TLS
SECURITY ASPECTS

RTR WORKSHOP 05.11.2015

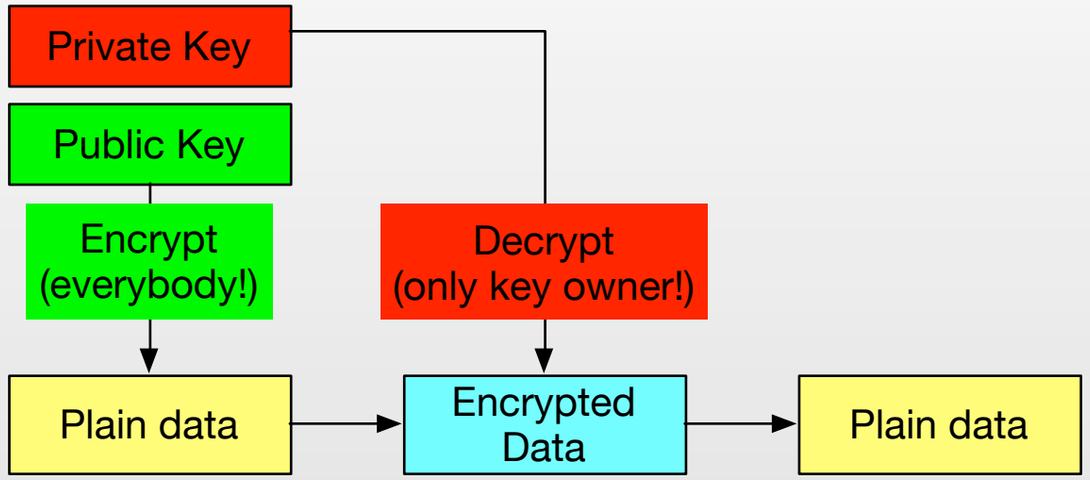
A-SIT PLUS GMBH
DR. PETER TEUFL

CRYPTO CRASH COURSE

Asymmetric cryptography: encryption, decryption

This key is really **private**, only the owner should have it!

This key is really **public**, everyone can and should have it!

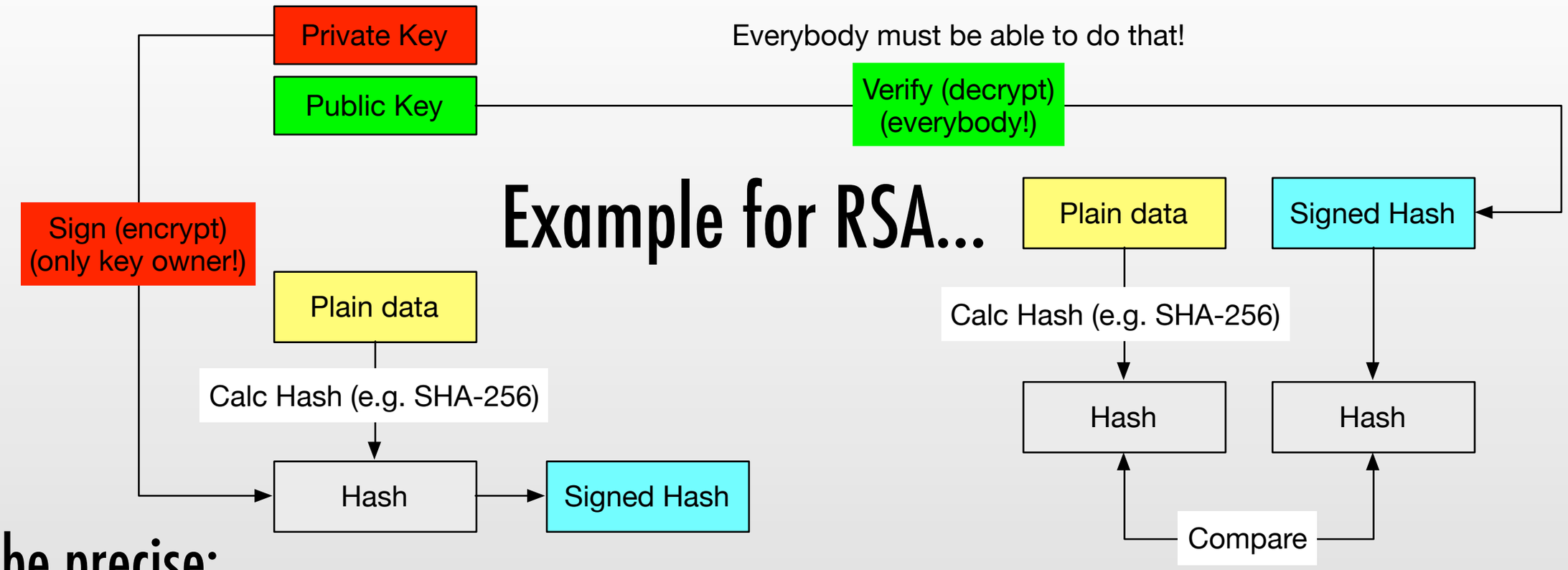


Example for RSA...

- To be precise
  - Data is typically not encrypted/decrypted with the asymmetric keys
  - Symmetric keys are used for that
  - Only the symmetric keys are encrypted/decrypted with the asymmetric ones (due to performance, security (block mode))

CRYPTO CRASH COURSE

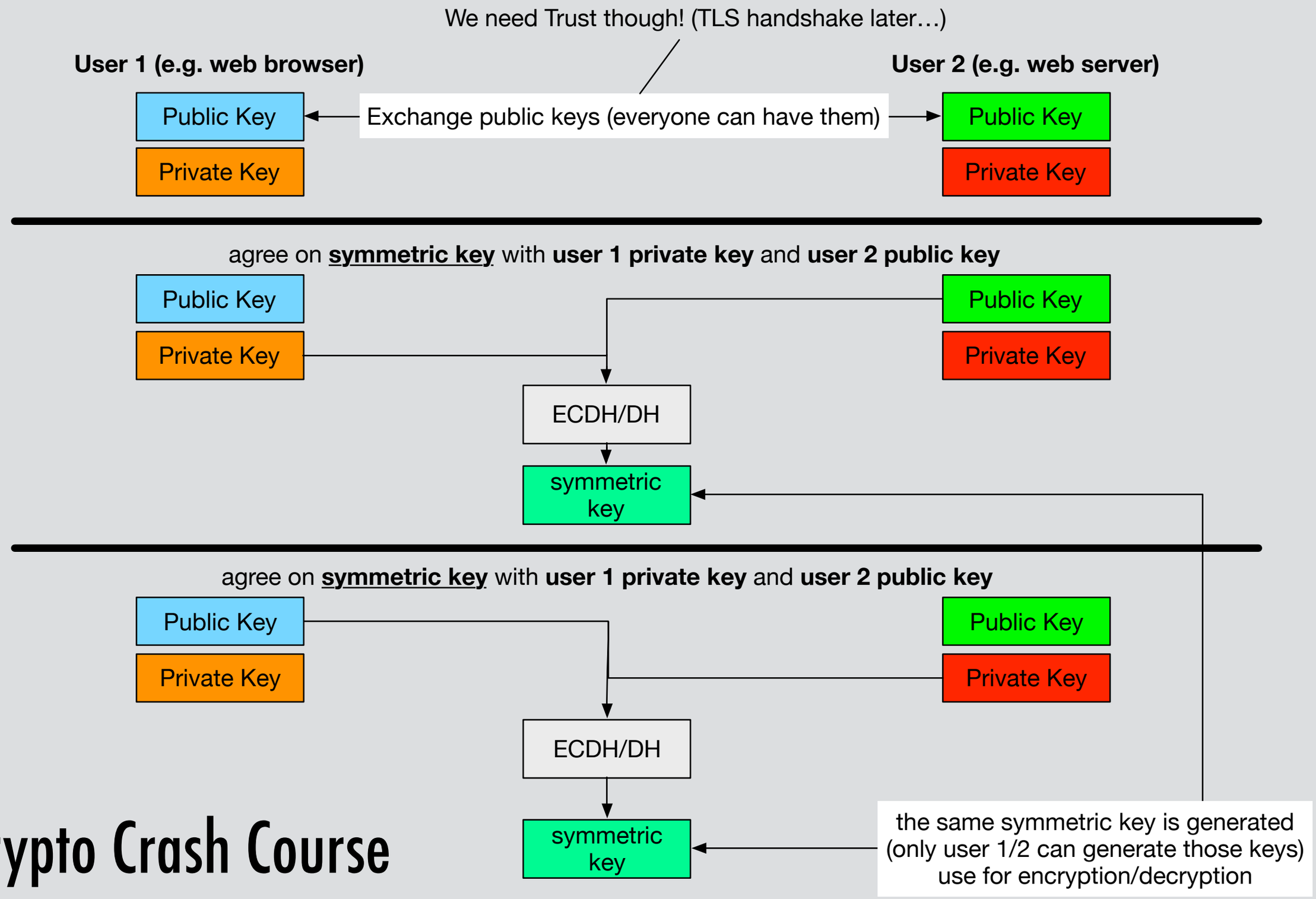
Asymmetric cryptography: signing, verification



To be precise:

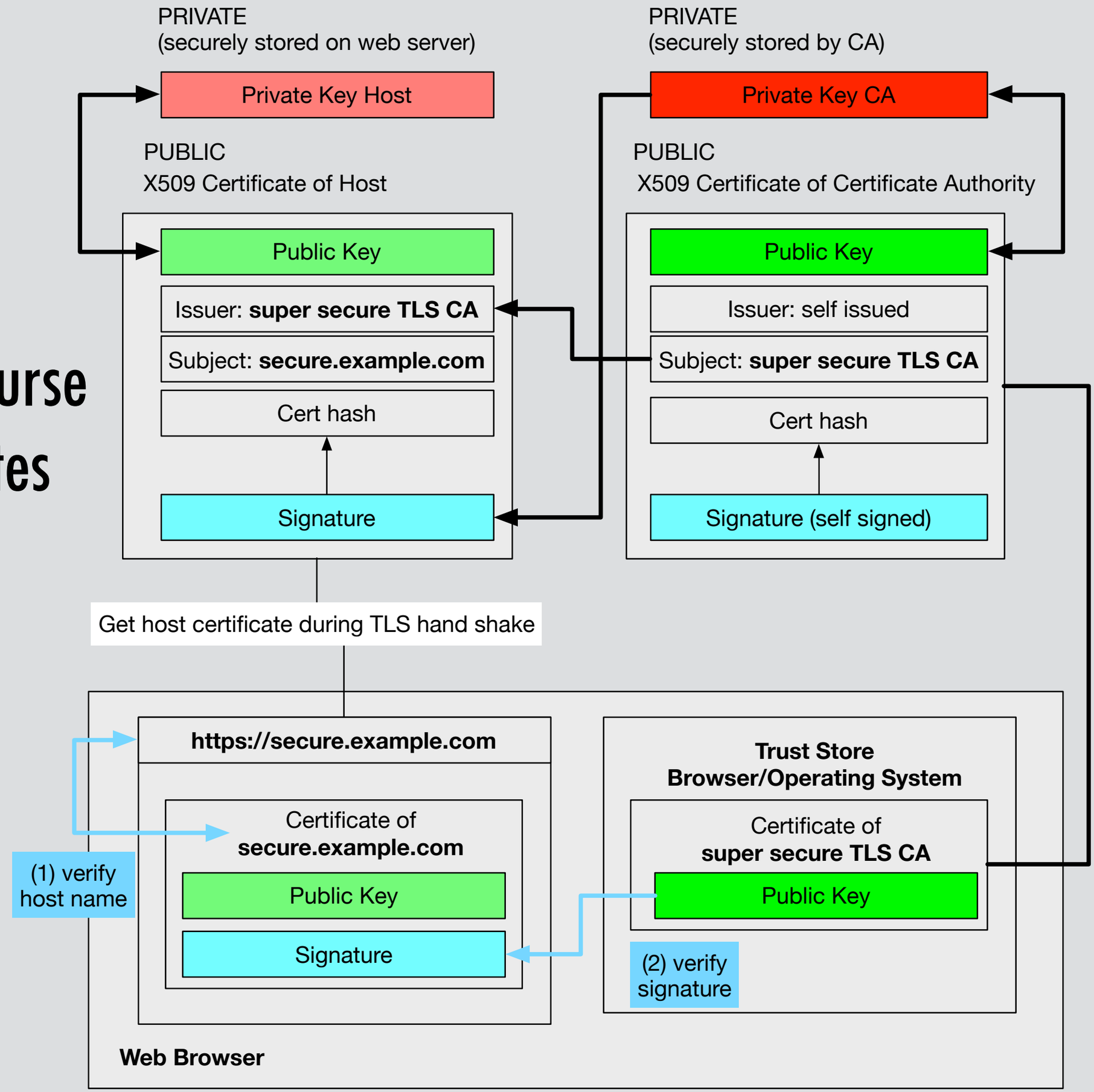
- Not the complete data is signed verified (same issues as with encryption)
- BUT, a short hash (e.g. 256 bit) is calculated and that is signed/verified
- Compare with thoughts on encryption in previous slide

Asymmetric cryptography: key agreement (for en/decryption)



Crypto Crash Course

Crypto Crash Course
X509 Certificates



TOPICS

- Crypto Crash Course
- TLS details
  - Handshake and how to achieve confidentiality, integrity, authenticity
  - Client TLS
  - Cipher suites, Perfect Forward Secrecy
  - HSTS, Certificate Pinning
- Attacks
  - Trust
  - Heartbleed
  - SSLStrip
  - Flame

TRANSPORT LAYER SECURITY (TLS)

- Perfect overview on WikiPedia (history, browser support etc.)
  - http://en.wikipedia.org/wiki/Transport_Layer_Security
  - ECC cipher suites: http://tools.ietf.org/html/rfc4492 (ECDH, ECDHE example)
  - TLS 1.2: http://www.ietf.org/rfc/rfc5246.txt (RSA example)
- Key protocol for secure communication
  - HTTPS, VPNs, for any secure communication
- Initial development by Netscape in the 90s.
  - First public release SSL 2.0 in 1995 (critical sec flaws!)

TRANSPORT LAYER SECURITY (TLS)

- SSL 3.0 in 1996, RFC 6101
- TLS 1.0 in 1999, RFC 2246
  - No significant changes when compared to SSL 3.0
  - downgrade option to SSL 3.0
- TLS 1.1 in 2006, RFC 4346
  - Security fixes
- TLS 1.2 in 2008, RFC 5246, old cipher suites removed, bugfixes
- TLS 1.3, Draft, April 2015 (drop insecure/problematic features)

TLS - PROTOCOL - BASIC STEPS

```
Client                                                Server

ClientHello                  -------->
                                                  ServerHello
                                                  Certificate*
                                            ServerKeyExchange*
                                           CertificateRequest*
                             <--------        ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished                     -------->
                                           [ChangeCipherSpec]
                             <--------             Finished
Application Data             <------->     Application Data

       Figure 1.  Message flow for a full handshake
```

Certificate of server. Standard HTTPS scenario

Certain cipher suites

Certificate required from client

Certificate from client

Client TLS/SSL

source: http://tools.ietf.org/html/rfc5246

CLIENTHELLO (CLIENT)

- TLS version
- Random number
- List of suggested cipher suites
- Compression methods
- Session ID if resumed

```
▷ Frame 698: 196 bytes on wire (1568 bits), 196 bytes captured (1568 bits)
▷ Point-to-Point Protocol
▷ Internet Protocol Version 4, Src: 129.27.152.205 (129.27.152.205), Dst: 173.19
▷ Transmission Control Protocol, Src Port: 54629 (54629), Dst Port: https (443),
▽ Secure Sockets Layer
  ▽ TLSv1 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 135
    ▽ Handshake Protocol: Client Hello
      Handshake Type: Client Hello (1)
      Length: 131
      Version: TLS 1.0 (0x0301)
    ▽ Random
        gmt_unix_time: Jun  1, 2013 09:09:32.000000000 CEST
        random_bytes: 28d2dd3144b736c1c4f3a8bbce57361ef7c5c8a5c2f463cc...
      Session ID Length: 0
      Cipher Suites Length: 50
    ▷ Cipher Suites (25 suites)
      Compression Methods Length: 1
    ▽ Compression Methods (1 method)
        Compression Method: null (0)
      Extensions Length: 40
    ▷ Extension: server_name
    ▷ Extension: elliptic_curves
    ▷ Extension: ec_point_formats
```

CLIENTHELLO - CIPHERSUITES

CipherSuites

```
Cipher Suites Length: 50
▽ Cipher Suites (25 suites)
    Cipher Suite: TLS_EMPTY_RENEGOTIATION_INFO_SCSV (0x00ff)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_RC4_128_SHA (0xc007)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA (0xc008)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
    Cipher Suite: TLS_ECDHE_RSA_WITH_RC4_128_SHA (0xc011)
    Cipher Suite: TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA (0xc012)
    Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA (0xc004)
    Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA (0xc005)
    Cipher Suite: TLS_ECDH_ECDSA_WITH_RC4_128_SHA (0xc002)
    Cipher Suite: TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA (0xc003)
    Cipher Suite: TLS_ECDH_RSA_WITH_AES_128_CBC_SHA (0xc00e)
    Cipher Suite: TLS_ECDH_RSA_WITH_AES_256_CBC_SHA (0xc00f)
    Cipher Suite: TLS_ECDH_RSA_WITH_RC4_128_SHA (0xc00c)
    Cipher Suite: TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA (0xc00d)
    Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
    Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)
    Cipher Suite: TLS_RSA_WITH_RC4_128_MD5 (0x0004)
    Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
    Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x0033)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x0039)
    Cipher Suite: TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA (0x0016)
```

CLIENTHELLO - CIPHERSUITES

- Structure
  - [SSL | TLS], [key exchange], [authentication], [bulk cipher], [message auth]
- Examples
  - TLS_RSA_WITH_AES_128_CBC_SHA (TLS, RSA, RSA, AES 128 CBC, SHA)
  - TLS_ECDHE_RSA_WITH_RC4_128_SHA (TLS, ECDHE, RSA, RC4 128, SHA)
- Many possible cryptographic protocols for key exchange, encryption, authentication, message integrity

SERVERHELLO (SERVER)

```
Client                                               Server

ClientHello                  -------->
                                                  ServerHello
                                                 Certificate*
                                           ServerKeyExchange*
                                          CertificateRequest*
                             <--------       ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished                     -------->
                                           [ChangeCipherSpec]
                             <--------             Finished
Application Data             <------->     Application Data
```

Figure 1. Message flow for a full handshake

SERVERHELLO (SERVER)

- Chosen TLS version
- Random number
- Selected cipher suite
- Selected compression method

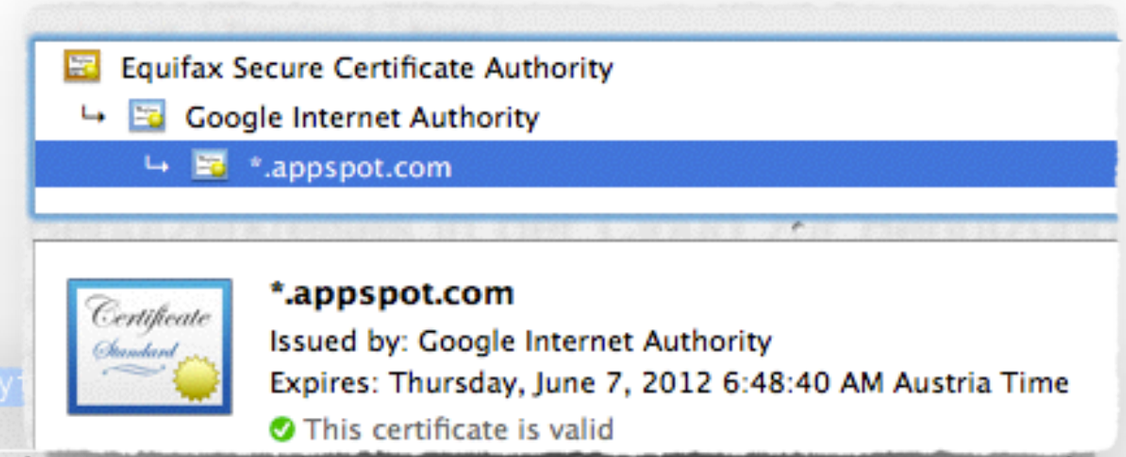
```
▷ Frame 700: 1284 bytes on wire (10272 bits), 1284 bytes captured (10272 bits)
▷ Point-to-Point Protocol
▷ Internet Protocol Version 4, Src: 173.194.35.22 (173.194.35.22), Dst: 129.27.152.205
▷ Transmission Control Protocol, Src Port: https (443), Dst Port: 54629 (54629), Seq: 1
▽ Secure Sockets Layer
  ▽ TLSv1 Record Layer: Handshake Protocol: Server Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 93
    ▽ Handshake Protocol: Server Hello
      Handshake Type: Server Hello (2)
      Length: 89
      Version: TLS 1.0 (0x0301)
    ▽ Random
        gmt_unix_time: Jun  1, 2013 09:09:32.000000000 CEST
        random_bytes: 827c1e0af999e39f4a6719a99fa4c684a232a4c077b85901...
      Session ID Length: 32
      Session ID: 47907ba2869794c3f3645d89f5df6a2fe2f8fff6dc38a288...
      Cipher Suite: TLS_ECDHE_RSA_WITH_RC4_128_SHA (0xc011)
      Compression Method: null (0)
      Extensions Length: 17
    ▷ Extension: server_name
    ▷ Extension: renegotiation_info
    ▷ Extension: ec_point_formats
```

CERTIFICATE (SERVER)

```
Client                                                Server

ClientHello                  -------->
                                                 ServerHello
                                                Certificate*
                                          ServerKeyExchange*
                                         CertificateRequest*
                             <--------       ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished                     -------->
                                          [ChangeCipherSpec]
                             <--------             Finished
Application Data             <------->     Application Data
```

Figure 1. Message flow for a full handshake

CERTIFICATE (SERVER)




```
▷ Frame 701: 751 bytes on wire (6008 bits), 751 byt
▷ Point-to-Point Protocol
▷ Internet Protocol Version 4, Src: 173.194.35.22 (173.194.35.22), Dst: 129.27.152.205 (129.27.152.205)
▷ Transmission Control Protocol, Src Port: https (443), Dst Port: 54629 (54629), Seq: 1436308311, Ack: 1
▷ [2 Reassembled TCP Segments (1825 bytes): #700(1130), #701(695)]
▽ Secure Sockets Layer
  ▽ TLSv1 Record Layer: Handshake Protocol: Certificate
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 1603
    ▽ Handshake Protocol: Certificate
        Handshake Type: Certificate (11)
        Length: 1599
        Certificates Length: 1596
      ▽ Certificates (1596 bytes)
          Certificate Length: 898
        ▷ Certificate (id-at-commonName=www.gmail.com,id-at-organizationName=Google Inc,id-at-localityN
          Certificate Length: 692
        ▷ Certificate (id-at-commonName=Google Internet Authority,id-at-organizationName=Google Inc,id-
  ▷ TLSv1 Record Layer: Handshake Protocol: Server Key Exchange
  ▷ TLSv1 Record Layer: Handshake Protocol: Server Hello Done
```

SERVERKEYEXCHANGE (SERVER)

```
Client                                               Server

ClientHello                  -------->

                                                ServerHello
                                               Certificate*
                                         ServerKeyExchange*
                                        CertificateRequest*
                             <--------      ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished                     -------->
                                          [ChangeCipherSpec]
                             <--------             Finished
Application Data             <------->     Application Data
```

Figure 1. Message flow for a full handshake

SERVERKEYEXCHANGE (SERVER)

```
> Frame 701: 751 bytes on wire (6008 bits), 751 bytes captured (6008 bits)
> Point-to-Point Protocol
> Internet Protocol Version 4, Src: 173.194.35.22 (173.194.35.22), Dst: 129.
> Transmission Control Protocol, Src Port: https (443), Dst Port: 54629 (54
> [2 Reassembled TCP Segments (1825 bytes): #700(1130), #701(695)]
v Secure Sockets Layer
  > TLSv1 Record Layer: Handshake Protocol: Certificate
  v TLSv1 Record Layer: Handshake Protocol: Server Key Exchange
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 203
    v Handshake Protocol: Server Key Exchange
        Handshake Type: Server Key Exchange (12)
        Length: 199
  > TLSv1 Record Layer: Handshake Protocol: Server Hello Done
```

CERTIFICATEREQUEST (SERVER)

```
Client                                               Server

ClientHello                  -------->
                                                ServerHello
                                               Certificate*
                                         ServerKeyExchange*
                                        CertificateRequest*
                             <--------      ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished                     -------->
                                          [ChangeCipherSpec]
                             <--------             Finished
Application Data             <------->     Application Data

       Figure 1.  Message flow for a full handshake
```

Missing... Most HTTPS servers do not require a client certificate

SERVERHELLODONE (SERVER)

```
Client                                               Server

ClientHello                  -------->
                                                ServerHello
                                               Certificate*
                                         ServerKeyExchange*
                                        CertificateRequest*
                             <--------      ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished                     -------->
                                          [ChangeCipherSpec]
                             <--------             Finished
Application Data             <------->     Application Data

       Figure 1.  Message flow for a full handshake
```

- Tells the client that ServerHello and associated messages have been sent

TLS - SERVERHELLODONE

```
▷ Frame 701: 751 bytes on wire (6008 bits), 751 bytes captured (6008 bits)
▷ Point-to-Point Protocol
▷ Internet Protocol Version 4, Src: 173.194.35.22 (173.194.35.22), Dst: 129.2
▷ Transmission Control Protocol, Src Port: https (443), Dst Port: 54629 (5462
▷ [2 Reassembled TCP Segments (1825 bytes): #700(1130), #701(695)]
▽ Secure Sockets Layer
  ▷ TLSv1 Record Layer: Handshake Protocol: Certificate
  ▷ TLSv1 Record Layer: Handshake Protocol: Server Key Exchange
  ▽ TLSv1 Record Layer: Handshake Protocol: Server Hello Done
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 4
    ▽ Handshake Protocol: Server Hello Done
        Handshake Type: Server Hello Done (14)
        Length: 0
```

CERTIFICATE (CLIENT)

Missing...

No certificate was requested by the server

```
Client                                                Server

ClientHello                        -------->
                                                 ServerHello
                                                Certificate*
                                          ServerKeyExchange*
                                         CertificateRequest*
                                   <--------      ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished                           -------->
                                                [ChangeCipherSpec]
                                   <--------             Finished
Application Data                   <------->     Application Data

       Figure 1.  Message flow for a full handshake
```

CLIENTKEYEXCHANGE (CLIENT)

```
Client                                               Server

ClientHello                  -------->
                                                ServerHello
                                               Certificate*
                                         ServerKeyExchange*
                                        CertificateRequest*
                             <--------      ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished                     -------->
                                          [ChangeCipherSpec]
                             <--------             Finished
Application Data             <------->     Application Data
```

Figure 1. Message flow for a full handshake

CLIENTKEYEXCHANGE (CLIENT)

```
▷ Frame 703: 131 bytes on wire (1048 bits), 131 bytes captured (1048 bits)
▷ Point-to-Point Protocol
▷ Internet Protocol Version 4, Src: 129.27.152.205 (129.27.152.205), Dst: 173
▷ Transmission Control Protocol, Src Port: 54629 (54629), Dst Port: https (44
▽ Secure Sockets Layer
  ▽ TLSv1 Record Layer: Handshake Protocol: Client Key Exchange
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 70
    ▽ Handshake Protocol: Client Key Exchange
        Handshake Type: Client Key Exchange (16)
        Length: 66
```

CHANGECIPHERSPEC (CLIENT)

```
Client                                               Server

ClientHello                  -------->
                                                ServerHello
                                               Certificate*
                                         ServerKeyExchange*
                                        CertificateRequest*
                             <--------      ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished                     -------->
                                          [ChangeCipherSpec]
                             <--------             Finished
Application Data             <------->     Application Data
```

Figure 1. Message flow for a full handshake

CHANGECIPHERSPEC (CLIENT)

```
Frame 704: 62 bytes on wire (496 bits), 62 bytes captured (496 bits)
Point-to-Point Protocol
Internet Protocol Version 4, Src: 129.27.152.205 (129.27.152.205), Dst: 17
Transmission Control Protocol, Src Port: 54629 (54629), Dst Port: https (4
Secure Sockets Layer
  TLSv1 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
      Content Type: Change Cipher Spec (20)
      Version: TLS 1.0 (0x0301)
      Length: 1
      Change Cipher Spec Message
```

From client
ChangeCipherSpec:

telling the server
that everything is encrypted from now

FINISHED (CLIENT)

```
Client                                               Server

ClientHello                  -------->
                                                ServerHello
                                               Certificate*
                                         ServerKeyExchange*
                                        CertificateRequest*
                             <--------      ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished                     -------->
                                          [ChangeCipherSpec]
                             <--------             Finished
Application Data             <------->     Application Data
```

Figure 1. Message flow for a full handshake

FINISHED (CLIENT)

- Finished:
  - sending hash, MAC of previous handshake messages (encrypted)
  - server decrypts message, verifies hashes

```
Frame 705: 97 bytes on wire (776 bits), 97 bytes captured (776 bits)
Point-to-Point Protocol
Internet Protocol Version 4, Src: 129.27.152.205 (129.27.152.205), Dst: 173.1
Transmission Control Protocol, Src Port: 54629 (54629), Dst Port: https (443)
Secure Sockets Layer
  TLSv1 Record Layer: Handshake Protocol: Encrypted Handshake Message
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 36
      Handshake Protocol: Encrypted Handshake Message
```

APPLICATION DATA (CLIENT, SERVER)

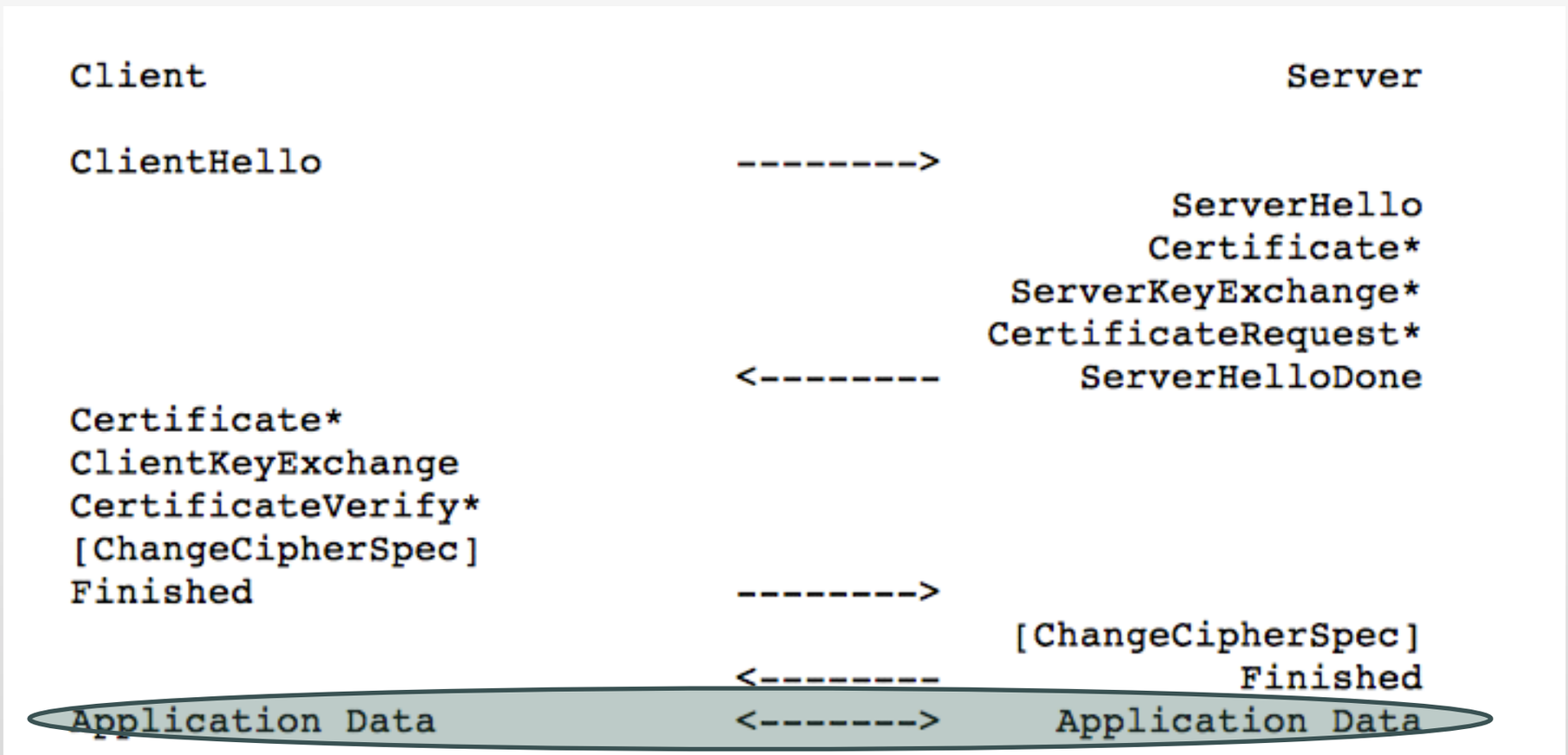
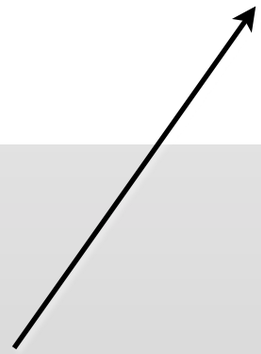


Figure 1. Message flow for a full handshake

TLS - ENCRYPTED APPLICATION DATA

```
> Frame 709: 406 bytes on wire (3248 bits), 406 bytes captured (3248 bits)
> Point-to-Point Protocol
> Internet Protocol Version 4, Src: 129.27.152.205 (129.27.152.205), Dst: 173.194.35.22 (173.1
> Transmission Control Protocol, Src Port: 54629 (54629), Dst Port: https (443), Seq: 1322976
v Secure Sockets Layer
  v TLSv1 Record Layer: Application Data Protocol: http
    Content Type: Application Data (23)
    Version: TLS 1.0 (0x0301)
    Length: 345
    Encrypted Application Data: 179abd705c3811b151ebf4a9c8771d42012a3381edfe878c...
```



Encrypted HTTP traffic

TOPICS

- Crypto Crash Course
- TLS details
  - Handshake and how to achieve confidentiality, integrity, authenticity
  - Client TLS
  - Cipher suites, Perfect Forward Secrecy
  - HSTS, Certificate Pinning
- Attacks
  - Trust
  - Heartbleed
  - SSLStrip
  - Flame

TOPICS

- Crypto Crash Course
- TLS details
  - Handshake and how to achieve confidentiality, integrity, authenticity
  - Client TLS
  - Cipher suites, Perfect Forward Secrecy
  - HSTS, Certificate Pinning
- Attacks
  - Trust
  - Heartbleed
  - SSLStrip
  - Flame

CIPHER SUITES

- key exchange, agreement: RSA, DH, DHE, ECDH, ECDHE
  - How to exchange the bulk encryption key? (req for confidentiality, integrity)
- authentication: RSA, DSS, ECDSA
  - How to verify whether the server is authentic? (authenticity)
- bulk ciphers: AES, 3DES, RC4
  - How to encrypt data? (confidentiality)
- message authentication: SHA{256, 384}, MD5 (!)
  - How to verify the integrity of the transmitted data? (integrity)
- perfect forward secrecy:
  - Depends on deployed key exchange/agreement algorithm

CIPHER SUITES

- Structure
  - [SSL | TLS], [key exchange], [authentication], [bulk cipher], [message auth]
- Examples
  - TLS_RSA_WITH_AES_128_CBC_SHA (TLS, RSA, RSA, AES 128 CBC, SHA)
  - TLS_ECDHE_RSA_WITH_RC4_128_SHA (TLS, ECDHE, RSA, RC4 128, SHA)
- Many possible cryptographic protocols for key exchange, encryption, authentication, message integrity

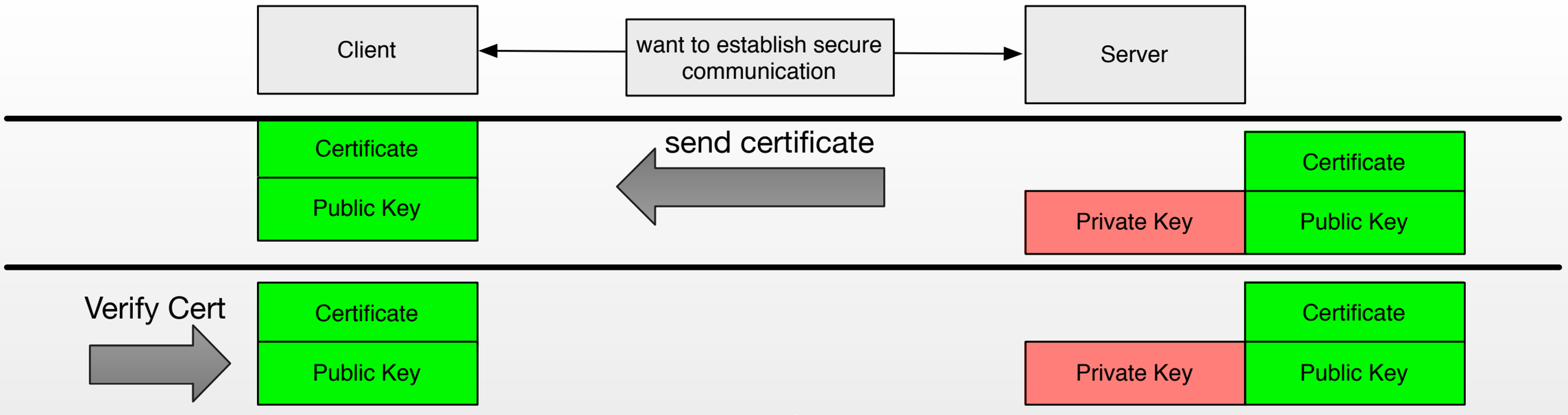
CIPHER SUITES

```
Cipher Suites Length: 50
▽ Cipher Suites (25 suites)
    Cipher Suite: TLS_EMPTY_RENEGOTIATION_INFO_SCSV (0x00ff)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_RC4_128_SHA (0xc007)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA (0xc008)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
    Cipher Suite: TLS_ECDHE_RSA_WITH_RC4_128_SHA (0xc011)
    Cipher Suite: TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA (0xc012)
    Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA (0xc004)
    Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA (0xc005)
    Cipher Suite: TLS_ECDH_ECDSA_WITH_RC4_128_SHA (0xc002)
    Cipher Suite: TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA (0xc003)
    Cipher Suite: TLS_ECDH_RSA_WITH_AES_128_CBC_SHA (0xc00e)
    Cipher Suite: TLS_ECDH_RSA_WITH_AES_256_CBC_SHA (0xc00f)
    Cipher Suite: TLS_ECDH_RSA_WITH_RC4_128_SHA (0xc00c)
    Cipher Suite: TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA (0xc00d)
    Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
    Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)
    Cipher Suite: TLS_RSA_WITH_RC4_128_MD5 (0x0004)
    Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
    Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x0033)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x0039)
    Cipher Suite: TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA (0x0016)
```

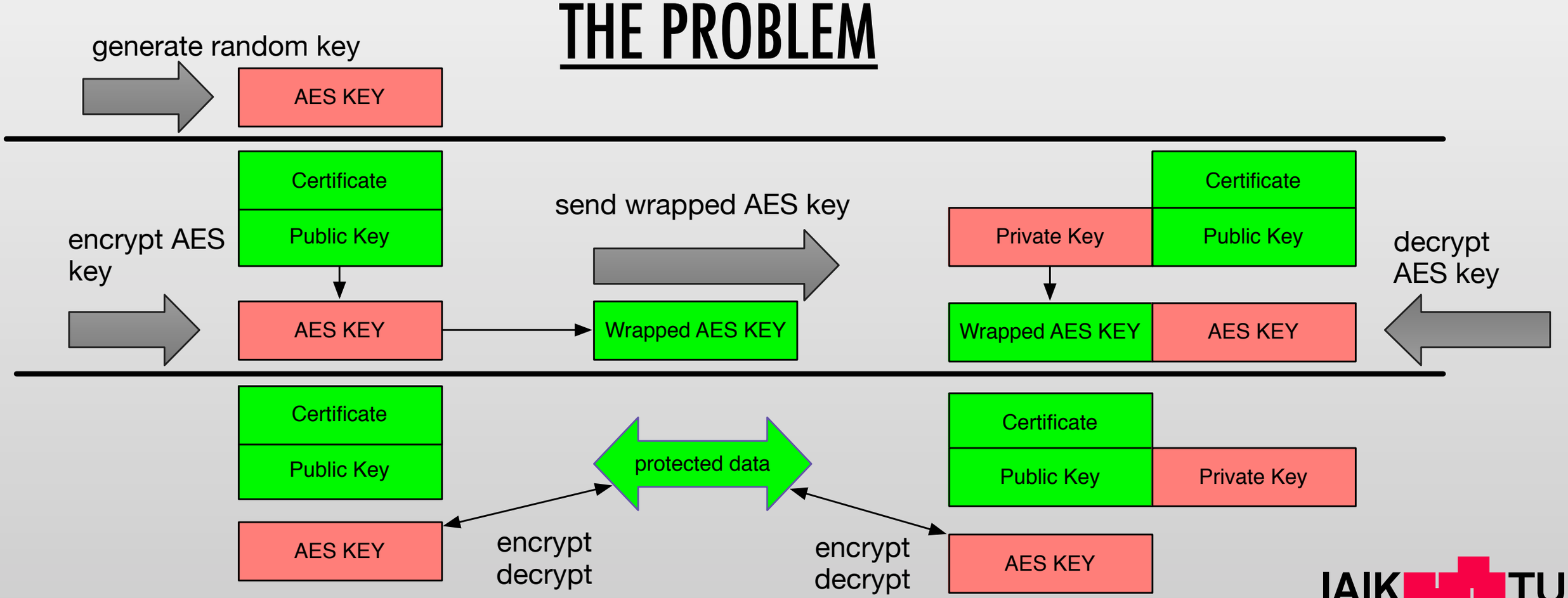
PERFECT FORWARD SECRECY

- Perfect Forward Secrecy (PFS)
  - Popped up again due to NSA topic...
  - Even if an attacker gains access to the long term keys
    - e.g. RSA keys used for the X509 certificates
  - the session keys used for bulk encryption cannot be derived
  - http://crypto.stackexchange.com/questions/8933/how-can-i-use-ssl-tls-with-perfect-forward-secrecy

NO PFS EXAMPLE (SIMPLIFIED TLS HANDSHAKE)

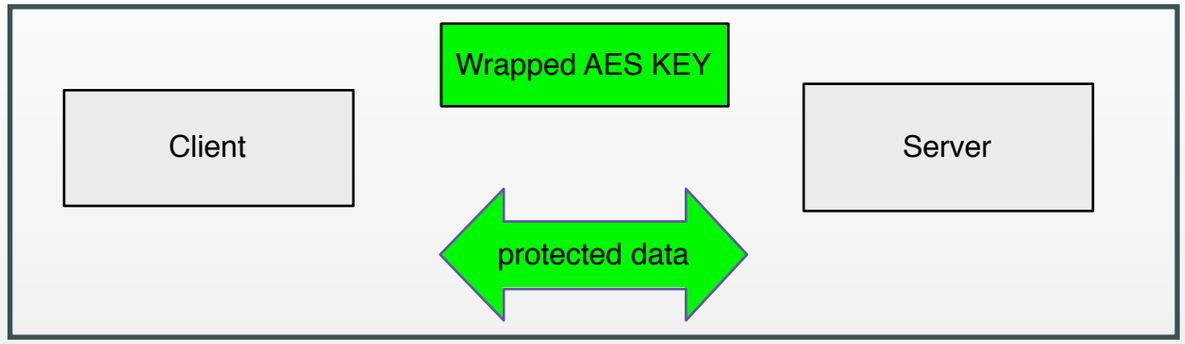


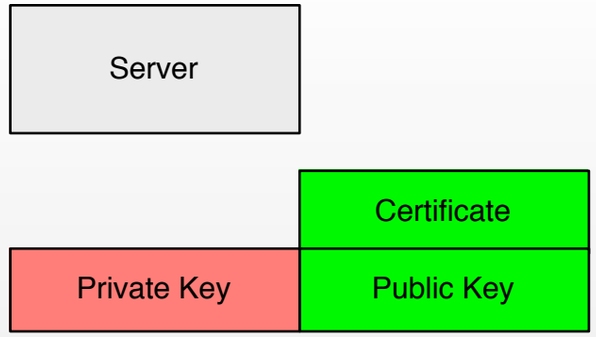
THE PROBLEM



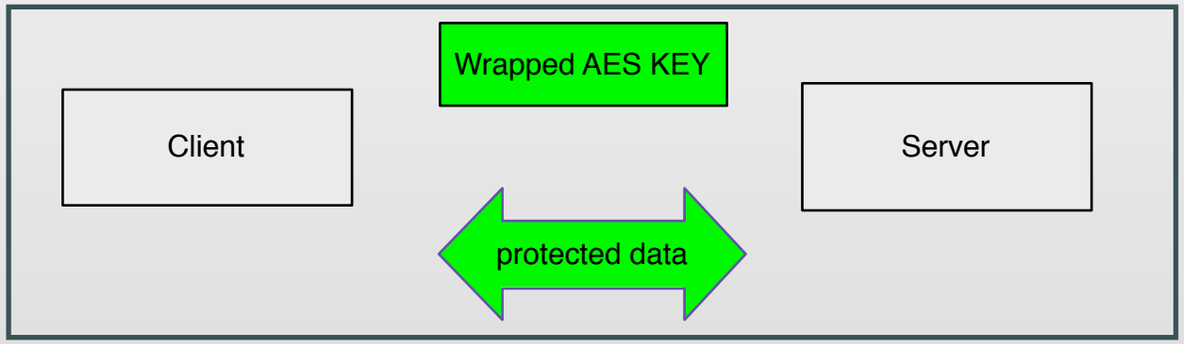
IAIK TU Graz

NO PFS EXAMPLE

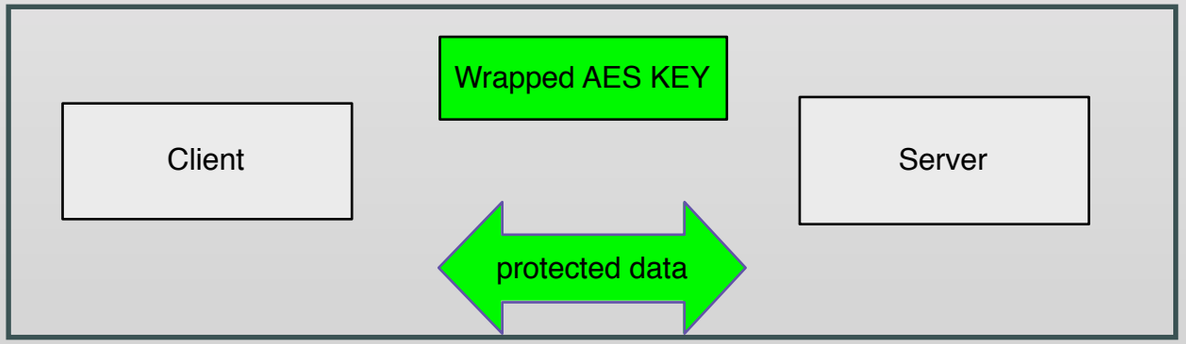




12.03.2012
capture and store data



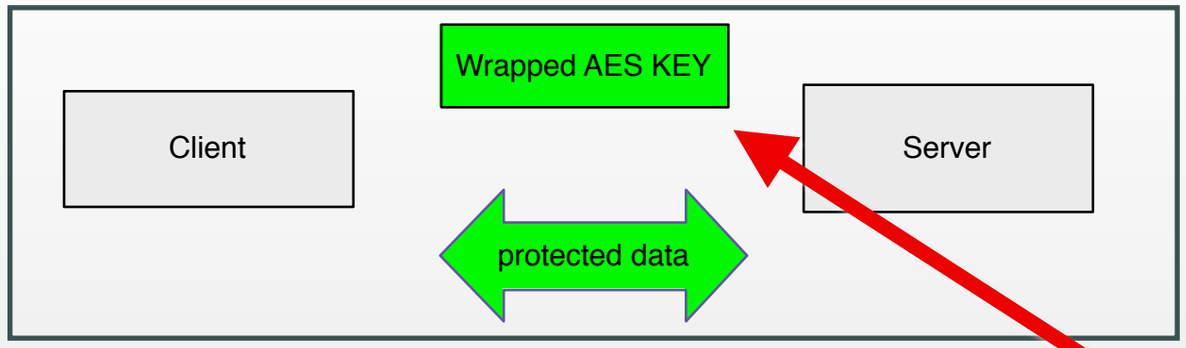
10.10.2013
capture and store data



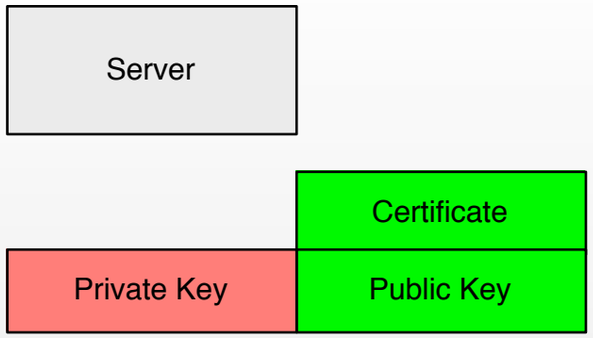
01.01.2014
capture and store data

IAIK TU Graz

NO PFS EXAMPLE



12.03.2012
capture and store data



10.04.2015
gain access to private key

decrypt keys and data

10.10.2013
capture and store data

01.01.2014
capture and store data

IAIK TU Graz

NO PFS EXAMPLE

- TLS_RSA_WITH_AES_128_CBC_SHA
- which actually means: TLS_RSA_RSA_WITH_AES_128_CBC_SHA
  - key exchange via RSA (gaining a bulk encryption key, here AES)
  - authentication via RSA (server proofs its authenticity)
  - bulk cipher: AES_128_CBC
  - message authentication: SHA based MAC

PFS EXAMPLE

- ECDHE_RSA_WITH_AES_128_GCM_SHA
- ECDH_RSA_WITH_AES_128_GCM_SHA
- which actually means
  - Key exchange: ECDHE - **E**lliptic **C**urve **D**iffie **H**ellman Key Exchange, **E**phemeral
  - Authentication: RSA
  - Bulk cipher: AES_128_GCM
  - Message authentication: SHA based MAC
- RSA and elliptic curves?

PFS EXAMPLE

ECDHE: Elliptic Curve Diffie Hellman - Ephemeral (PFS)

ephemeral | ɪˈfɛm(ə)r(ə)l, -ˈfiːm- |
adjective
lasting for a very short time. *fashions are ephemeral: new ones regularly drive out the old. works of more than ephemeral interest.*
• (chiefly of plants) having a very short life cycle. *chickweed is an ephemeral weed, producing several generations in one season.*

noun
an ephemeral plant. *ephemerals avoid the periods of drought as seeds.*

DERIVATIVES
ephemerality | -ˈralɪti | noun ,
ephemerally adverb

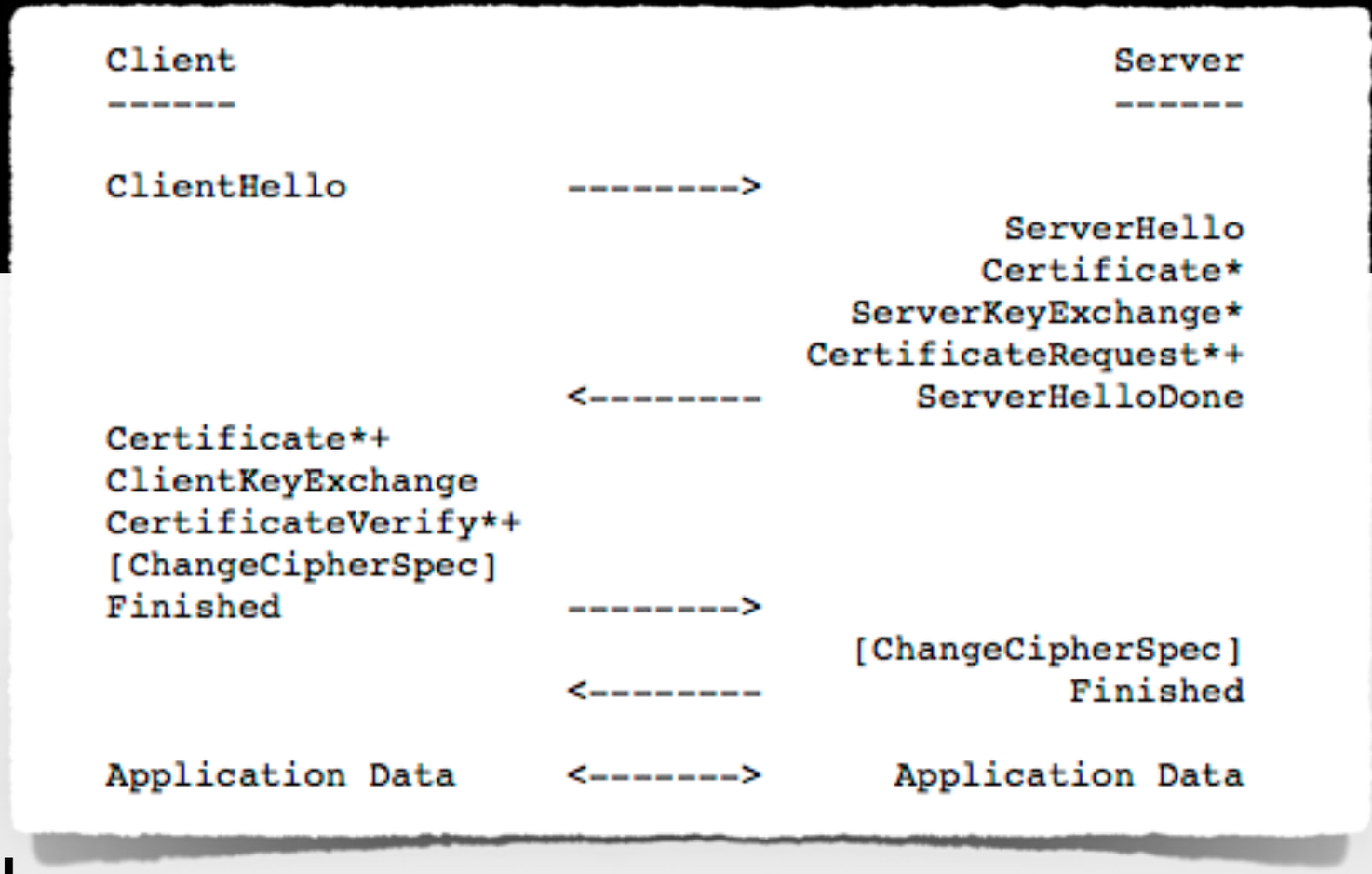
ORIGIN late 16th cent.: from Greek ***ephēmeros*** (see EPHEMERA) + -AL.

PFS EXAMPLE

- ClientHello, ServerHello messages may include curve parameters for ECC use
- Certificate: sends certificate to client

```
Client                                               Server
------                                               ------

ClientHello                --------->
                                                  ServerHello
                                                 Certificate*
                                           ServerKeyExchange*
                                          CertificateRequest*+
                           <---------         ServerHelloDone
Certificate*+
ClientKeyExchange
CertificateVerify*+
[ChangeCipherSpec]
Finished                   --------->
                                          [ChangeCipherSpec]
                           <---------                Finished

Application Data           <--------->     Application Data
```

PFS EXAMPLE



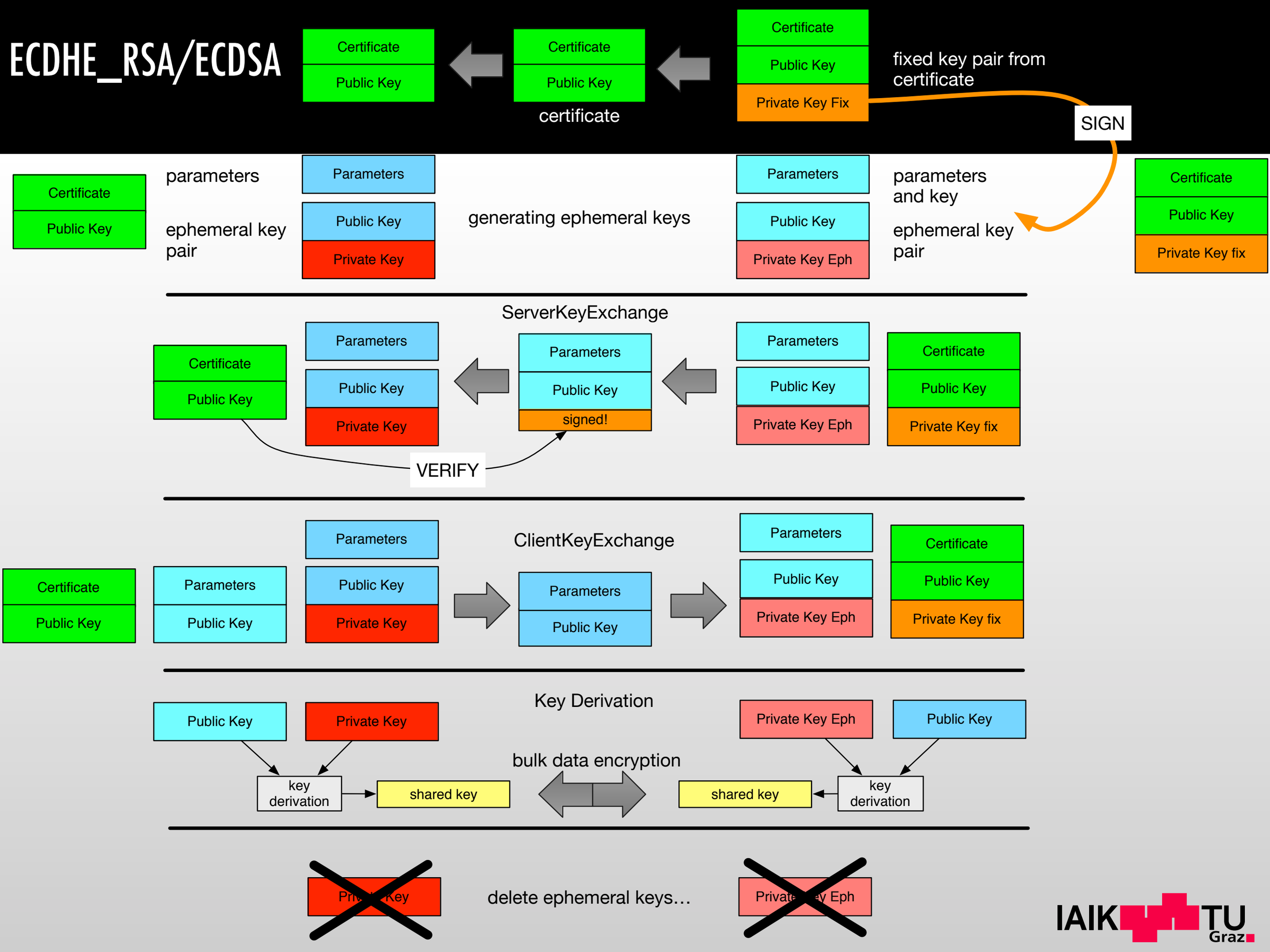
- ECDHE
  Elliptic Curve Diffie Hellman - Ephemeral
  - ServerKeyExchange needed!
  - Server generates "ephemeral key pair"
  - Sends signed public key and parameters to the client
  - Signed with private key of server certificate
    - Could be RSA or ECDSA etc. depending on the certificate

PFS EXAMPLE

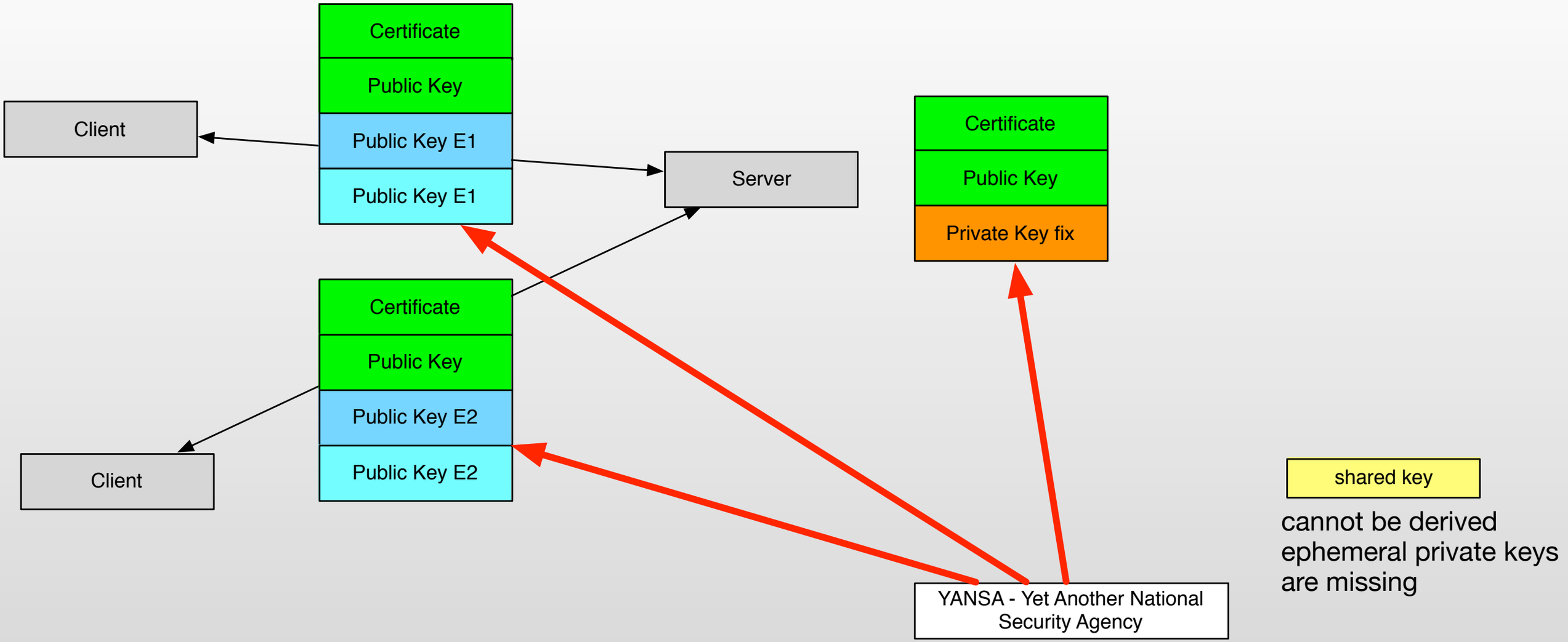
```
Client                                              Server
------                                              ------

ClientHello                --------->
                                                 ServerHello
                                                Certificate*
                                           ServerKeyExchange*
                                          CertificateRequest*+
                           <---------         ServerHelloDone
Certificate*+
ClientKeyExchange
CertificateVerify*+
[ChangeCipherSpec]
Finished                   --------->
                                          [ChangeCipherSpec]
                           <---------             Finished

Application Data           <--------->     Application Data
```

- ClientKeyExchange!
- For ECDHE and ECHD, client always generates ephemeral key pair
- Parameters, keys are sent to server via ClientKeyExchange Message
- Client, Server can calculate common secret via ECDH

ECDHE_RSA/ECDSA



PFS EXAMPLE


Certificate
Public Key
Client
Public Key E1
Public Key E1
Server
Certificate
Public Key
Private Key fix
Certificate
Public Key
Public Key E2
Public Key E2
Client
YANSA - Yet Another National Security Agency
shared key
cannot be derived ephemeral private keys are missing

TOPICS

- Crypto Crash Course
- TLS details
  - Handshake and how to achieve confidentiality, integrity, authenticity
  - Client TLS
  - Cipher suites, Perfect Forward Secrecy
  - HSTS, Certificate Pinning
- Attacks
  - Trust
  - Heartbleed
  - SSLStrip
  - Flame

HSTS - HTTP STRICT TRANSPORT SECURITY

- Attacks are often based on HTTPS to HTTP downgrades
- Web page offers HTTPS/HTTP,
  attacker injects HTTP links to force user to user weak HTTP communication
- Web page offers HTTPS only
  attacker uses a proxy (SSLSSTRIP) to move user to HTTP communications.
- How to deal with that?

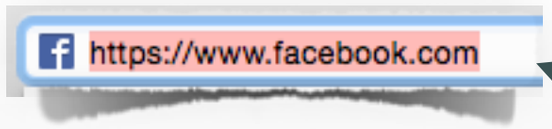
HSTS - HTTP STRICT TRANSPORT SECURITY

- Tell the browser that all connections to a domain/host are HTTPS only
- From now on the browser does not accept HTTP communication to that site
- How?
  - via an HTTP header

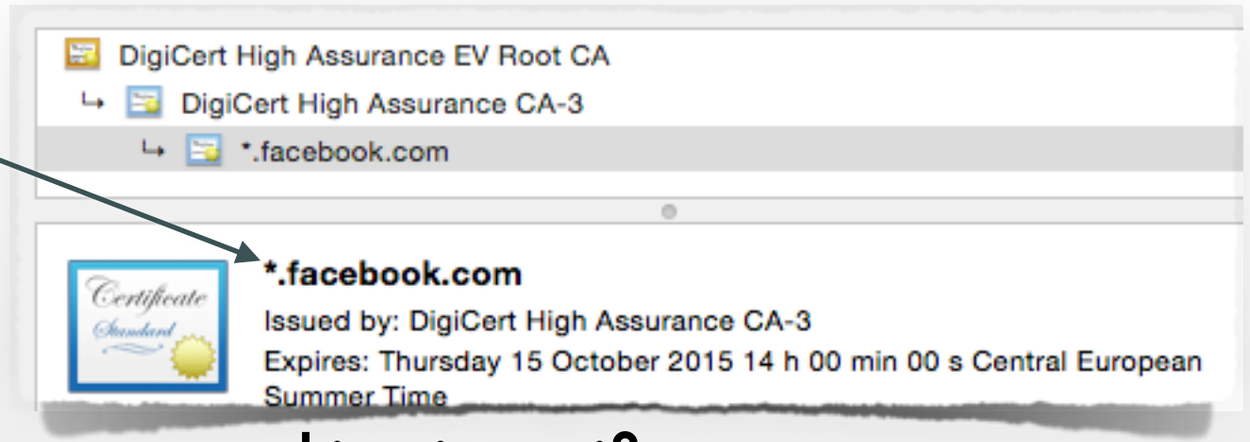
```
Strict-Transport-Security: max-age=31536000; includeSubDomains;
```

  - header can only be set during a valid HTTPS request, headers in HTTP only communication are ignored
  - http://tools.ietf.org/html/rfc6797
  - that's it? is everything secure now?

CERTIFICATE PINNING



dns entry



subject in certificate

- PKI trust
  - many CAs in trust store
    - TLS trust based on: (referring to crypto crash course)
      - (1) certificate issued by a trusted CA
      - (2) compare DNS host name with host name in CN of certificate
    - what happens if:

CERTIFICATE PINNING

- Introduce certificate pinning (http://www.rfc-editor.org/rfc/rfc7469.txt)
  - remember hash values (pins) of public keys associated with X509 certificates of TLS servers
  - if PIN changes (meaning that the certificate changes), drop connection even if certificate would be trustworthy and DNS name matches with subject CN
  - certificate pins are either stored in browser (or app) or submitted (like HSTS) via HTTP headers during the first connection (same issues as with HSTS, first connection must be secure)

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

CERTIFICATE PINNING

- Getting necessary
  - to avoid MITM attacks
  - to deal with the problem of many trusted CAs in the browser that have different quality levels
- Many more details for different operating systems
  - https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning