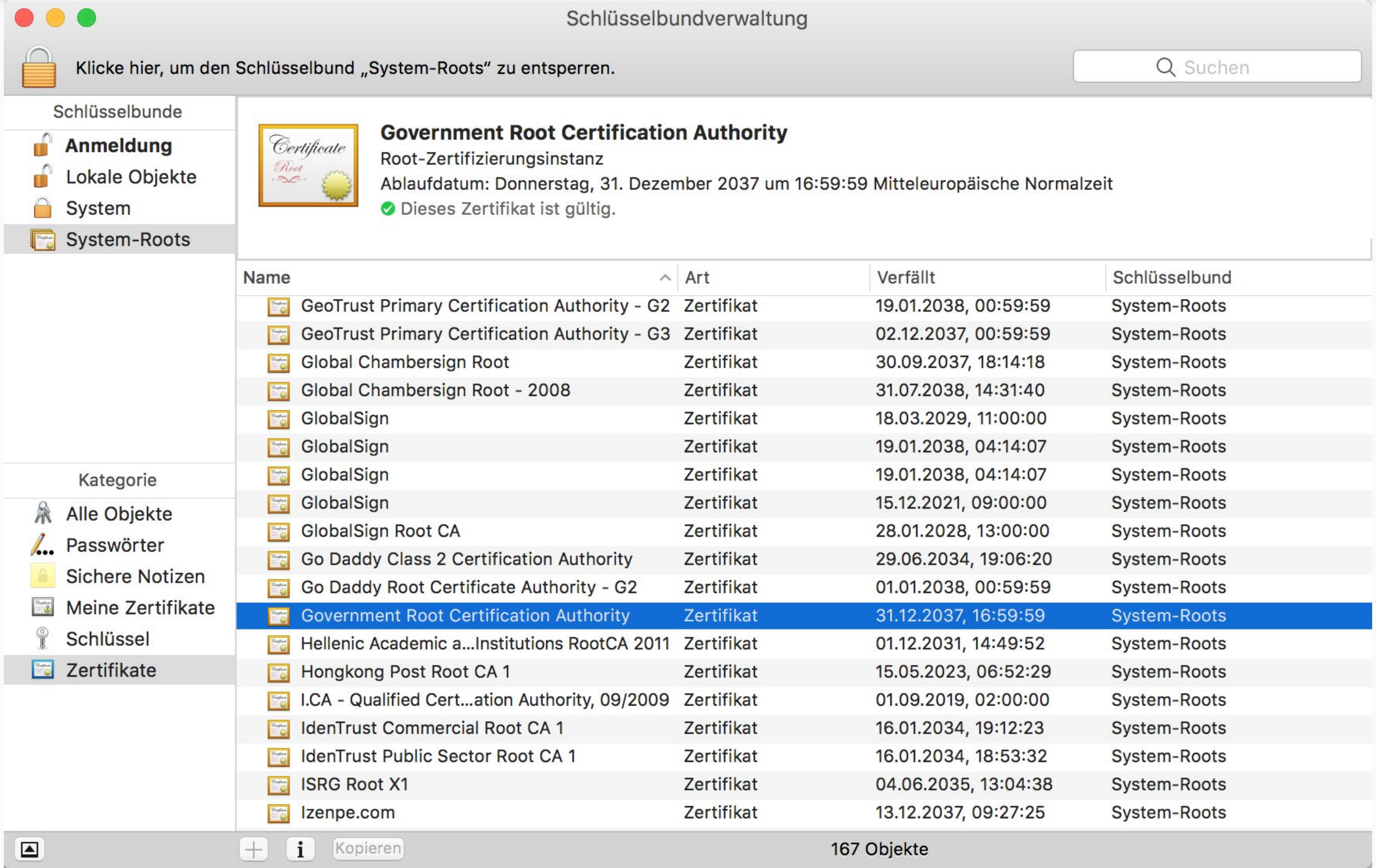
- The Subject Alternative Name extension for X.509 v3 can be used for a certificate that is used by more than just one hostname

Certificate verification chains

- The public key infrastructure (PKI) allows to build a chain from trusted root entities to the issued certificate
- The root entities must be available to a the verifier
  - Typically a set of root CA certificates that are built into the OS or the browser
- Various levels of intermediate certificate entities



TLS root CA certificates



Klicke hier, um den Schlüsselbund „System-Roots" zu entsperren.

Government Root Certification Authority

Root-Zertifizierungsinstanz

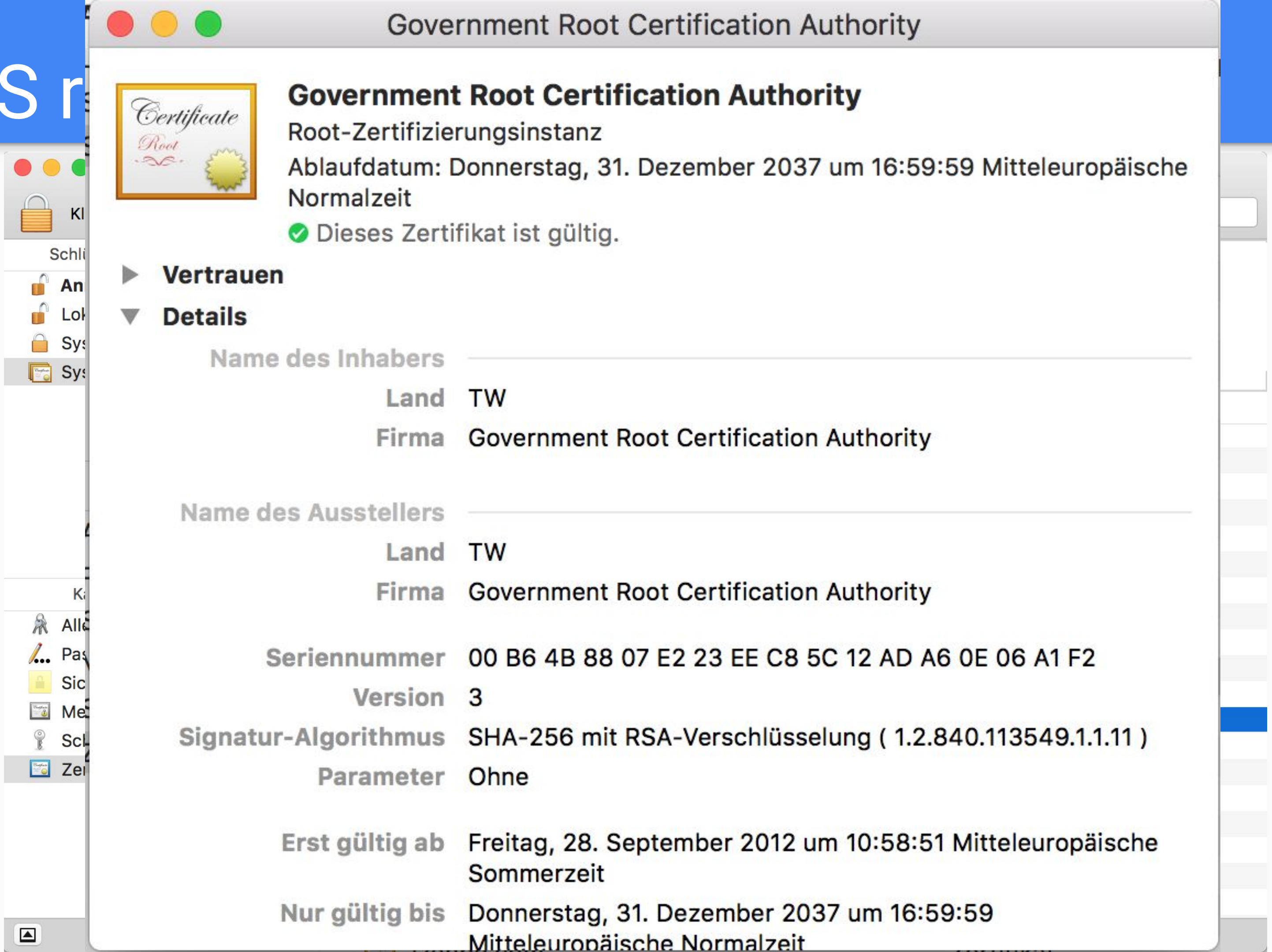
Ablaufdatum: Donnerstag, 31. Dezember 2037 um 16:59:59 Mitteleuropäische Normalzeit

✓ Dieses Zertifikat ist gültig.

Name	Art	Verfällt	Schlüsselbund
GeoTrust Primary Certification Authority - G2	Zertifikat	19.01.2038, 00:59:59	System-Roots
GeoTrust Primary Certification Authority - G3	Zertifikat	02.12.2037, 00:59:59	System-Roots
Global Chambersign Root	Zertifikat	30.09.2037, 18:14:18	System-Roots
Global Chambersign Root - 2008	Zertifikat	31.07.2038, 14:31:40	System-Roots
GlobalSign	Zertifikat	18.03.2029, 11:00:00	System-Roots
GlobalSign	Zertifikat	19.01.2038, 04:14:07	System-Roots
GlobalSign	Zertifikat	19.01.2038, 04:14:07	System-Roots
GlobalSign	Zertifikat	15.12.2021, 09:00:00	System-Roots
GlobalSign Root CA	Zertifikat	28.01.2028, 13:00:00	System-Roots
Go Daddy Class 2 Certification Authority	Zertifikat	29.06.2034, 19:06:20	System-Roots
Go Daddy Root Certificate Authority - G2	Zertifikat	01.01.2038, 00:59:59	System-Roots
Government Root Certification Authority	Zertifikat	31.12.2037, 16:59:59	System-Roots
Hellenic Academic a...Institutions RootCA 2011	Zertifikat	01.12.2031, 14:49:52	System-Roots
Hongkong Post Root CA 1	Zertifikat	15.05.2023, 06:52:29	System-Roots
I.CA - Qualified Cert...ation Authority, 09/2009	Zertifikat	01.09.2019, 02:00:00	System-Roots
IdenTrust Commercial Root CA 1	Zertifikat	16.01.2034, 19:12:23	System-Roots
IdenTrust Public Sector Root CA 1	Zertifikat	16.01.2034, 18:53:32	System-Roots
ISRG Root X1	Zertifikat	04.06.2035, 13:04:38	System-Roots
Izenpe.com	Zertifikat	13.12.2037, 09:27:25	System-Roots

167 Objekte

TLS r



Government Root Certification Authority

Government Root Certification Authority

Root-Zertifizierungsinstanz

Ablaufdatum: Donnerstag, 31. Dezember 2037 um 16:59:59 Mitteleuropäische Normalzeit

✓ Dieses Zertifikat ist gültig.

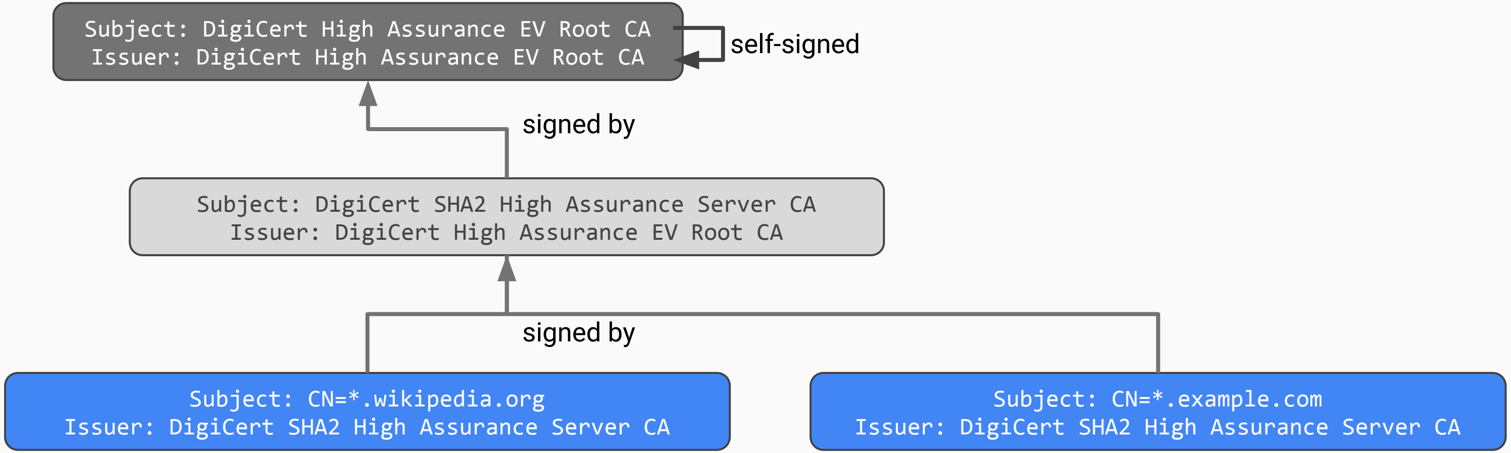
▶ **Vertrauen**

▼ **Details**

Name des Inhabers	
Land	TW
Firma	Government Root Certification Authority
Name des Ausstellers	
Land	TW
Firma	Government Root Certification Authority
Seriennummer	00 B6 4B 88 07 E2 23 EE C8 5C 12 AD A6 0E 06 A1 F2
Version	3
Signatur-Algorithmus	SHA-256 mit RSA-Verschlüsselung (1.2.840.113549.1.1.11)
Parameter	Ohne
Erst gültig ab	Freitag, 28. September 2012 um 10:58:51 Mitteleuropäische Sommerzeit
Nur gültig bis	Donnerstag, 31. Dezember 2037 um 16:59:59 Mitteleuropäische Normalzeit

Certificate verification chain: Example

- The server’s certificate was issued and signed by an intermediate certificate authority (DigiCert SHA2 High Assurance Server CA)
- The intermediate CA certificate was signed by the root CA (DigiCert High Assurance EV Root CA)



Subject: DigiCert High Assurance EV Root CA
Issuer: DigiCert High Assurance EV Root CA

self-signed

signed by

Subject: DigiCert SHA2 High Assurance Server CA
Issuer: DigiCert High Assurance EV Root CA

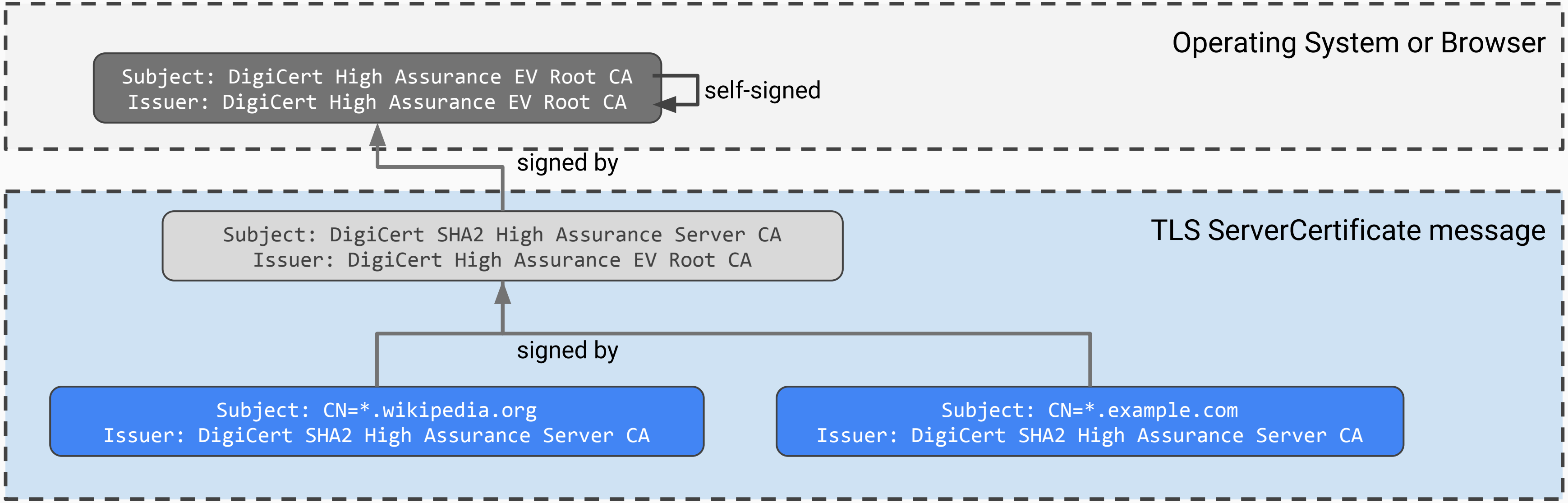
signed by

Subject: CN=*.wikipedia.org
Issuer: DigiCert SHA2 High Assurance Server CA

Subject: CN=*.example.com
Issuer: DigiCert SHA2 High Assurance Server CA

Certificate verification chain: Example

- The server's certificate was issued and signed by an intermediate certificate authority (DigiCert SHA2 High Assurance Server CA)
- The intermediate CA certificate was signed by the root CA (DigiCert High Assurance EV Root CA)



Operating System or Browser

```
Subject: DigiCert High Assurance EV Root CA
Issuer: DigiCert High Assurance EV Root CA
```

TLS ServerCertificate message

```
Subject: DigiCert SHA2 High Assurance Server CA
Issuer: DigiCert High Assurance EV Root CA
```

```
Subject: CN=*.wikipedia.org
Issuer: DigiCert SHA2 High Assurance Server CA
```

```
Subject: CN=*.example.com
Issuer: DigiCert SHA2 High Assurance Server CA
```

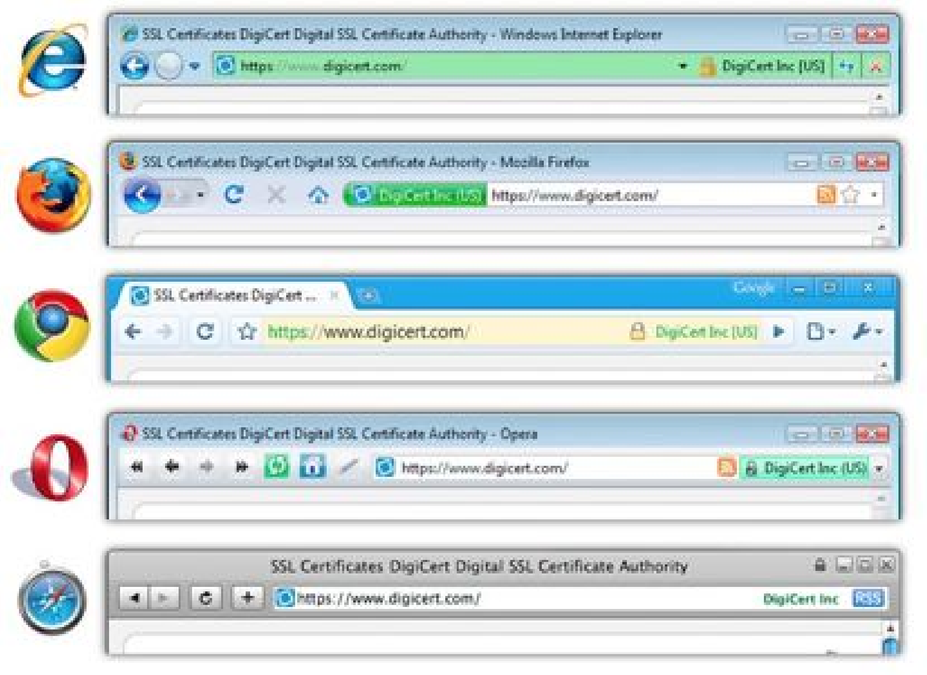
Certificate Revocation, OCSP

- What if a certificate gets compromised?
- Two possibilities
  - It ages out (has a defined end date for the validity)
  - It can be revoked
- Revocation requires that the verifying parties query a service
  - Certificate Revocation List (CRL) can be downloaded
  - Online Certificate Status Protocol (OCSP) is a commonly used protocol to check for revocation of a certificate
- How does the verifier know who to ask?
  - The OCSP endpoint and/or the CRL endpoint is encoded in the certificate

```
X509v3 extensions:
            ...
            Authority Information Access:
                OCSP - URI:http://ocsp2.globalsign.com/gsorganizationvalsha2g2
            ...
            X509v3 CRL Distribution Points:
                Full Name:
                  URI:http://crl.globalsign.com/gs/gsorganizationvalsha2g2.crl
```

Extended Validation (EV) Certificates

- Extended Validation (EV)
  - Stricter issuance and verification policies
  - Protects users against valid certificates of similar-looking (phishing) domains
- Similar-looking domains should not be able to get EV certificates
  - E.g., deutcshe-bank.com
- Requirements
  - Legal identity as well as the operational and physical presence of website owner
  - Applicant is the domain name owner or has exclusive control over domain
  - Confirm the identity and authority of the individuals acting for the website owner
- No wildcard pattern allowed


SSL Certificates DigiCert Digital SSL Certificate Authority - Windows Internet Explorer
https://www.digicert.com/
DigiCert Inc [US]
SSL Certificates DigiCert Digital SSL Certificate Authority - Mozilla Firefox
DigiCert Inc (US)
https://www.digicert.com/
SSL Certificates DigiCert ...
Google
https://www.digicert.com/
DigiCert Inc [US]
SSL Certificates DigiCert Digital SSL Certificate Authority - Opera
https://www.digicert.com/
DigiCert Inc (US)
SSL Certificates DigiCert Digital SSL Certificate Authority
https://www.digicert.com/
DigiCert Inc
RSS

TLS: Session IDs

- The TLS handshake is expensive and does not scale well
- The concept of a session describes a context (including the chosen cipher suite) which can be reused across multiple TLS connections
- The client can provide a session ID to signal the server that it would like to continue an existing session
- Less effort to resume a session or add parallel connections (e.g., in case of HTTPS where multiple connections to the same target are often observed)

TLS 1.3 (in draft)

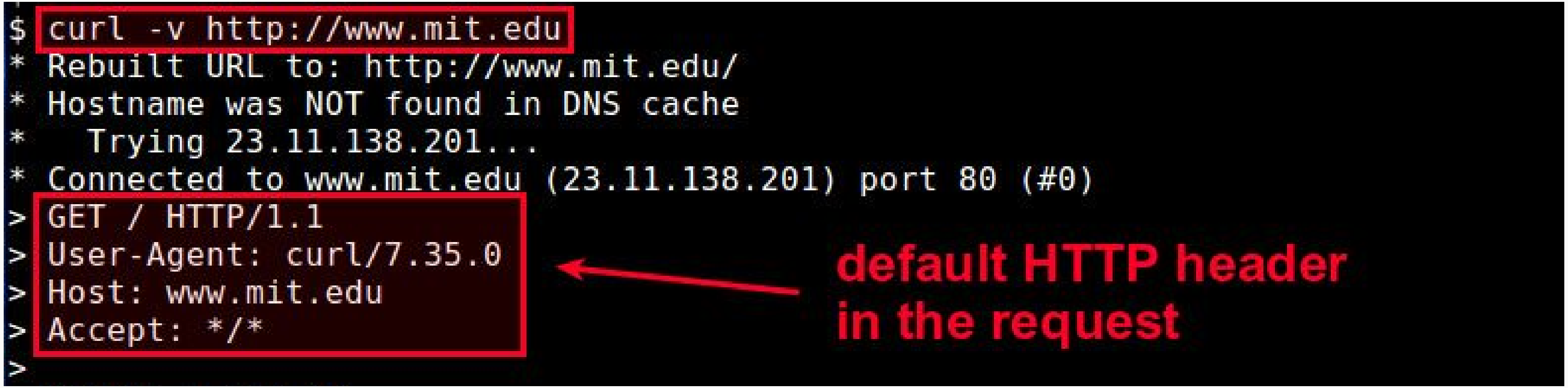
- Remove problematic aspects from older TLS versions
  - Compression
  - RSA key exchange (use DH instead)
  - MAC-then-Encrypt (use Authenticated Encryption instead)
  - No longer allows CBC mode of operation (use GCM instead)
  - Renegotiation
  - Weak ciphers (export ciphers): RC4, MD5, SHA1
- Avoid downgrade attacks
  - Sign elements of the handshake (including the list of supported cipher suites)
  - MITM can not change the list of cipher suites (without breaking the signature)
- Speed up the handshake
  - TLS 1.2: 2 full roundtrip messages
  - TLS 1.3: Make assumptions on supported cipher suites and send key derivation material up front (“just in case you also support AES-GCM, here is some key derivation material”)

HTTPS: TLS with HTTP

- Establish a TLS connection
- Transmit HTTP messages over the TLS channel
- Require that the host name equals the server certificate's common name
- Require a server certificate with a valid certificate chain that originates in a certificate authority (CA) certificate
- Require valid issue timestamp
    - Current date and time > certificate issue date and time
    - Current date and time < certificate validity end date and time
- Enforce certain minimum cipher suites (do not fall back to NULL cipher suite from old SSL/TLS versions)

TLS extensions: Server Name Indication

- Problem: The TLS connection is established before the first application layer message reaches the destination
- Some protocols bind the DNS resolution result to a view on the application
- For example, HTTP has the `Host` header
  - It specifies the host name that was used to resolve the IP address of the destination
  - It controls which view (virtual host) the server uses to deliver responses



```
$ curl -v http://www.mit.edu
* Rebuilt URL to: http://www.mit.edu/
* Hostname was NOT found in DNS cache
*   Trying 23.11.138.201...
* Connected to www.mit.edu (23.11.138.201) port 80 (#0)
> GET / HTTP/1.1
> User-Agent: curl/7.35.0
> Host: www.mit.edu
> Accept: */*
>
```

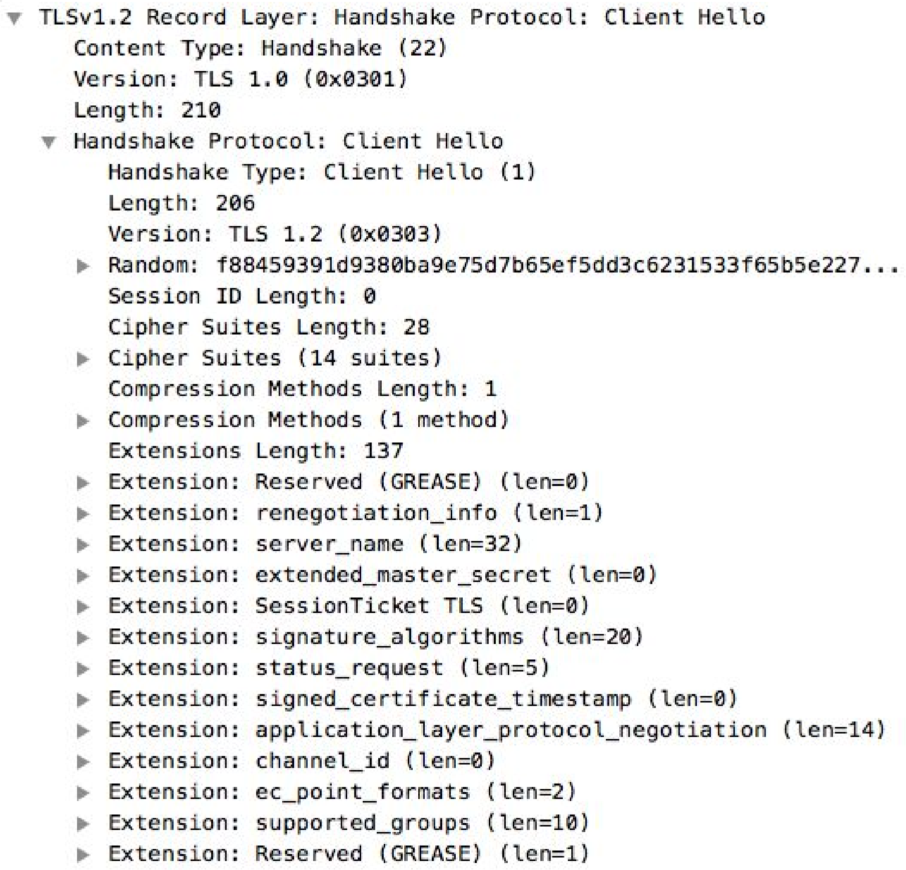
default HTTP header in the request

TLS example traffic

TLS example: Connection to Google APIs

The following example shows how a TLS connection is established between a Chrome browser (version 62) and the host safebrowsing.googleapis.com.

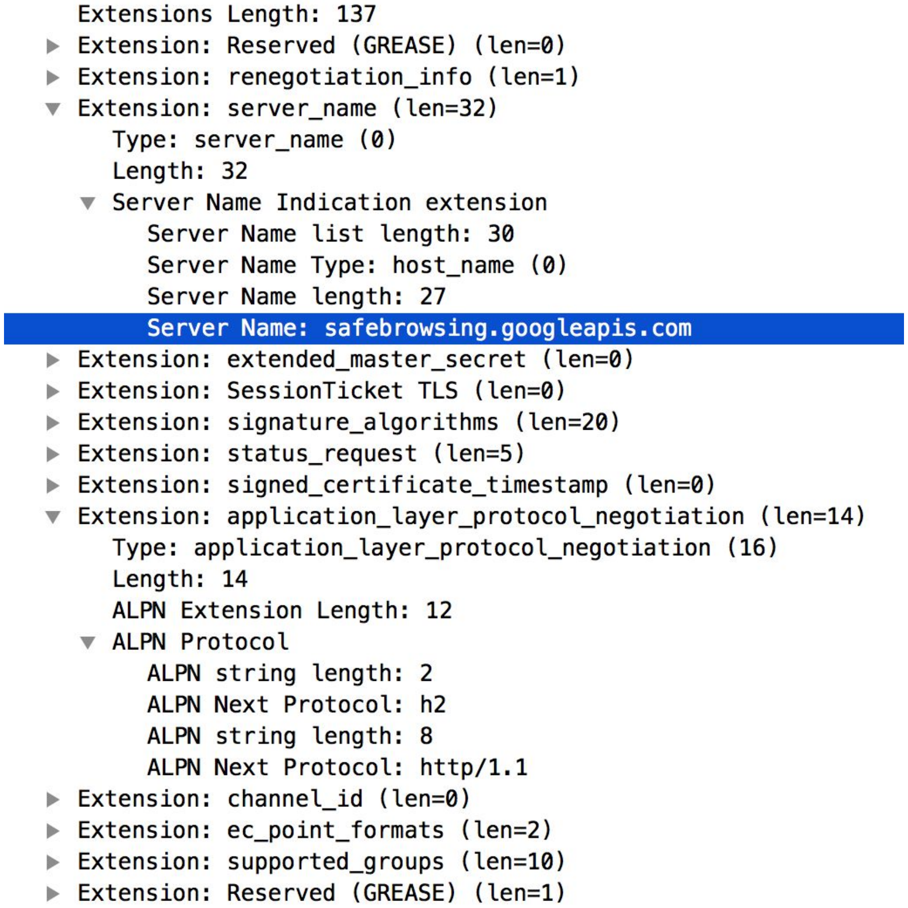
TLS example: ClientHello



```
▼ TLSv1.2 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 210
  ▼ Handshake Protocol: Client Hello
      Handshake Type: Client Hello (1)
      Length: 206
      Version: TLS 1.2 (0x0303)
    ▶ Random: f88459391d9380ba9e75d7b65ef5dd3c6231533f65b5e227...
      Session ID Length: 0
      Cipher Suites Length: 28
    ▶ Cipher Suites (14 suites)
      Compression Methods Length: 1
    ▶ Compression Methods (1 method)
      Extensions Length: 137
    ▶ Extension: Reserved (GREASE) (len=0)
    ▶ Extension: renegotiation_info (len=1)
    ▶ Extension: server_name (len=32)
    ▶ Extension: extended_master_secret (len=0)
    ▶ Extension: SessionTicket TLS (len=0)
    ▶ Extension: signature_algorithms (len=20)
    ▶ Extension: status_request (len=5)
    ▶ Extension: signed_certificate_timestamp (len=0)
    ▶ Extension: application_layer_protocol_negotiation (len=14)
    ▶ Extension: channel_id (len=0)
    ▶ Extension: ec_point_formats (len=2)
    ▶ Extension: supported_groups (len=10)
    ▶ Extension: Reserved (GREASE) (len=1)
```

- TLS record has version TLS 1.0
- ClientHello signals the client can speak TLS 1.2
- Random: client-generated nonce
- Offers 14 cipher suites
- Offers 1 compression algorithm
- Supports 13 TLS extensions including
  - Server Name Indication (server_name)
  - Elliptic curve point formats
  - Application layer protocol negotiation (Signals that the client wants to speak http/1.1)

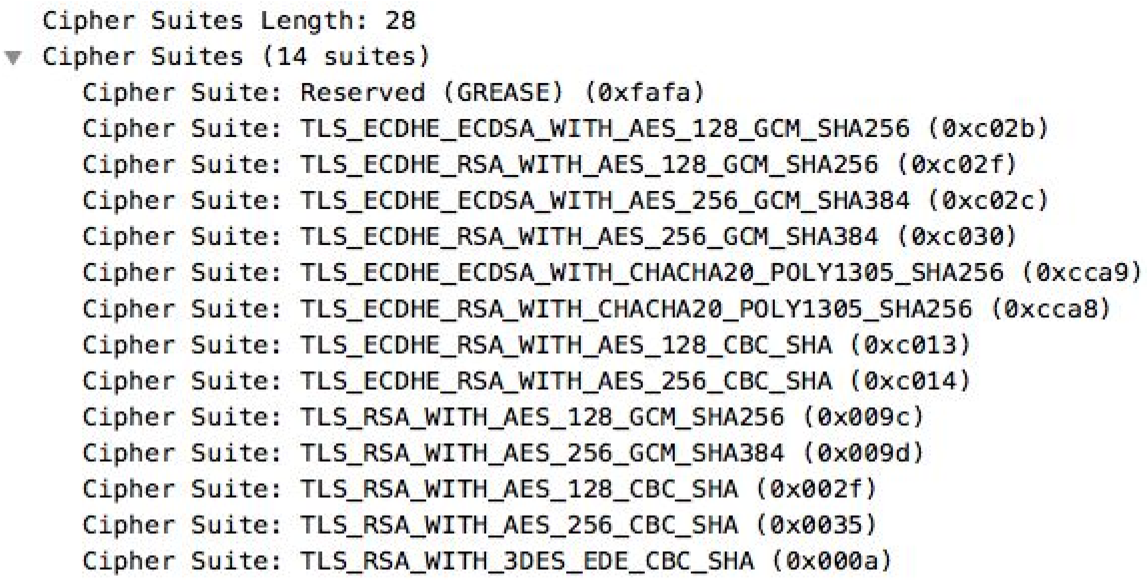
TLS example: ClientHello (SNI)



```
    Extensions Length: 137
  ▶ Extension: Reserved (GREASE) (len=0)
  ▶ Extension: renegotiation_info (len=1)
  ▼ Extension: server_name (len=32)
      Type: server_name (0)
      Length: 32
    ▼ Server Name Indication extension
        Server Name list length: 30
        Server Name Type: host_name (0)
        Server Name length: 27
        Server Name: safebrowsing.googleapis.com
  ▶ Extension: extended_master_secret (len=0)
  ▶ Extension: SessionTicket TLS (len=0)
  ▶ Extension: signature_algorithms (len=20)
  ▶ Extension: status_request (len=5)
  ▶ Extension: signed_certificate_timestamp (len=0)
  ▼ Extension: application_layer_protocol_negotiation (len=14)
      Type: application_layer_protocol_negotiation (16)
      Length: 14
      ALPN Extension Length: 12
    ▼ ALPN Protocol
        ALPN string length: 2
        ALPN Next Protocol: h2
        ALPN string length: 8
        ALPN Next Protocol: http/1.1
  ▶ Extension: channel_id (len=0)
  ▶ Extension: ec_point_formats (len=2)
  ▶ Extension: supported_groups (len=10)
  ▶ Extension: Reserved (GREASE) (len=1)
```

- Supports 13 TLS extensions including
  - Server Name Indication (server_name) `safebrowsing.googleapis.com`
  - Elliptic curve point formats
  - Application layer protocol negotiation (Signals that the client wants to speak http/1.1)

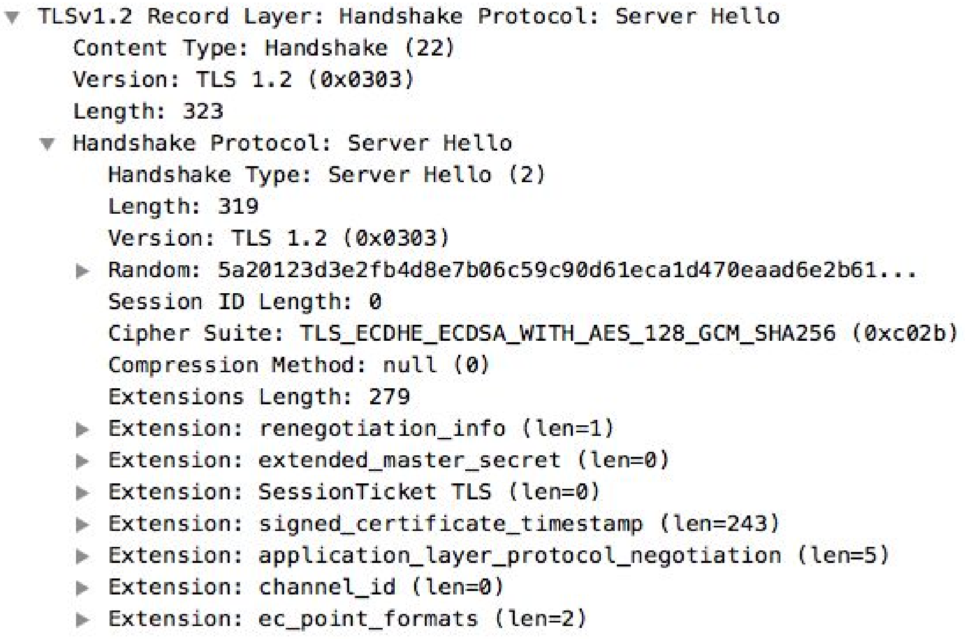
TLS example: ClientHello: Cipher suites



```
  Cipher Suites Length: 28
▼ Cipher Suites (14 suites)
    Cipher Suite: Reserved (GREASE) (0xfafa)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9)
    Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
    Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)
    Cipher Suite: TLS_RSA_WITH_AES_256_GCM_SHA384 (0x009d)
    Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
    Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
    Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)
```

- Offers 14 cipher suites
- Each cipher suite is encoded in two bytes
- The total length is 14*2 = 28 bytes
- The list is sorted by cryptographic strength

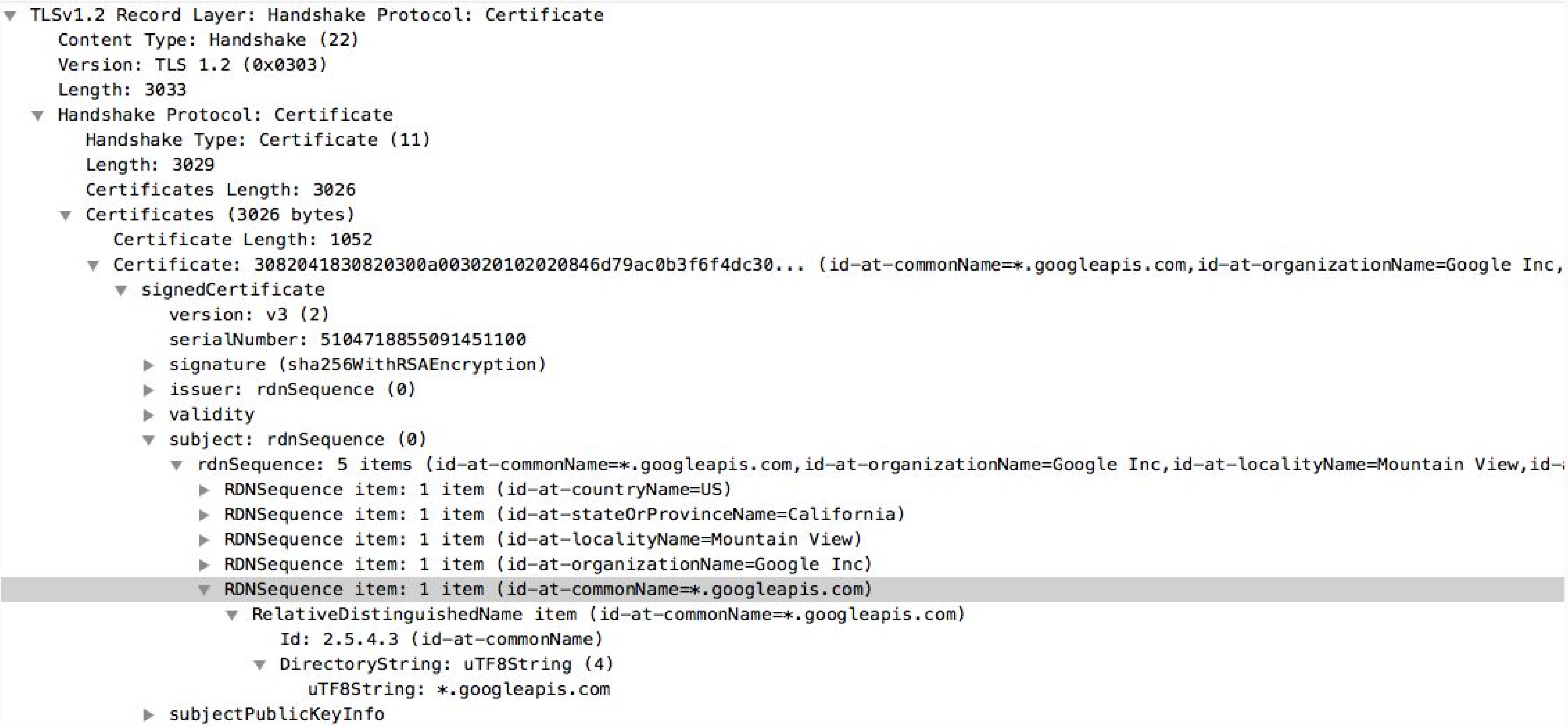
TLS example: ServerHello



```
▼ TLSv1.2 Record Layer: Handshake Protocol: Server Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 323
  ▼ Handshake Protocol: Server Hello
      Handshake Type: Server Hello (2)
      Length: 319
      Version: TLS 1.2 (0x0303)
    ▶ Random: 5a20123d3e2fb4d8e7b06c59c90d61eca1d470eaad6e2b61...
      Session ID Length: 0
      Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
      Compression Method: null (0)
      Extensions Length: 279
    ▶ Extension: renegotiation_info (len=1)
    ▶ Extension: extended_master_secret (len=0)
    ▶ Extension: SessionTicket TLS (len=0)
    ▶ Extension: signed_certificate_timestamp (len=243)
    ▶ Extension: application_layer_protocol_negotiation (len=5)
    ▶ Extension: channel_id (len=0)
    ▶ Extension: ec_point_formats (len=2)
```

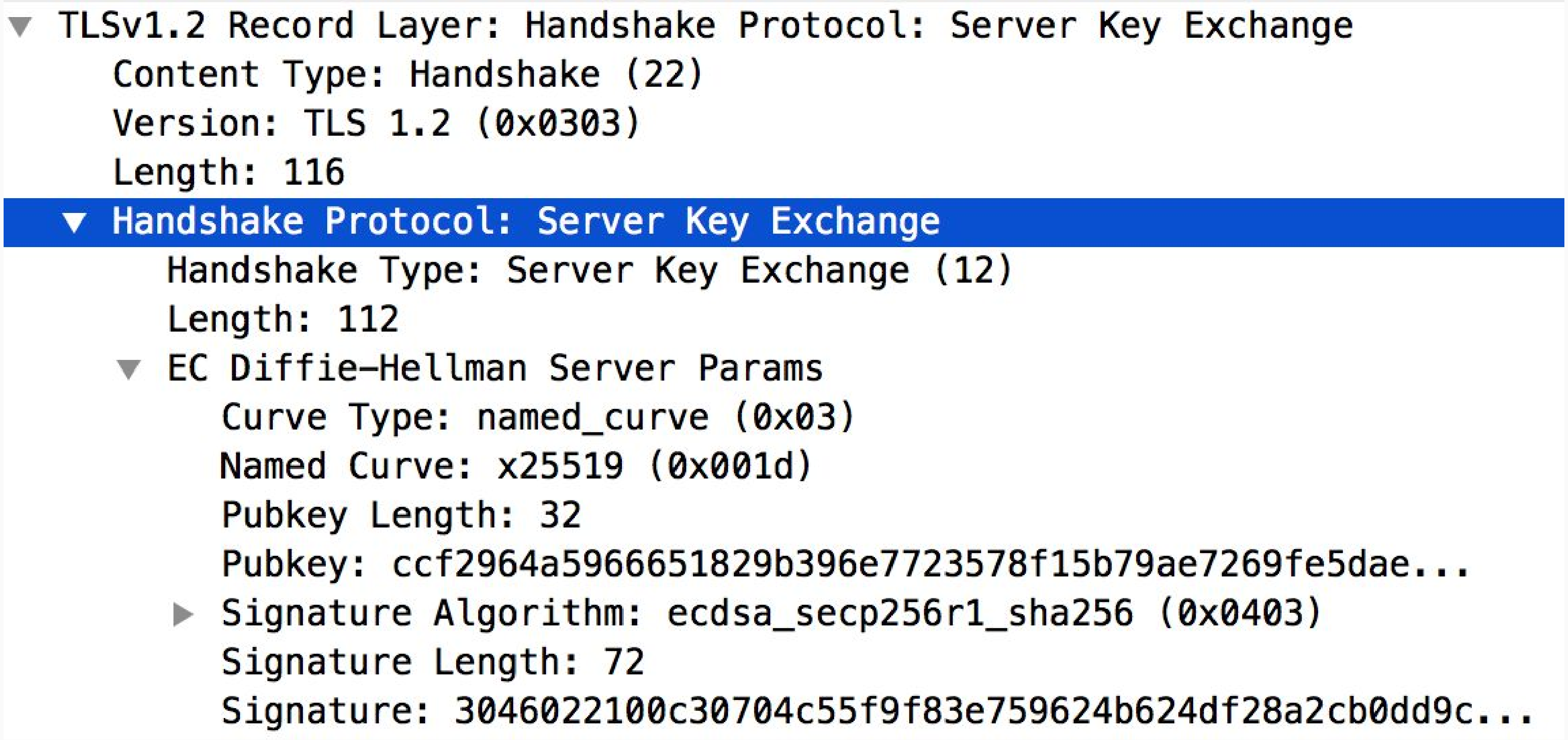
- TLS record has version TLS 1.2
- ServerHello signals TLS 1.2 was chosen
- Random: server-generated nonce
- Chooses cipher suite
- No compression
- Does not set a session ID
- Responds to 7 TLS extensions

TLS example: Server certificate



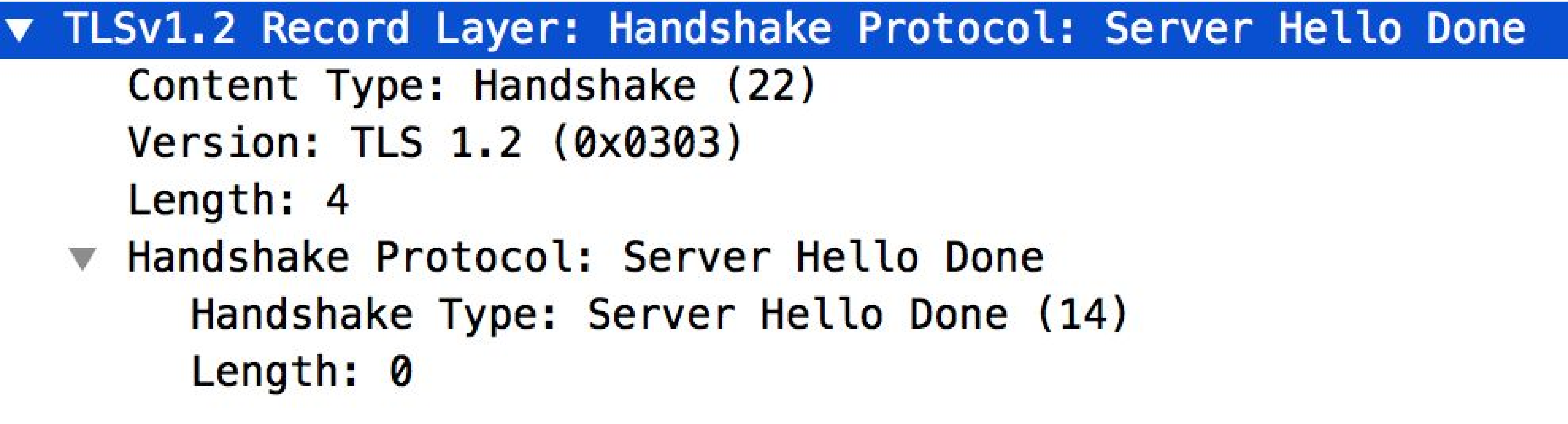
```
▼ TLSv1.2 Record Layer: Handshake Protocol: Certificate
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 3033
  ▼ Handshake Protocol: Certificate
      Handshake Type: Certificate (11)
      Length: 3029
      Certificates Length: 3026
    ▼ Certificates (3026 bytes)
        Certificate Length: 1052
      ▼ Certificate: 3082041830820300a003020102020846d79ac0b3f6f4dc30... (id-at-commonName=*.googleapis.com,id-at-organizationName=Google Inc,
        ▼ signedCertificate
            version: v3 (2)
            serialNumber: 5104718855091451100
          ▶ signature (sha256WithRSAEncryption)
          ▶ issuer: rdnSequence (0)
          ▶ validity
          ▼ subject: rdnSequence (0)
            ▼ rdnSequence: 5 items (id-at-commonName=*.googleapis.com,id-at-organizationName=Google Inc,id-at-localityName=Mountain View,id-
              ▶ RDNSequence item: 1 item (id-at-countryName=US)
              ▶ RDNSequence item: 1 item (id-at-stateOrProvinceName=California)
              ▶ RDNSequence item: 1 item (id-at-localityName=Mountain View)
              ▶ RDNSequence item: 1 item (id-at-organizationName=Google Inc)
              ▼ RDNSequence item: 1 item (id-at-commonName=*.googleapis.com)
                ▼ RelativeDistinguishedName item (id-at-commonName=*.googleapis.com)
                    Id: 2.5.4.3 (id-at-commonName)
                  ▼ DirectoryString: uTF8String (4)
                      uTF8String: *.googleapis.com
          ▶ subjectPublicKeyInfo
```

TLS example: Server Key Exchange



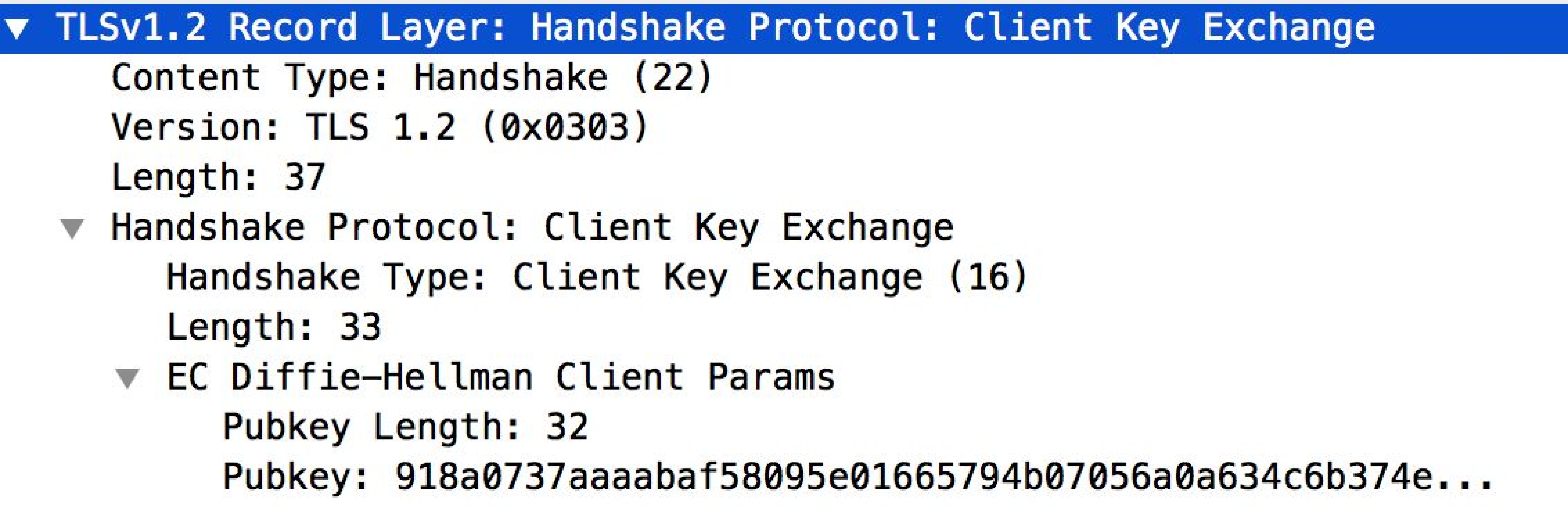
```
▼ TLSv1.2 Record Layer: Handshake Protocol: Server Key Exchange
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 116
  ▼ Handshake Protocol: Server Key Exchange
      Handshake Type: Server Key Exchange (12)
      Length: 112
    ▼ EC Diffie-Hellman Server Params
        Curve Type: named_curve (0x03)
        Named Curve: x25519 (0x001d)
        Pubkey Length: 32
        Pubkey: ccf2964a5966651829b396e7723578f15b79ae7269fe5dae...
      ▶ Signature Algorithm: ecdsa_secp256r1_sha256 (0x0403)
        Signature Length: 72
        Signature: 3046022100c30704c55f9f83e759624b624df28a2cb0dd9c...
```

TLS example: Server Hello Done



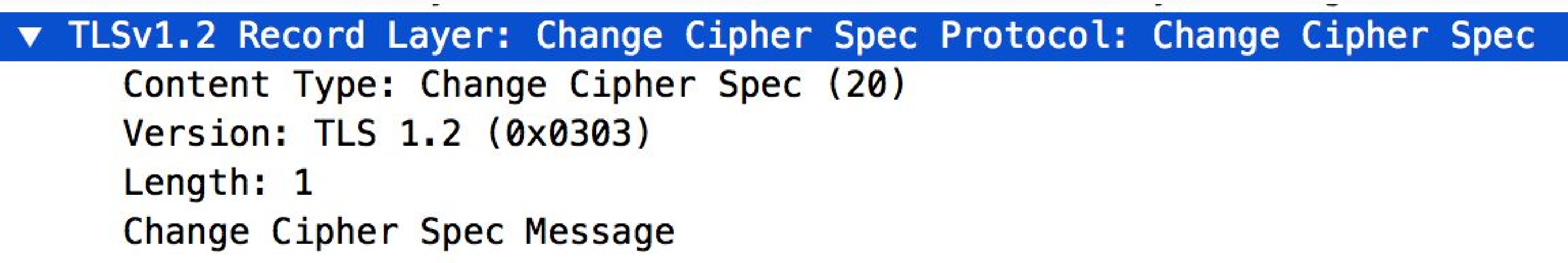
```
▼ TLSv1.2 Record Layer: Handshake Protocol: Server Hello Done
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 4
  ▼ Handshake Protocol: Server Hello Done
      Handshake Type: Server Hello Done (14)
      Length: 0
```

TLS example: Client Key Exchange



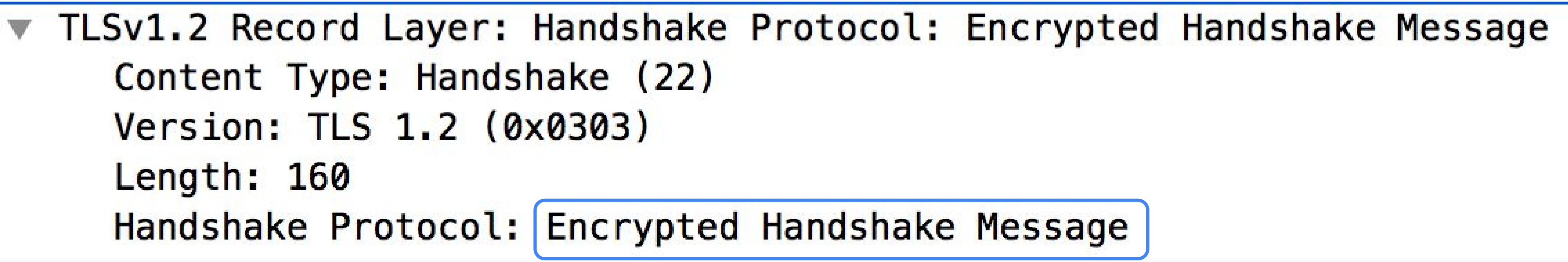
```
▼ TLSv1.2 Record Layer: Handshake Protocol: Client Key Exchange
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 37
  ▼ Handshake Protocol: Client Key Exchange
      Handshake Type: Client Key Exchange (16)
      Length: 33
    ▼ EC Diffie-Hellman Client Params
        Pubkey Length: 32
        Pubkey: 918a0737aaaabaf58095e01665794b07056a0a634c6b374e...
```

TLS example: Change Cipher Spec



```
▼ TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
      Content Type: Change Cipher Spec (20)
      Version: TLS 1.2 (0x0303)
      Length: 1
      Change Cipher Spec Message
```

TLS example: Finished

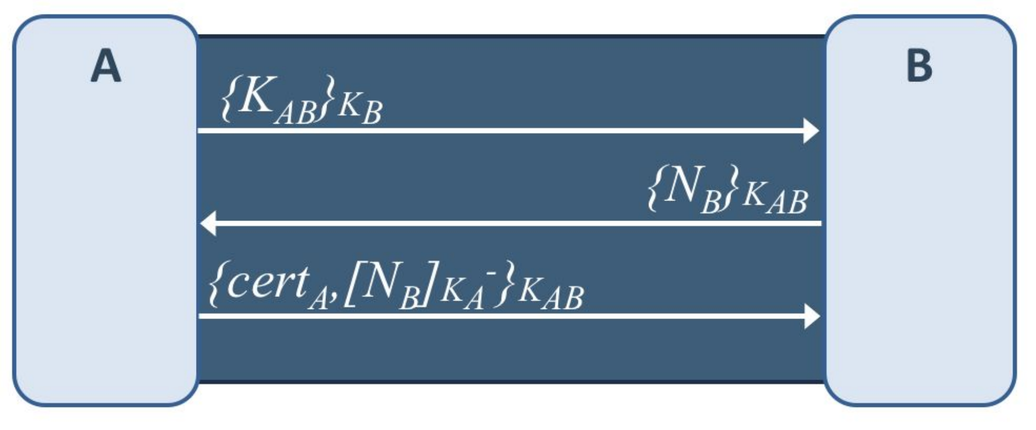


```
▼ TLSv1.2 Record Layer: Handshake Protocol: Encrypted Handshake Message
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 160
    Handshake Protocol: Encrypted Handshake Message
```

Attacks against TLS

Weakness in SSL 1.0 (1/2)

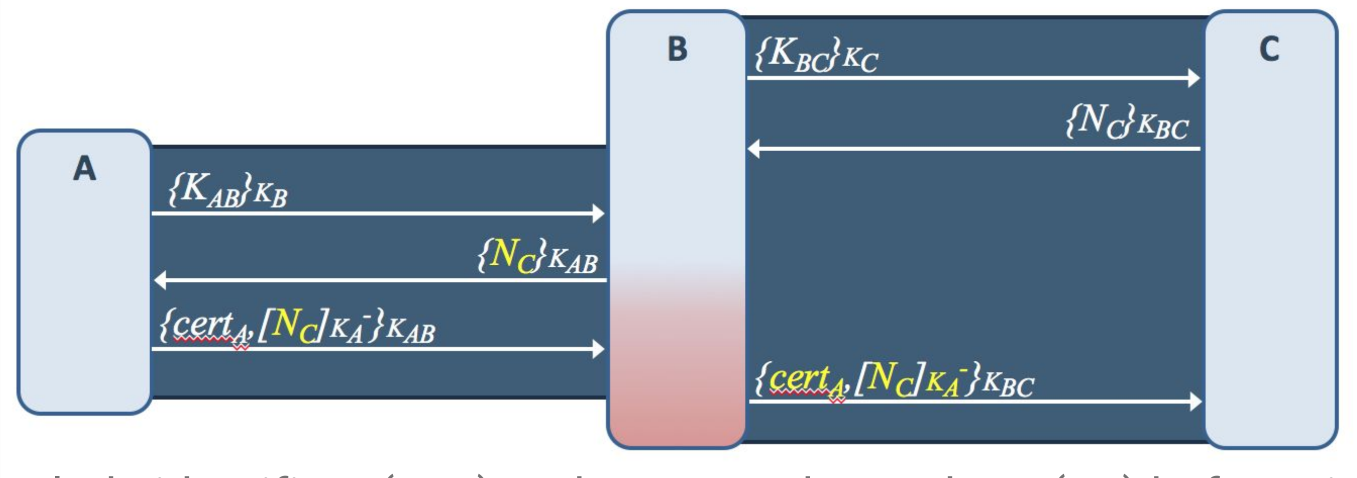
- A chooses session key, B then authenticates A
    - Assumption: A and B know each other's public keys or can obtain certificates
    - A sends B a session key K_{AB} encrypted under B's public key
    - B challenges A with random nonce N_B encrypted under session key
    - A authenticates against B by reflecting a signed version of the decrypted nonce



{ }	Encryption
[ ]	Signature

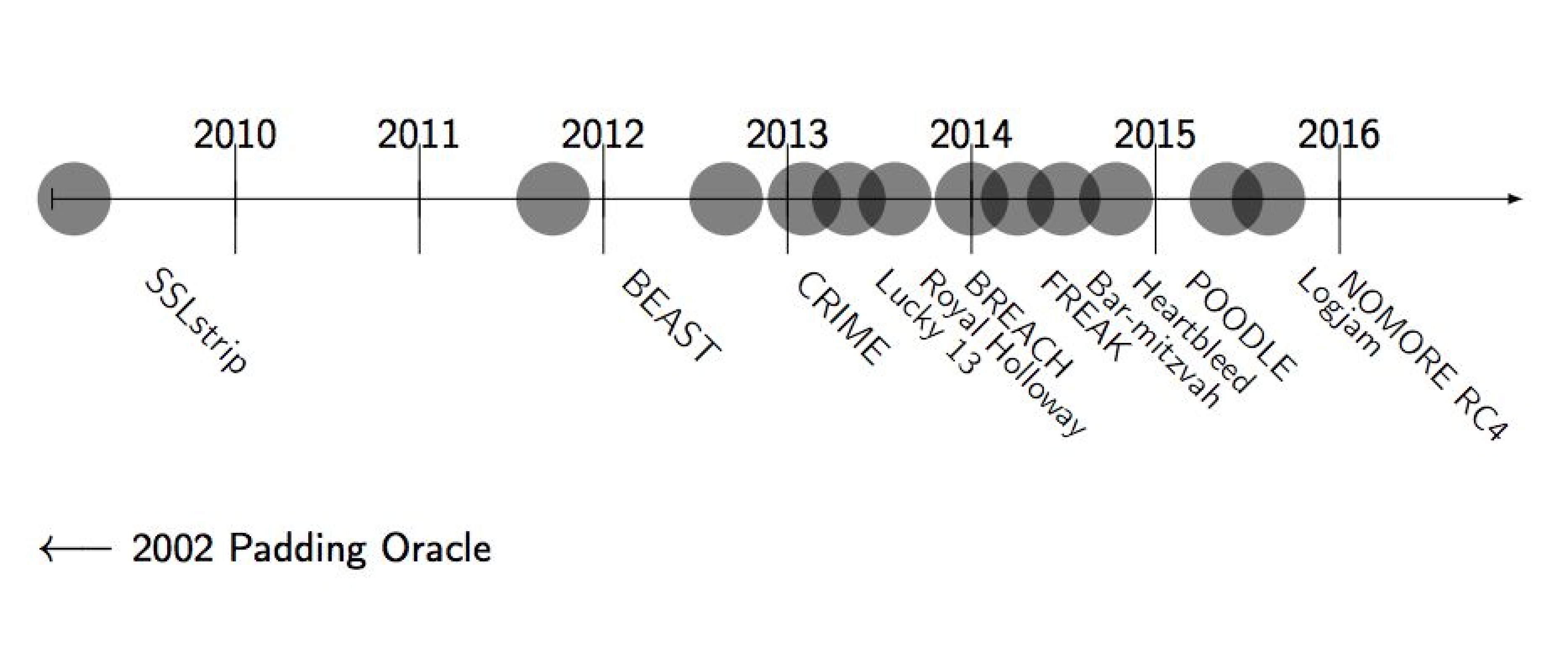
Weakness in SSL 1.0 (2/2)

- Attack: B can impersonate A in a session with C
  - A will blindly sign any nonce without binding result to identifier
  - C challenges B with a nonce, but B will forward this challenge to A
  - A can replay nonce signed by A to other session with C



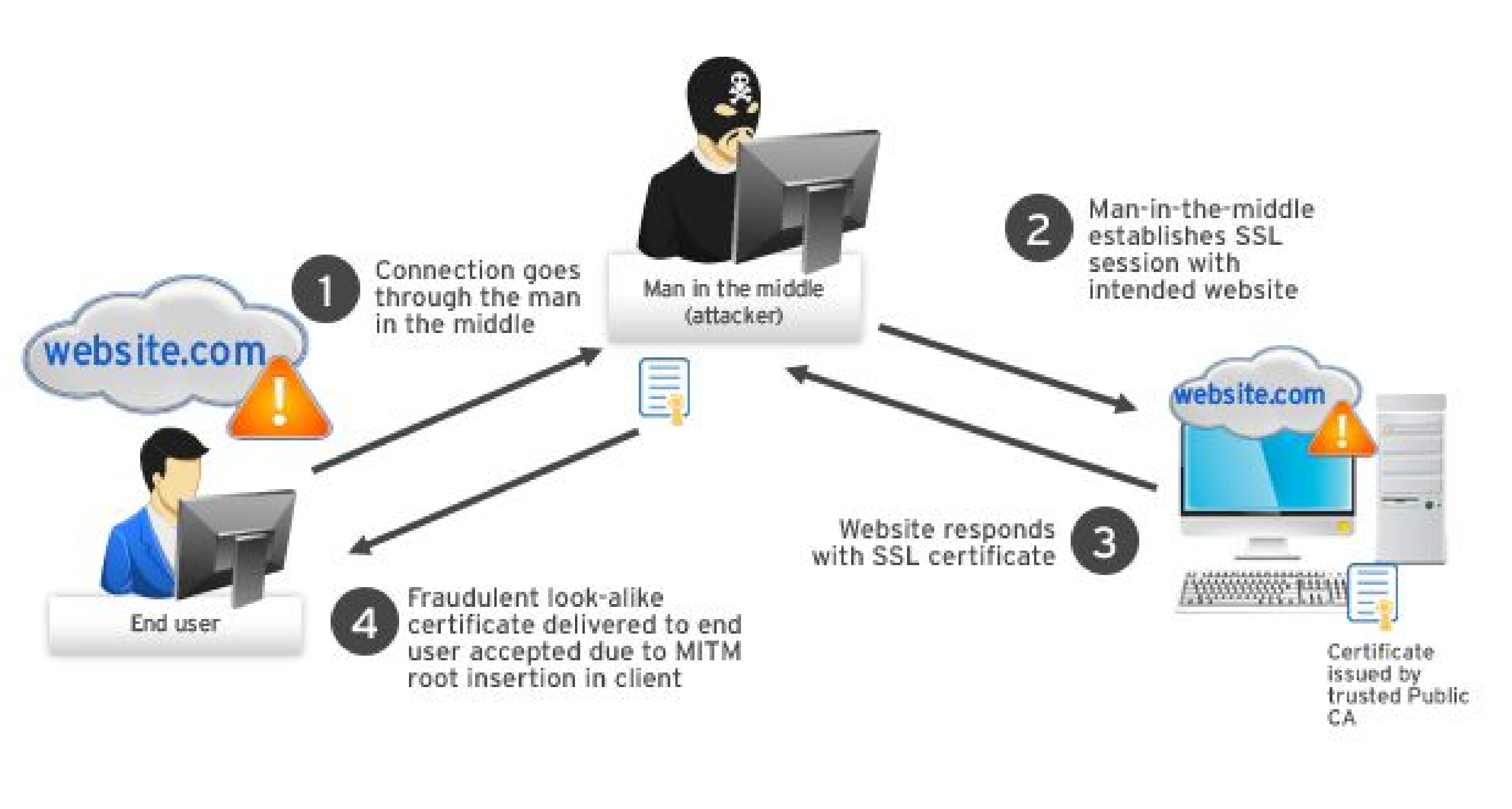
- Fix: Include identifiers (A, B) and a nonce chosen by A (N_A) before signing
- Lesson learned: Do not include signing oracles. Bind names before signing.

Attacks against TLS/SSL: Timeline



Source: Ruben Niederhagen, Attacks on TLS/SSL, Timeline

Man-in-the-middle attack against TLS



1 Connection goes through the man in the middle
Man in the middle (attacker)
2 Man-in-the-middle establishes SSL session with intended website
website.com
End user
website.com
3 Website responds with SSL certificate
4 Fraudulent look-alike certificate delivered to end user accepted due to MITM root insertion in client
Certificate issued by trusted Public CA

http://blog.trendmicro.com/trendlabs-security-intelligence/extended-validation-certificates-warning-against-mitm-attacks/

Man-in-the-middle attack against TLS

- The end user and the server do not know each other beforehand
- The attacker places herself in the middle between the victim and the target ⇒ hence the name of the attack

- What prevents a successful man-in-the-middle attack?

Man-in-the-middle attack against TLS

- The end user and the server do not know each other beforehand
- The attacker places herself in the middle between the victim and the target
  ⇒ hence the name of the attack

- What prevents a successful man-in-the-middle attack?
  - The authentication via certificates!
  - Assuming that the involved certificates are valid
  - No compromised CA
  - No attacker-controlled CA certificate on the victim system

Popular attacks against TLS

- Padding Oracle Attacks
  - Initially discovered in 2002, practical attack shown in 2013 “Lucky 13”: Timing side channel attack against ciphers in CBC mode of operation, Royal Holloway London
  - Underlying problem: MAC-then-encrypt
  - POODLE attack in 2014: Vulnerability in SSL 3.0 (1999)
  - ⇒ Use Encrypt-then-MAC or even better: Use Authenticated Encryption ciphers/modes
- BEAST (Browser Exploit Against SSL/TLS)
  - Described in 2004, practical attack shown in 2011
  - Predictable initialization vectors (IV) in SSL 3.0 and TLS 1.0
  - Chosen-plaintext attack can be used to derive parts of the plaintext
  - ⇒ Randomize IVs
- Compression attacks
  - Compression is a bad idea in TLS
  - Redundancy in the plaintext will compress better, hence the ciphertext will be shorter
  - Practical attack shown in 2012 as Compression Ratio Info-leak Made Easy (CRIME)
  - ⇒ No more compression in TLS 1.3

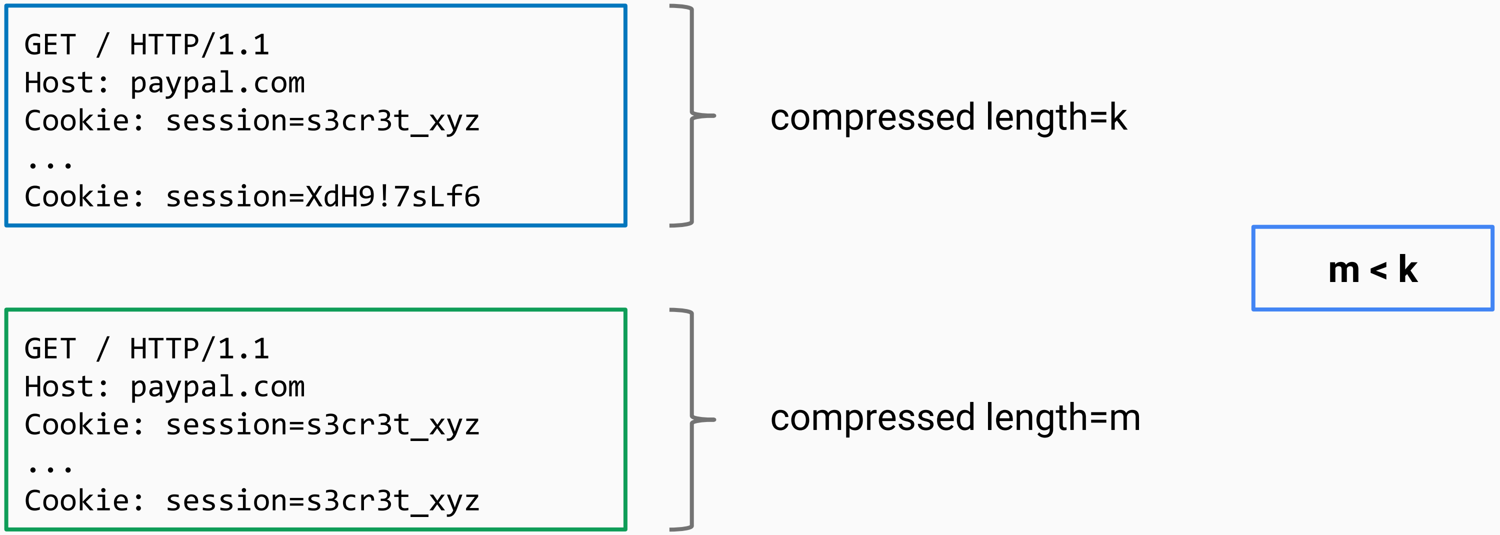
CRIME: Compression Ratio Info-leak Made Easy (2012)

- Leverages compression side effects to break TLS
- Idea: Try to produce an exact copy of an initial secret value in the same packet
- Typical secret of interest: the Cookie header

```
GET / HTTP/1.1
Host: paypal.com
Cookie: session=s3cr3t_xyz
...
```

CRIME: Compression Ratio Info-leak Made Easy (2012)

- Leverages compression side effects to break TLS
- Idea: Try to produce an exact copy of an initial secret value in the same packet
- Typical secret of interest: the Cookie header



```
GET / HTTP/1.1
Host: paypal.com
Cookie: session=s3cr3t_xyz
...
Cookie: session=XdH9!7sLf6
```

compressed length=k

m < k

```
GET / HTTP/1.1
Host: paypal.com
Cookie: session=s3cr3t_xyz
...
Cookie: session=s3cr3t_xyz
```

compressed length=m

CRIME: Compression Ratio Info-leak Made Easy (2012)

- Mitigation
  - Disable compression!
  - TLS 1.3 will no longer allow compression
  - Prevent attacker-controlled payload in the victim’s context
    - Class of attacks: cross-site request forgery (CSRF)

- BREACH: Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext, 2013
- BREACH is an attack similar to CRIME
  - The attack leverages HTTP compression instead of TLS compression
  - Likely also works with TLS 1.3
  - Again: disable compression

TLS attack: Compromised Root CA

- If a root CA is compromised, the security of the entire system is at stake

- Impact of compromised root CA
  - A root CA can issue certificates for *every* domain
  - A root CA can issue code signing certificates for programs

- Significant incidents
  - Comodo, 2011: Compromised CA
  - DigiNotar, 2011: Compromised CA issued certificates to Iranian actors
  - Trustwave subordinate, 2012: Used for interception of TLS connections

Compromised Root CA: Comodo

- Comodo accepts certificate orders submitted through resellers
  - Reseller uses a program to authenticate to Comodo and submit an order with a domain name and public key, Comodo automatically issues a certificate for this site
- A hacker broke into instantSSL.it and globalTrust.it resellers, decompiled their certificate issuance program, learned the credentials of their reseller account and how to use Comodo API
  - username: gtadmin, password: globaltrust
- The hacker wrote his own program for submitting orders and obtaining Comodo certificates
- Successfully retrieved certificates for various sites
  - mail.google.com, login.live.com, login.yahoo.com, login.skype.com, addons.mozilla.org

Compromised Root CA: DigiNotar

- DigiNotar, 2011: Compromised CA issued certificates to Iranian actors
    - PKIoverheid: Dutch national public key infrastructure (PKI)
    - Dutch government took over
    - More than 500 fake certificates signed by DigiNotar found
    - Including a wildcard certificate: *.google.com
    - Likely used for surveillance of Iranian dissidents
- What caused the incident?
    - Questionable security procedures at DigiNotar
        - `The audit [...] revealed that DigiNotar had lacked basic security safeguards, such as strong passwords, anti-virus protection, and up-to-date software patches.`
        - Core certificates in a centralized store on a Windows domain
        - Admin-Passwort: "Pr0d@dm1n" *Seriously?!*
    - In total, 531 fraudulent certificates issued

TLS attack: Compromised Root CA

- If a root CA is compromised, the security of the entire system is at stake

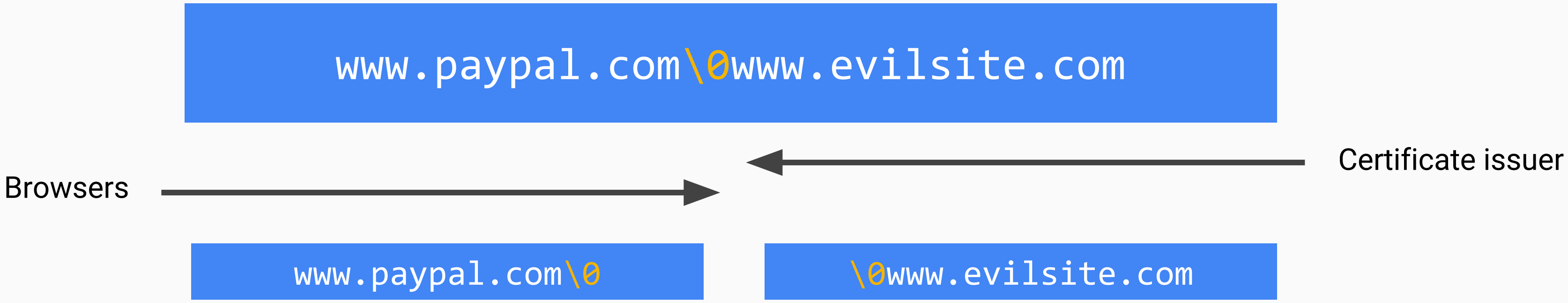
- Mitigation thoughts
  - Limit the geographic area/country/language where a root CA may issue certificates
  - Limit the TLD for which certificates may be issued
  - Basically all of these approaches build on an artificial segmentation of the certificate name space
  - HTTP Public Key Pinning (HPKP), RFC7469
    - Pin public keys to a website
    - The browser learns these and can compare on subsequent visits

```
Public-Key-Pins: max-age=2592000;
pin-sha256="E9CZ9INDbd+2eRQozYqqbQ2yXLVKB9+xcprMF+44U1g=";
pin-sha256="LPJNul+wow4m6DsqxbninhsWHlwfp0JecwQzYpOLmCQ=";
report-uri="http://example.com/pkp-report"
```

Podcast on the DigiNotar breach: https://darknetdiaries.com/episode/3/

TLS attack: Common name parsing issues

- Problem: A NULL byte in the common name (Marlinspike, 2009)
- Requirements
  - Attacker-controlled site: www.evilsite.com
  - Target site www.paypal.com
- Certificate issuer parses the common name field of an X.509 certificate differently than a browser
  - Certificate issuer: Starts parsing from the end
  - Browser: Starts parsing from the start



```
www.paypal.com\0www.evilsite.com
```

Certificate issuer

Browsers

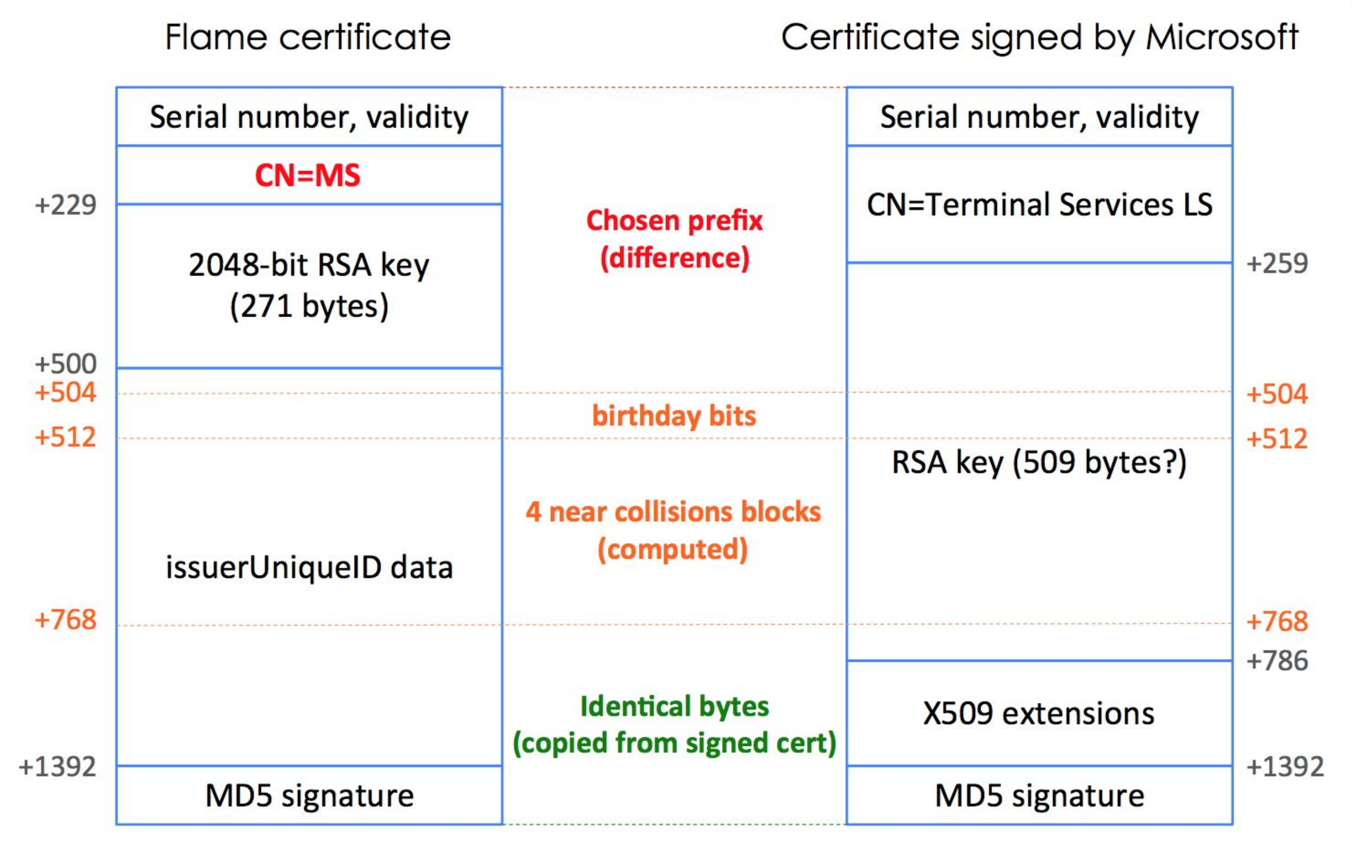
```
www.paypal.com\0
```

```
\0www.evilsite.com
```

TLS attack: Exploiting hash collisions

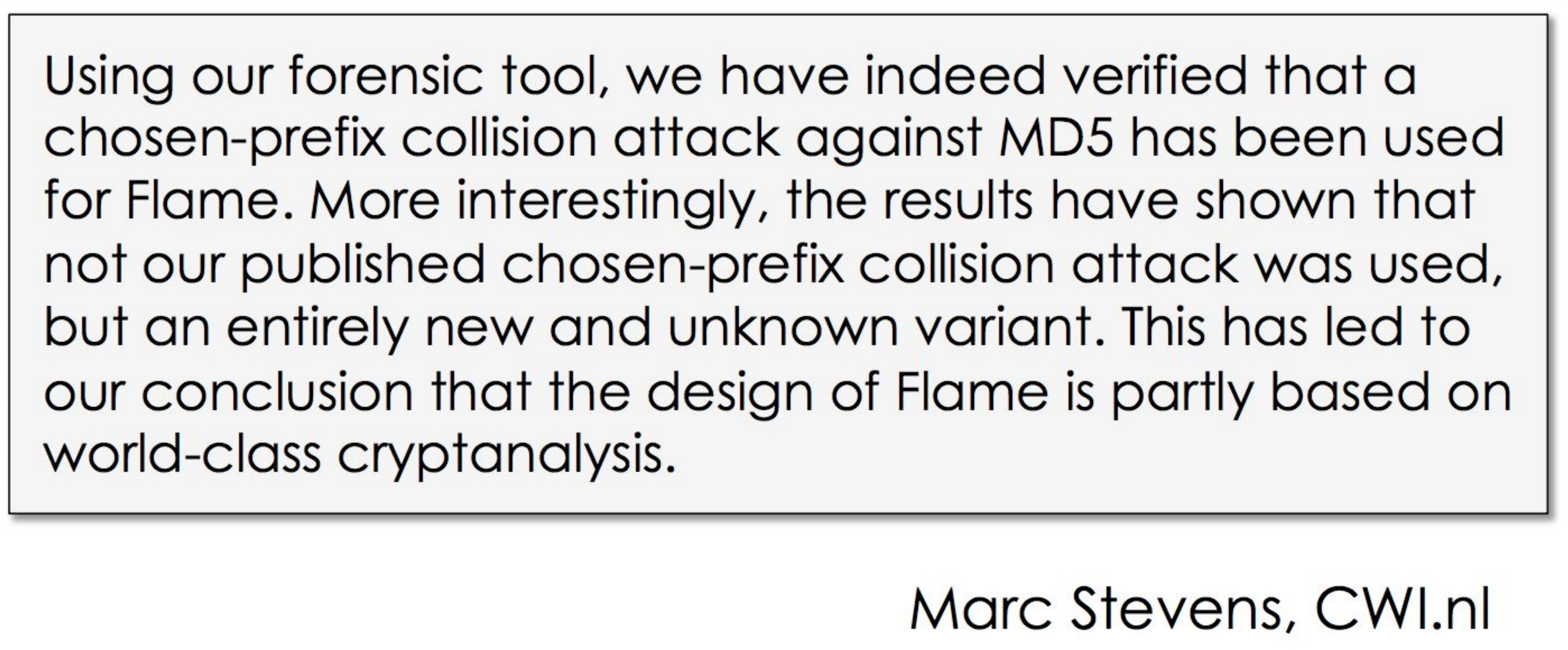
- The Flame targeted attack malware
- Used in cyber espionage ca. 2010 to 2012
- Signed with a fraudulent *intermediate* CA certificate that appears to be issued by Microsoft
  - The fraudulent certificate is accepted by Windows Update
  - Fake intermediate CA certificate was created using an MD5 chosen-prefix collision against an obscure Microsoft Terminal Server Licensing Service certificate that was enabled for code signing and still used MD5
- Considered by many as an example of world-class applied cryptanalysis

TLS attack: Exploiting hash collisions



Source: https://trailofbits.files.wordpress.com/2012/06/flame-md5.pdf

TLS attack: Exploiting hash collisions



Using our forensic tool, we have indeed verified that a chosen-prefix collision attack against MD5 has been used for Flame. More interestingly, the results have shown that not our published chosen-prefix collision attack was used, but an entirely new and unknown variant. This has led to our conclusion that the design of Flame is partly based on world-class cryptanalysis.

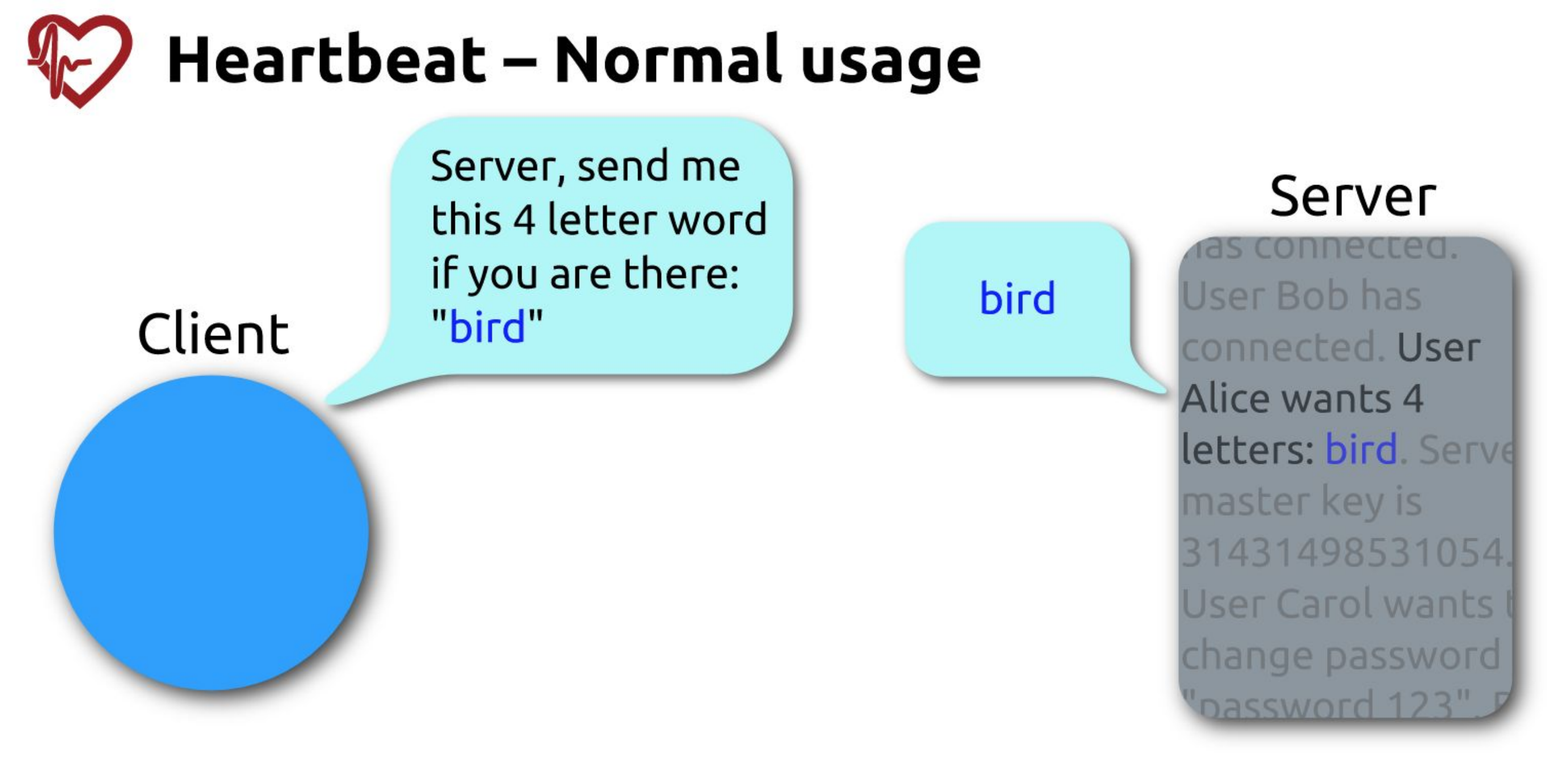
Marc Stevens, CWI.nl

TLS implementation issues

- Implementing cryptography is hard
- Heartbleed bug in OpenSSL, 2014
- The Heartbeat mechanism allows to check whether the connection is still alive
- From the RFC:

  ```
  When a HeartbeatRequest message is received [...], the
  receiver MUST send a corresponding HeartbeatResponse message
  carrying an exact copy of the payload of the received
  HeartbeatRequest.
  ```
- The bug consists in not checking the actual length versus the announced length
- Buffer over-read vulnerability in the *implementation*, not in the standard

TLS implementation issues: HEARTBLEED



Heartbeat – Normal usage

Client

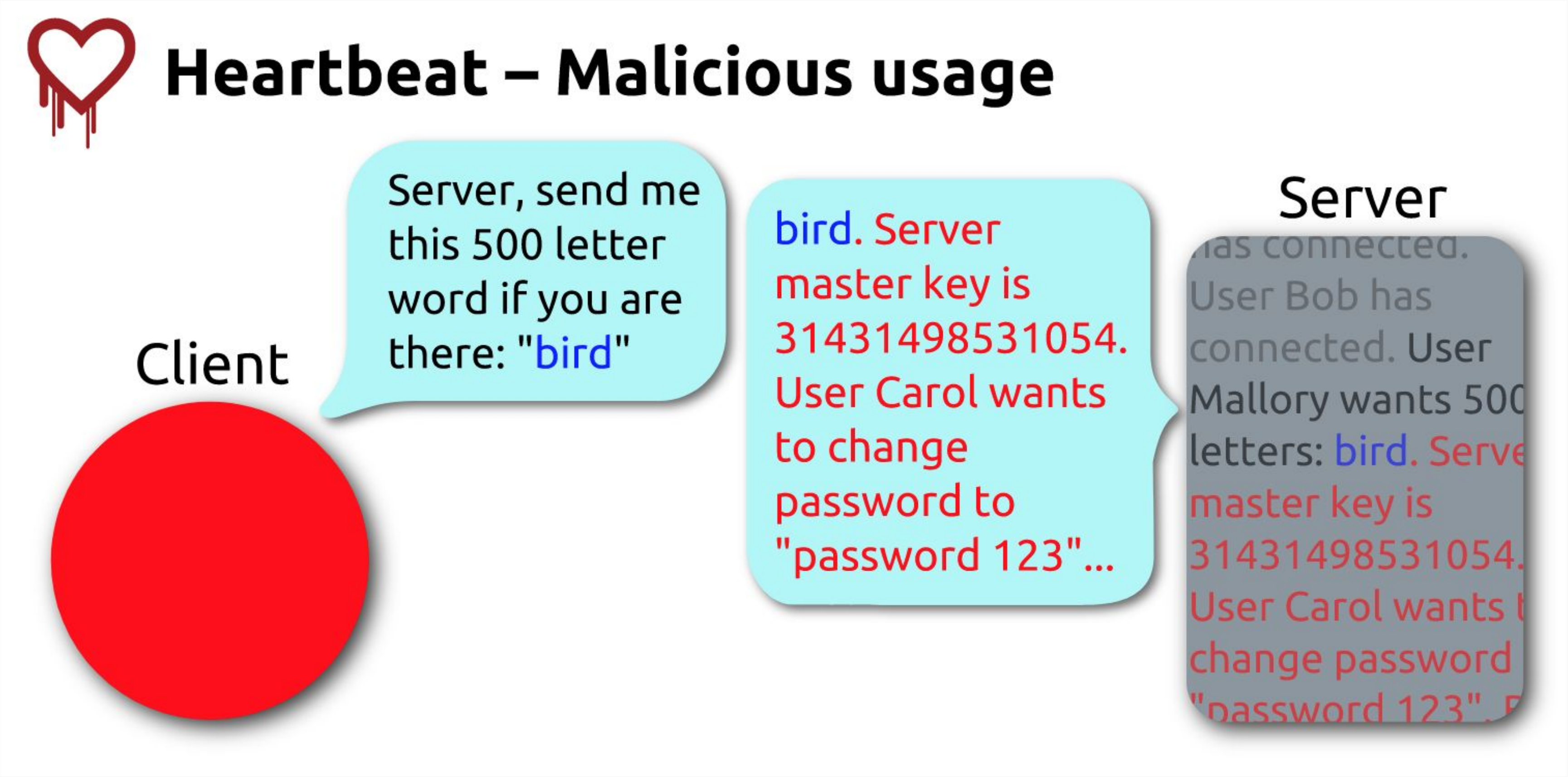
Server, send me this 4 letter word if you are there: "bird"

bird

Server

as connected. User Bob has connected. User Alice wants 4 letters: bird. Serve master key is 31431498531054. User Carol wants t change password "password 123". P

TLS implementation issues: HEARTBLEED



Heartbeat – Malicious usage

Client

Server, send me this 500 letter word if you are there: "bird"

bird. Server master key is 31431498531054. User Carol wants to change password to "password 123"...

Server

has connected. User Bob has connected. User Mallory wants 500 letters: bird. Serve master key is 31431498531054. User Carol wants t change password "password 123".

References

- Prof. Dr. Christian Rossow, Cryptography lecture, CISPA/Uni Saarbrücken
- RFC 5246
- RFC 6176
- Ruben Niederhagen, Applied Crypto lecture, TU Eindhoven
- https://www.wired.com/2011/09/diginotar-bankruptcy/
- Angriffe auf Zertifizierungsdiensteanbieter und Auswirkungen, https://www.isb.admin.ch/dam/isb/de/dokumente/themen/sicherheit/PKI_4-5-2012.pdf.download.pdf/PKI_4-5-2012.pdf

Thank you. Questions?