Was ist neu bei TLS 1.3?

TSLv1.3 – 21nd Century Internet Transmission Security

Dr. Erwin Hoffmann

November, 20th, 2018

Todays Agenda

- History of TLS and it's cryptographic concepts
- Working model of former TLS implementations
- Changes done for TLS 1.3
- OpenSSL 1.1.1 under OmniOSce
- Installation of fehQlibs + ucspi-ssl
- Protocol analysis using WireShark and 'testssl'

TLS 1.3. has been published in RFC 8446 after years of discussion in the IETF in August 2018. Since September 2018 OpenSSL 1.1.1 is available supporting TLSv1.3. My software ucspi-ssl-0.10 is OpenSSL 1.1.1 enabled and used here together with OmniOS (r15028).

Internet security protocols

Transport Layer Security (TLS) – aka Secure Socket Layer (SSL) – is a fundamental IT security concept and in particular used for Web encryption (HTTPS) and encrypted email transmission (ESMTPS/ESMTP+StartTLS).

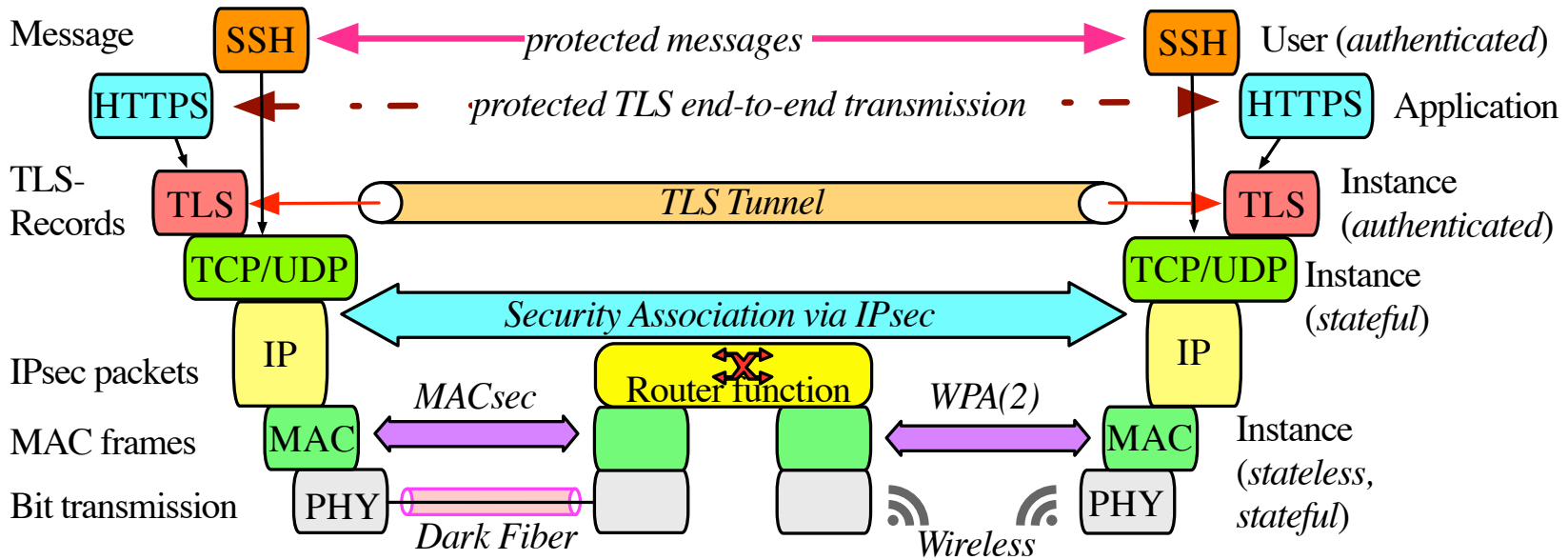

Figure: IT security protocols in a layered view

↪ The cryptographic framework and routines are also used for SSH, PGP, IPSec and WiFi (WPA2), though in a different context.

Development of cryptographic standards and protocols

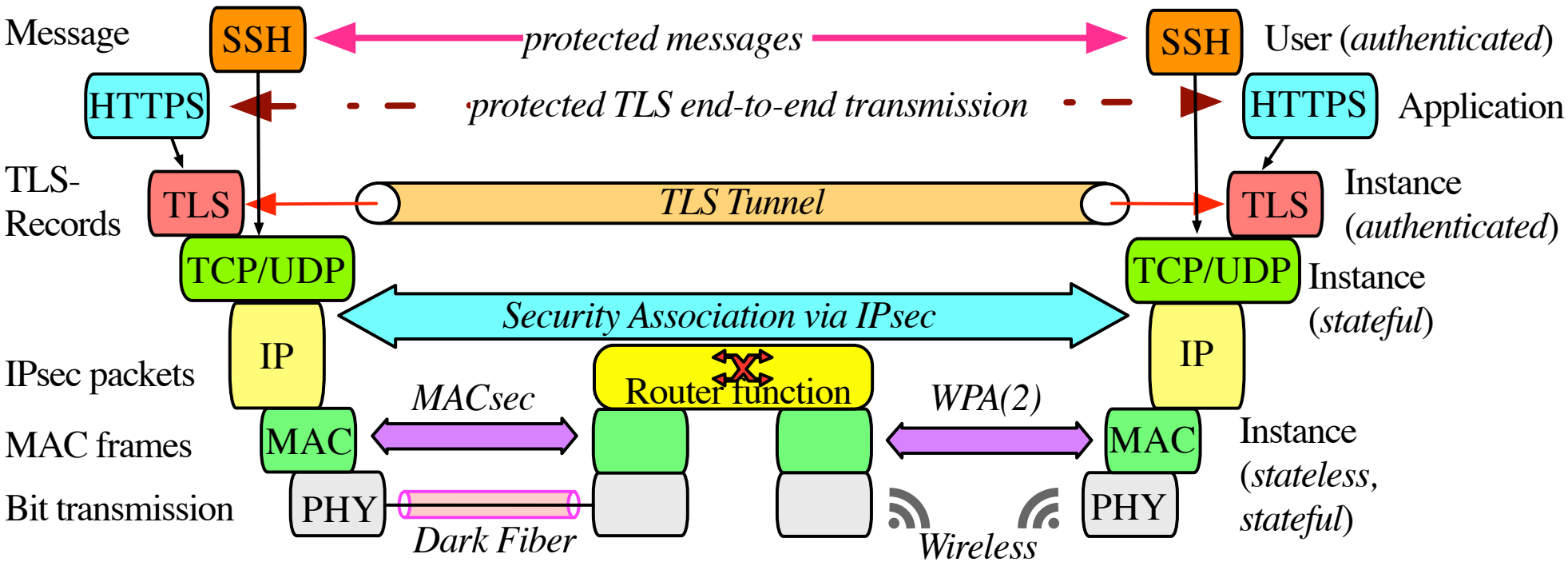

Figure: Cryptographic standards since the Unix epoch

The #4 Crypto Primitives

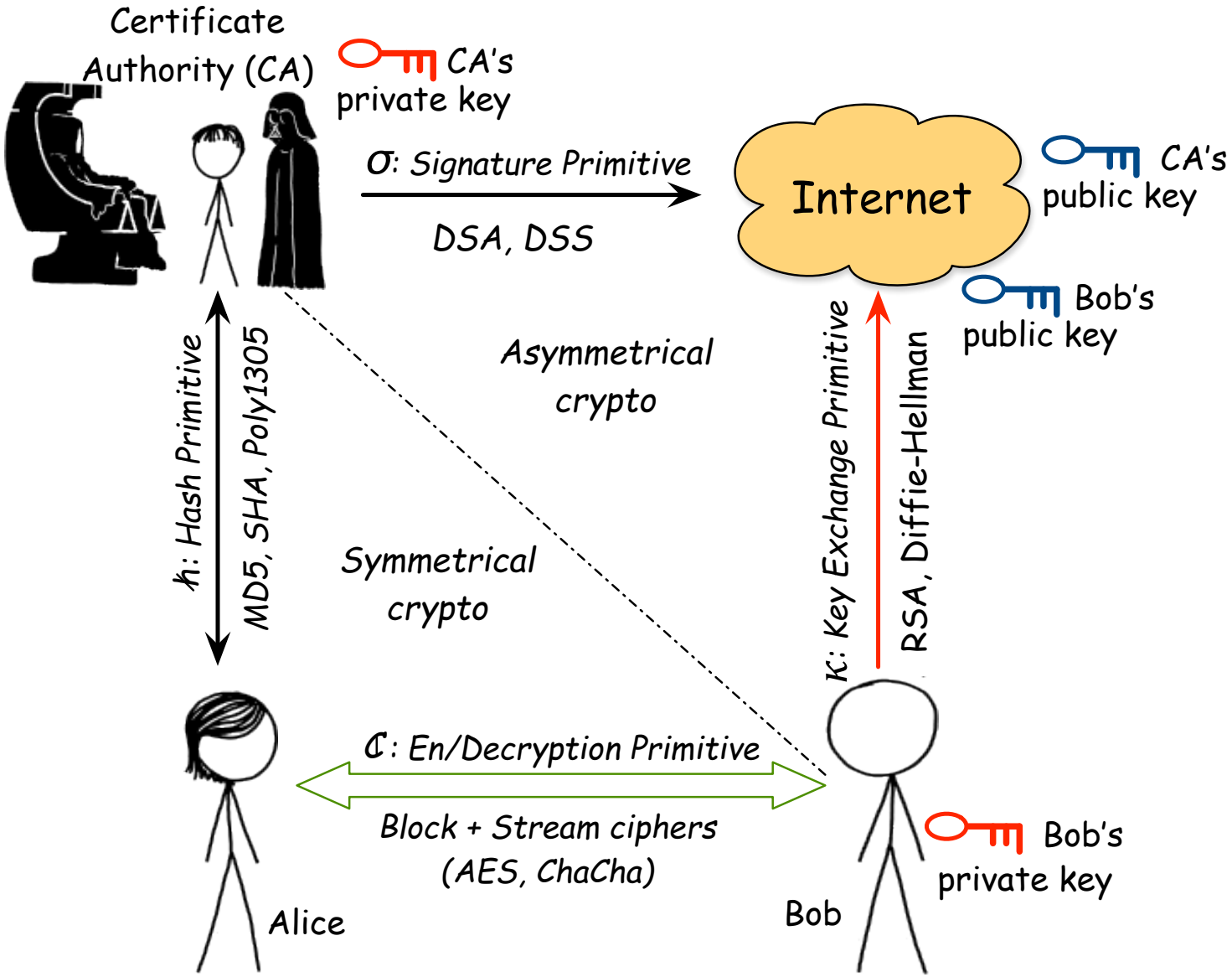

Figure: The #4 crypto primitives

TLS Use Cases

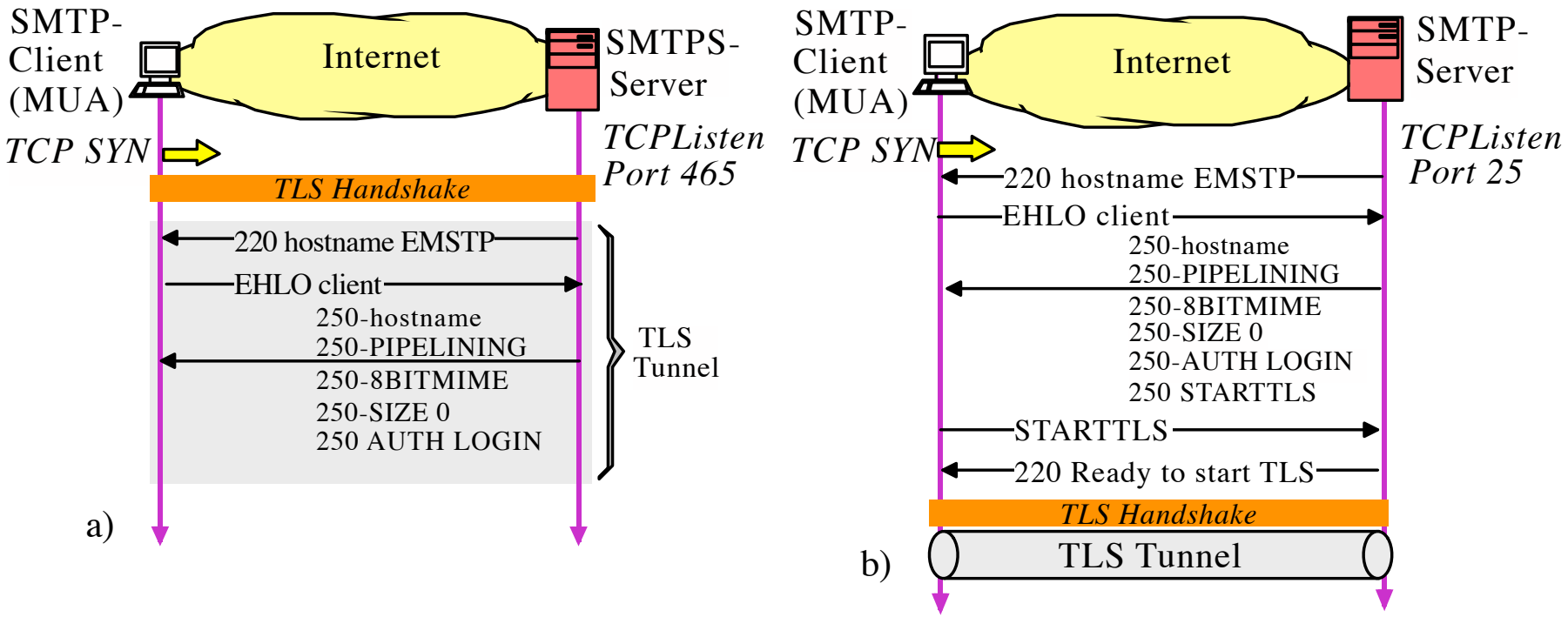

Figure: Immediate TLS and delayed TLS encryption; aka STARTTLS / STLS

- Immediate/mandatory TLS encryption: HTTPS, SMTPS, LDAPS, IMAPS, POP3S, QMTPS
- Delayed/optional TLS encryption: ESMTP + StartTLS, POP3 + STLS

TLS Networking Layering

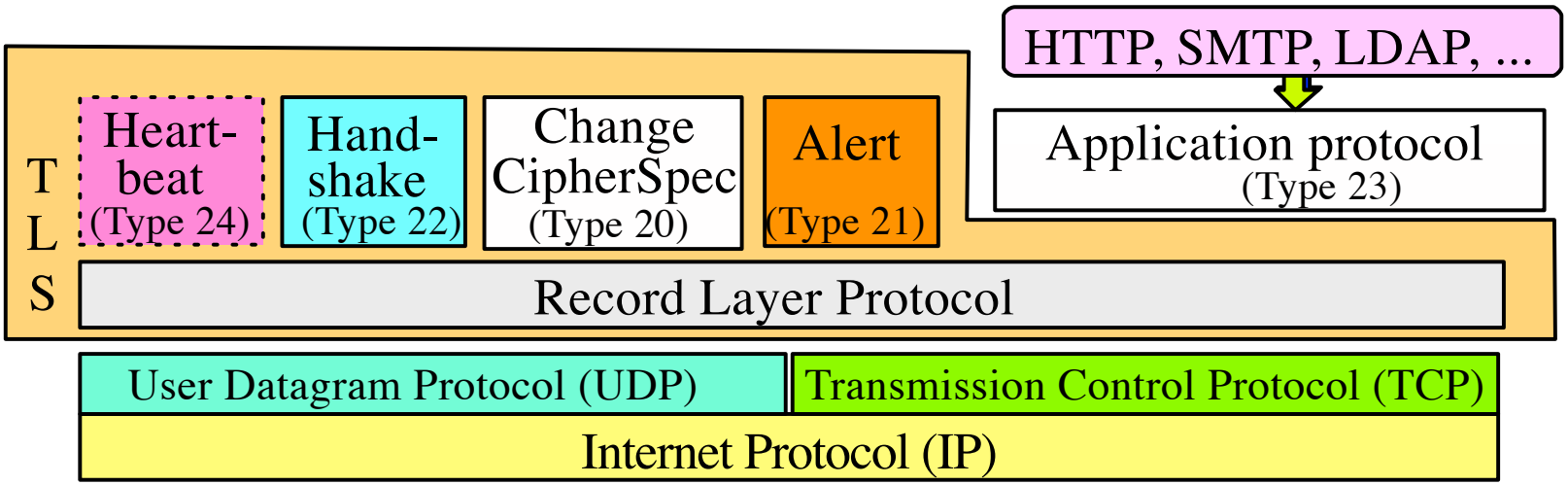

Figure: TLS protocol layering

- TLS is located on top of TCP/UDP (layer 4+5) and below the application layer (7).
- It provides a 'mini' layering.
- The **Record** layer can be considered as transmission layer: The workhorse.
- **Handshake**, **Change Cipher** and **Alert** messages are mostly un-encrypted (using a NULL-encryption).
- The **Application** messages are encrypted and protected.
- The **Heartbeat** Protocol was added in 2011 with only rough evaluation (and fatal consequences).

The #4 Crypto Primitives within TLS < 1.3: The Handshake

TLS < 1.3 provides three different ways of handshakes:

- **RSA** handshake using static RSA public and private keys (the RSA public key is part of the X.509 certificate).
- **Diffie-Hellman** using the *Discrete Logarithm* algorithm (DHE) while providing ephemeral keys. The DH parameters can be configured and changed frequently.
- **Diffie-Hellman** with *Elliptic Curve Cryptography* (ECDHE). Curves and the DH params (as starting point for the calculation are typically fixed by the implementation.

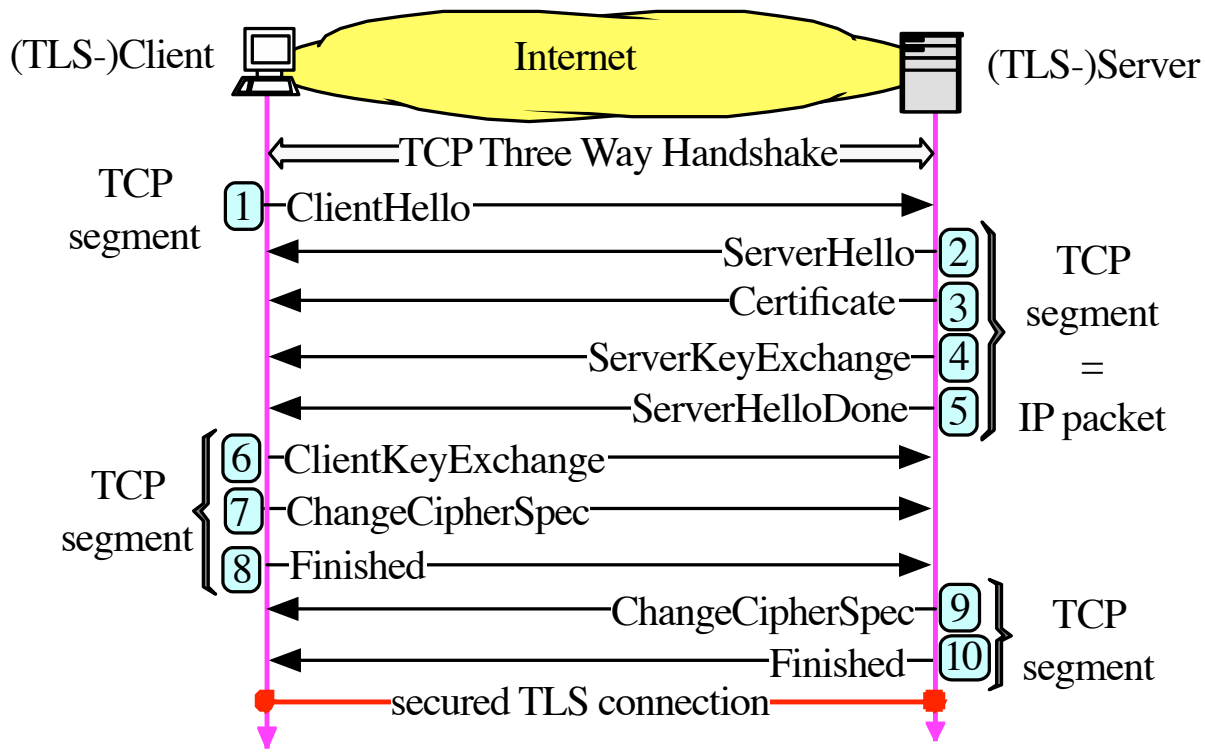

Figure: TLS handshake message exchange

The #4 Crypto Primitives within TLS < 1.3: Encryption

TLS as the successor of SSL supports different sets of symmetrical de/encryption:

Block ciphers:

- 64 bit key-length ~~DES~~
- 128 bit key-length ~~3DES~~
- 128, 256, 384, 512 bit key-length AES

Stream ciphers:

- 40 + 128 bit key-length ~~RC4~~
- ChaCha20 (with TLS 1.2)

↪ The Block Ciphers are often used in 'streaming mode': CBC, OFB, GCM.

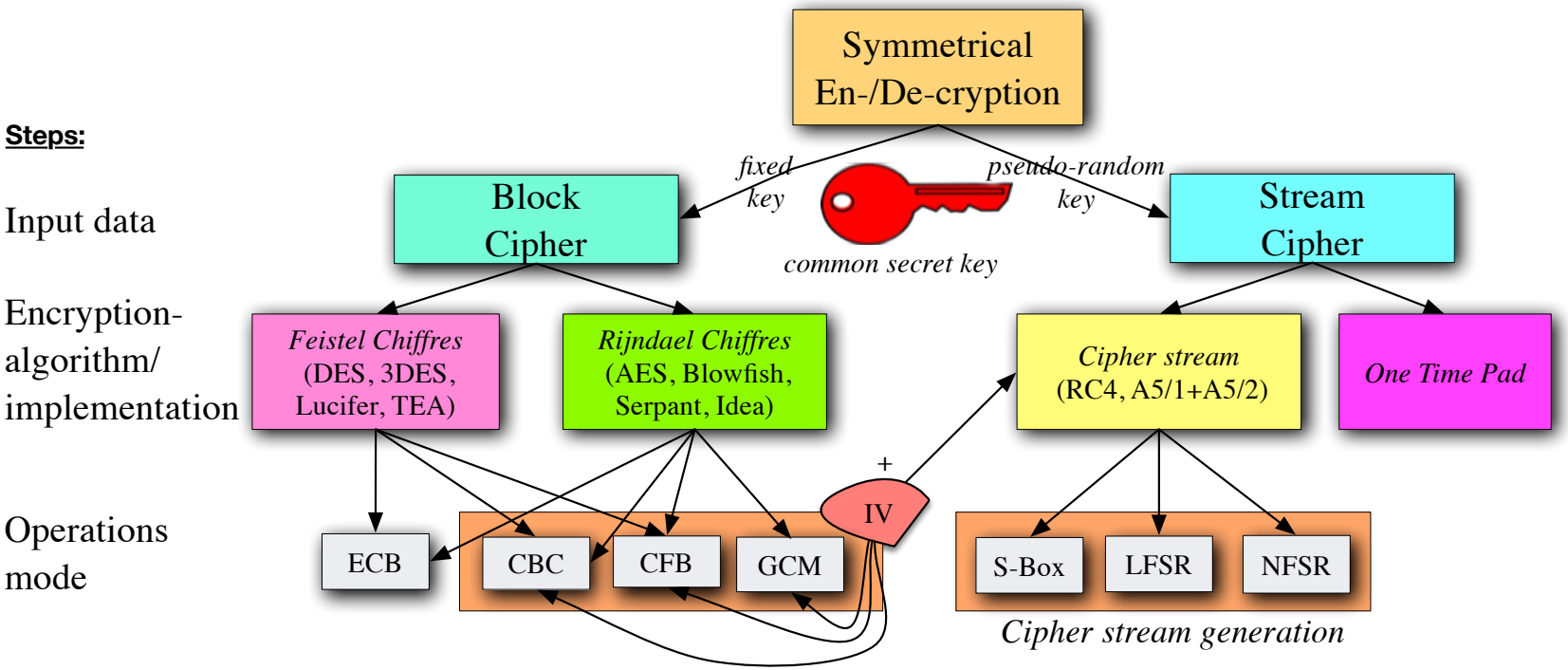

Figure: TLS symmetric encryption

The #4 Crypto Primitives within TLS < 1.3: Integrity Check

- TLS (1.2) uses a keyword authenticated hash to provide authenticity and integrity for the data: *Message Authentication Code* (keyed MAC).
- Since TLS employs both stream and block encryption, hashing is done per TLS record prior of encryption: *MAC-then-encrypt* (MtE).

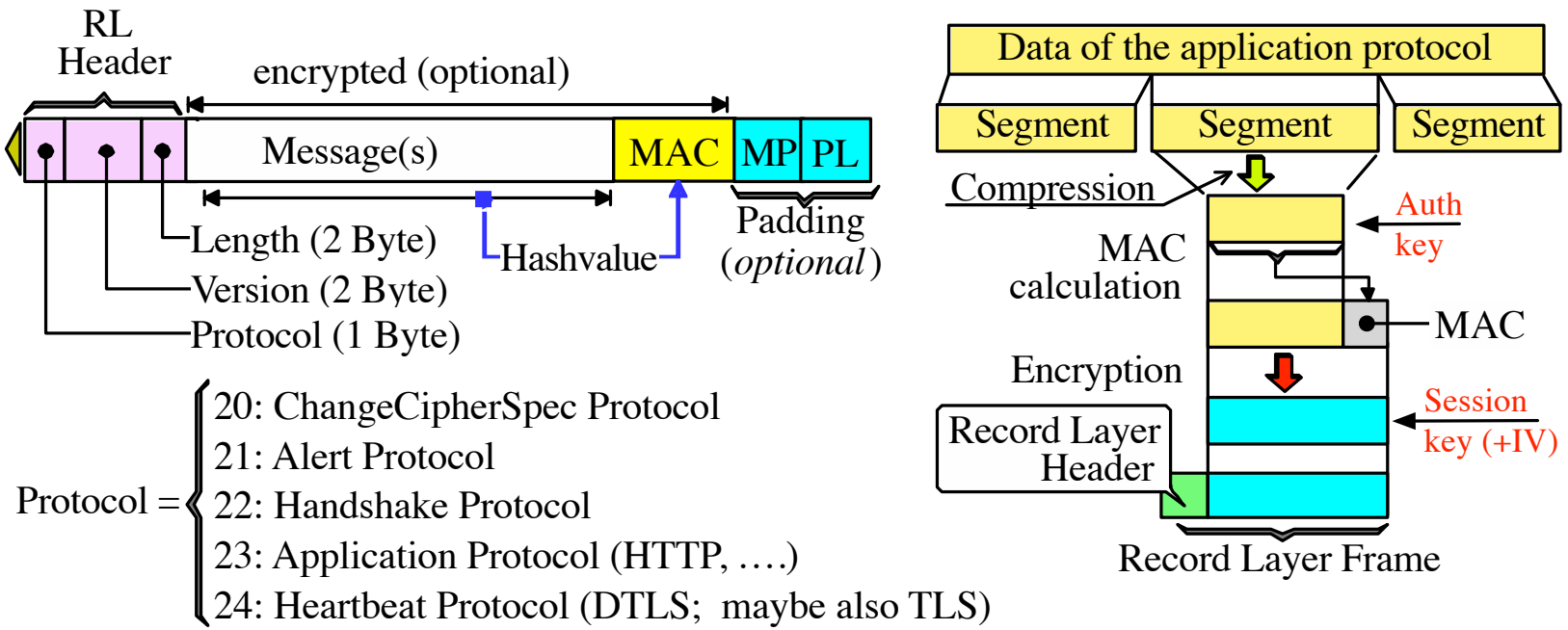

Figure: Checking the integrity of transmitted data

The #4 Crypto Primitives within TLS < 1.3: Cipher Suites

- Crypto primitives used for the current session are provided as Cipher Suite.
- The TLS protocol enumerates the sets fo crypto primitives.

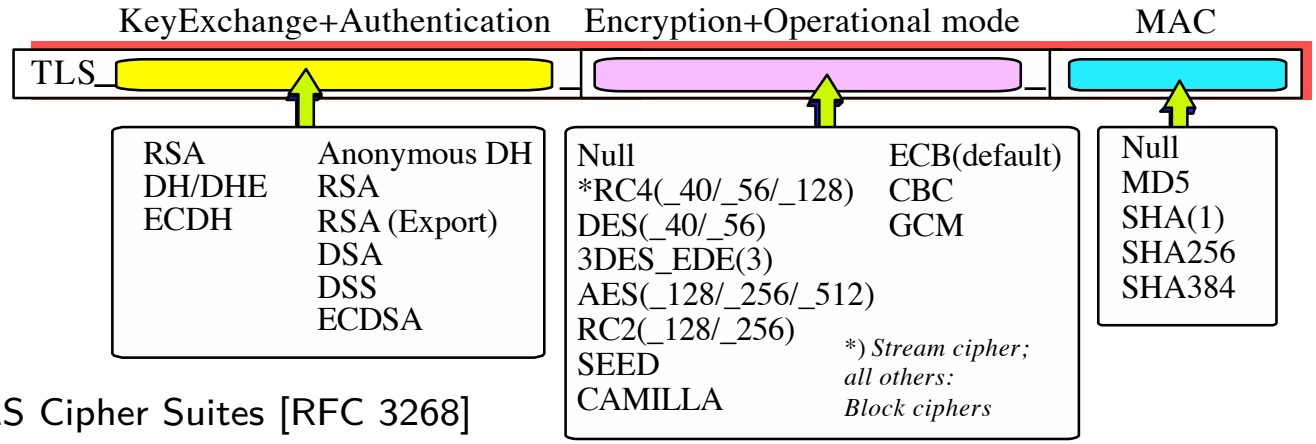

Figure: TLS Cipher Suites [RFC 3268]

```
CipherSuite TLS_RSA_WITH_AES_128_CBC_SHA      = { 0x00, 0x2F };
CipherSuite TLS_DH_DSS_WITH_AES_128_CBC_SHA   = { 0x00, 0x30 };
CipherSuite TLS_DH_RSA_WITH_AES_128_CBC_SHA   = { 0x00, 0x31 };
CipherSuite TLS_DHE_DSS_WITH_AES_128_CBC_SHA  = { 0x00, 0x32 };
CipherSuite TLS_DHE_RSA_WITH_AES_128_CBC_SHA  = { 0x00, 0x33 };
CipherSuite TLS_DH_anon_WITH_AES_128_CBC_SHA  = { 0x00, 0x34 };

CipherSuite TLS_RSA_WITH_AES_256_CBC_SHA      = { 0x00, 0x35 };
CipherSuite TLS_DH_DSS_WITH_AES_256_CBC_SHA   = { 0x00, 0x36 };
CipherSuite TLS_DH_RSA_WITH_AES_256_CBC_SHA   = { 0x00, 0x37 };
CipherSuite TLS_DHE_DSS_WITH_AES_256_CBC_SHA  = { 0x00, 0x38 };
CipherSuite TLS_DHE_RSA_WITH_AES_256_CBC_SHA  = { 0x00, 0x39 };
CipherSuite TLS_DH_anon_WITH_AES_256_CBC_SHA  = { 0x00, 0x3A };
```

TLS < 1.3: Keymaterial and Keys

- TLS uses a variety of hash functions to derive *deterministically* the **PreMasterSecret** and the Session keys from the input material.
- Hash functions are used to 'diffuse' the input and thus pseudo random sequences with significant entropy are the result.
- This is called a '*Pseudo Random Function*' PRF.

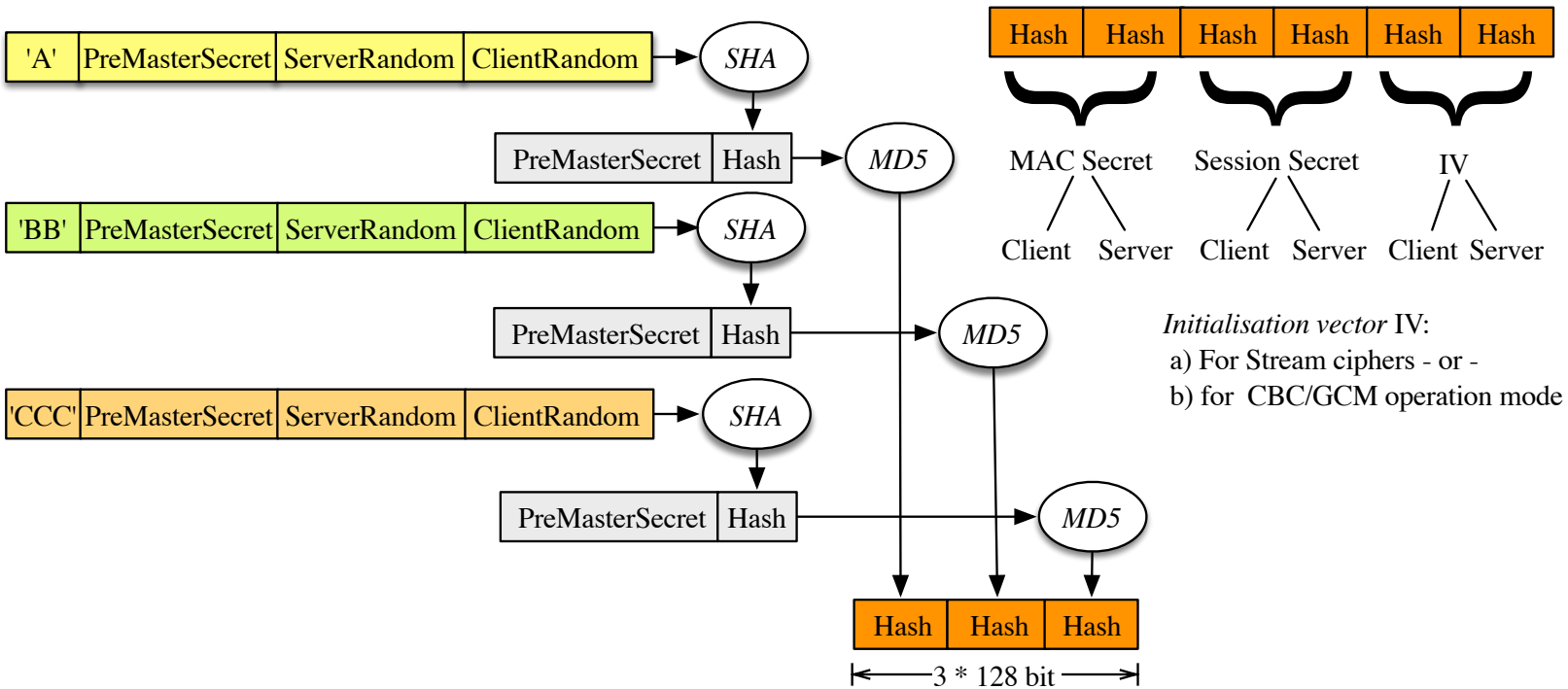

Figure: Generating the PreMaster and the Session secrets

↪ Given RSA encryption, the only 'random' number is the client's random provided during the handshake. For Diffie-Hellman, client and server deliver a random value.

TLS < 1.3: Weaknesses

- *Backward compatibility* with ancient SSL 2.0 (older versions): *Renegotiation*.
- '*Export ciphers*' suited for NSA de-cryption.
- Control messages (Alerts, CCS) transmitted in *clear text*.
- No particular *state model*, thus attacker can interrogate at any point.
- Some Cipher suites (in particular with *CBC*) are weak or even bad.
- Very little control on *negotiated Cipher suites* (`mod_ssl`).
- Handshake is *slow*.
- *Session Resumption* with persistent data.
- Handshake can be broken up (MitM).
- MtE delivery valuable information for a hacker given its data decryption.

$\hookrightarrow$ TLS (as successor of SSL) is broken by design; it is 'just' working and deployed ... *everywhere*.

TLS 1.3 – Summary

After years of discussion, the IETF released TLS 1.3 in RFC 8446 (and RFC 8447.

- The TLS 1.3 core is a complete redesign.
- Compatibility with older TLS version is provided supporting its 'skeleton' behaviour and faking a TLS 1.2 handshake.
- As already used in previous TLS releases, TLS 1.3 uses the 'Hello Message' to transport the new parameters.
- ECDHE key exchange is mandatory and the only one!
- TLS uses only a very restricted set of Cipher Suites.
- Those Ciphers Suites (apart from AES and GCM) are mainly based on *Dan Bernstein*'s developments.
- TLS 1.3 protects the negotiated data as soon as it is possible during the handshake.
- 'Early' Application Data can be transmitted in the handshake.
- The handshake in TLS 1.3 is much more efficient and – using Session Resumption – provides a `0RTT` capability.

↪ Though trying to cope with older TLS versions, TLS 1.3 is a different beast. Some 'Middleboxes' don't allow to let the TLS 1.3 traffic through, since it does not look like as expected.

TLS 1.3 – Handshake

The TLS 1.3 is much more efficient using an early encryption scheme.

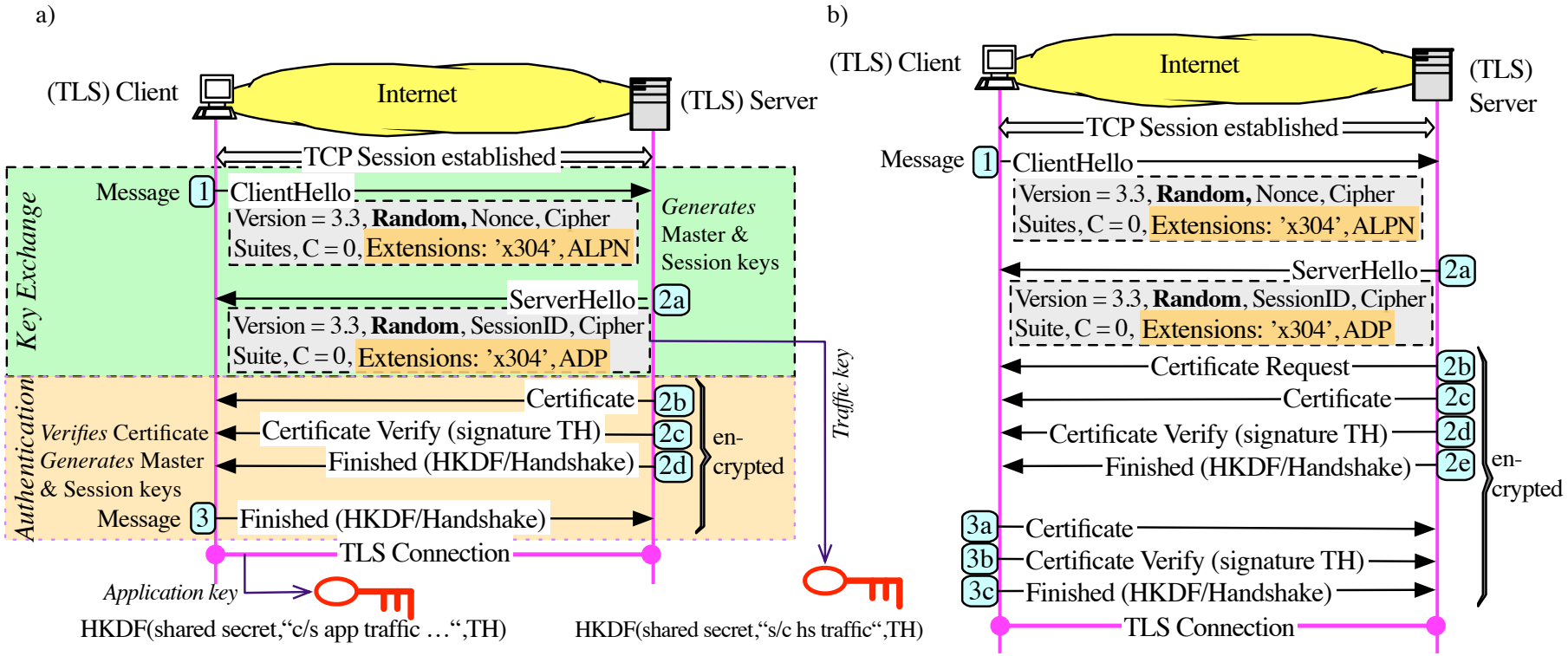

Figure: TLS 1.3 Handshake; (a) without and (b) with Client Certificate Request; ALPN: Application Layer Protocol Notifications

Only three messages are exchanged:

Client → Server The (unencrypted) Client `Hello` message.

Server → Client The Server `Hello` message: The first part including protocol artefacts in clear text; the further parts are encrypted with a provisional secret (Traffic Key) covering in particular the X.509 cert.

Client → Server The encrypted `Finish` message, telling that the Application Key is ready for use.

TLS 1.3 – Cipher Suites

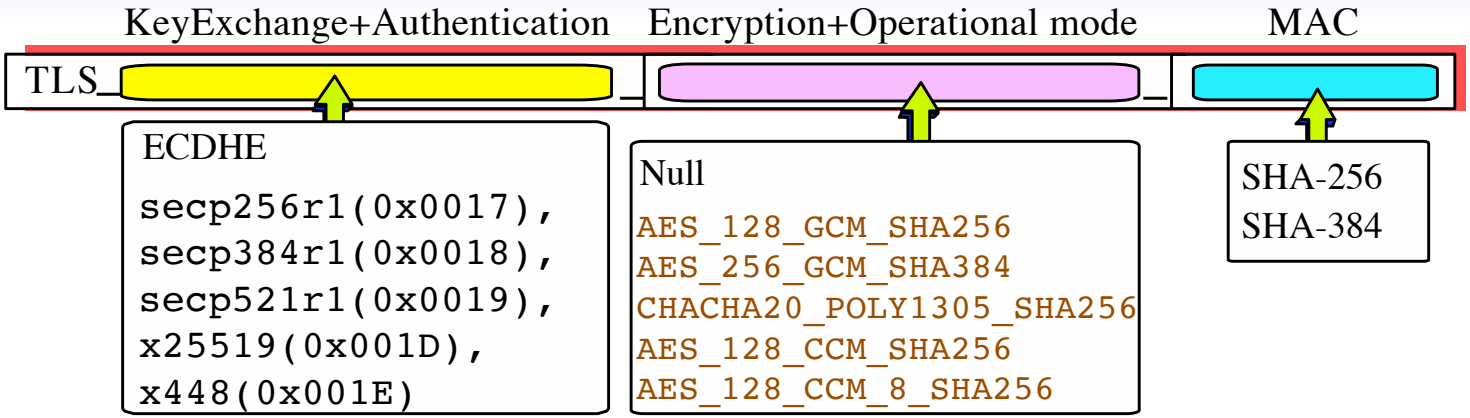

Figure: TLS 1.3 Cipher Suites

```
CipherSuite    TLS_AES_128_GCM_SHA256          {0x13,0x01}
CipherSuite    TLS_AES_256_GCM_SHA384          {0x13,0x02}
CipherSuite    TLS_CHACHA20_POLY1305_SHA256    {0x13,0x03}
CipherSuite    TLS_AES_128_CCM_SHA256          {0x13,0x04}
CipherSuite    TLS_AES_128_CCM_8_SHA256        {0x13,0x05}
```

Note: `CCM` = Cipher Block Chaining - Message Authentication Code (`CBC-MAC`)

↪ No particular Authentication method is indicated. Apart from PSK all Transcript Messages are authenticated requiring a valid server X.509 certificate.

```
openssl ecparam -list_curves
```

TLS 1.3 – Record Layer

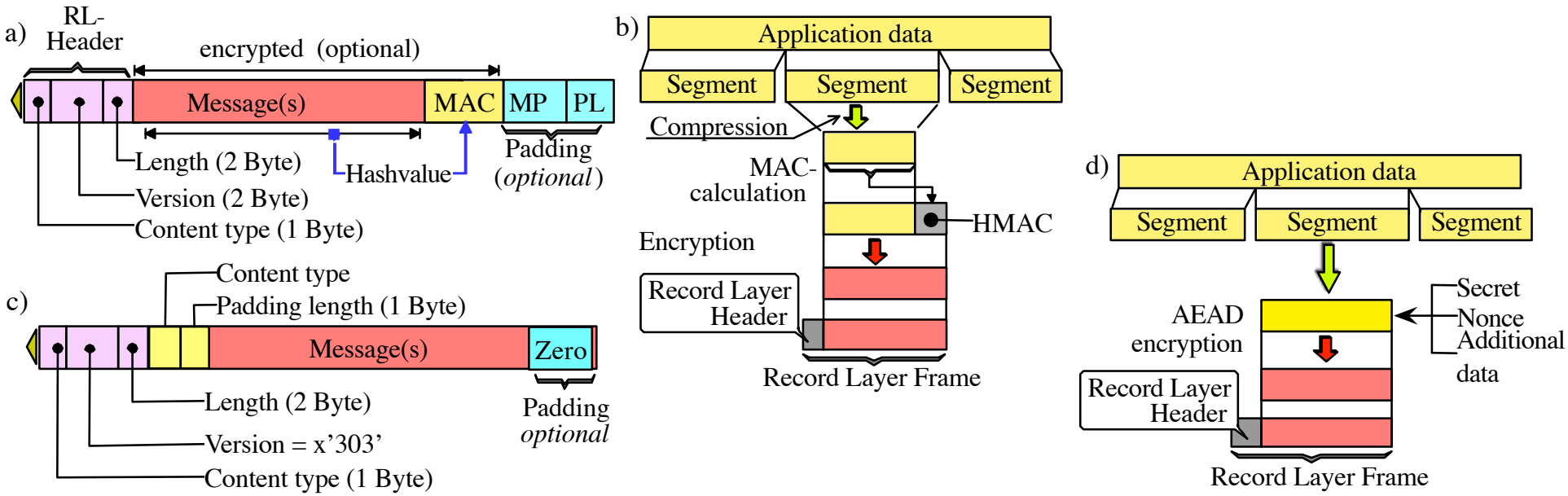

Figure: TLS 1.3 a) Record layer structure w and c) w/o MAC, b) Record with MAC, d) Record with AEAD

TLS 1.3 – Keys

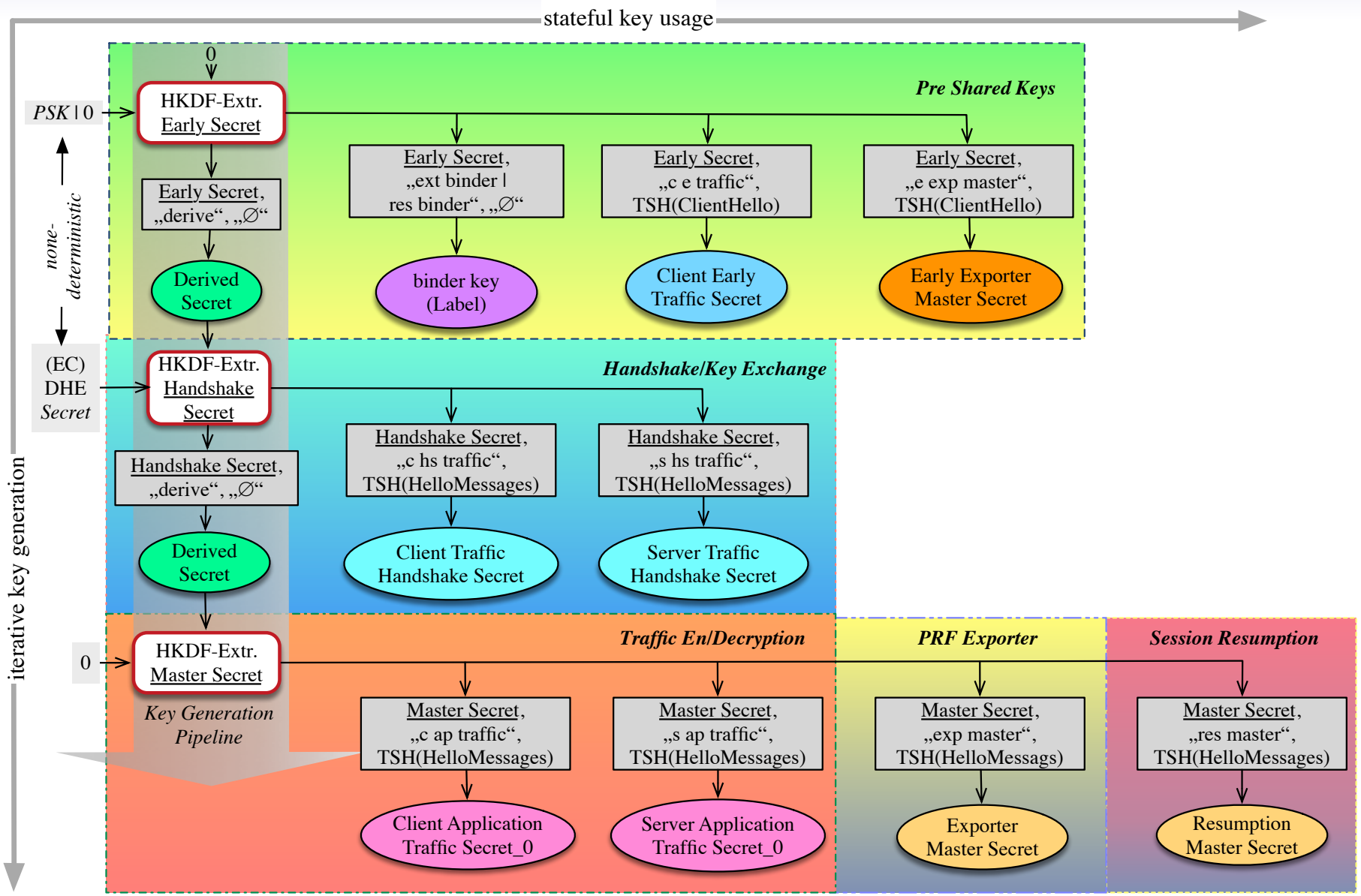

Figure: TLS 1.3 Key generating procedure using HKDF

TLS 1.3 – PSK & 0RTT and Grease

Session Resumption is possible within TLS 1.3 provided, both client and server store the negotiated secret (persistently):
Pre-shared Keys (PSK).

The PSK secret is indicated in the Client Hello message providing a hash of the PSK and coupled with the identity of the client.

↪ In this case, a TLS session can literally succeed in one step, thus 0-RTT. However, this breaks
Perfect Forward Secrecy (PFS)

Grease:
Generate Random Extensions And Sustain Extensibility can applied by client or server to 'test' the counterparts TLS 1.3 capabilities indicating 'invalid' Cipher Suites in the Hello message.

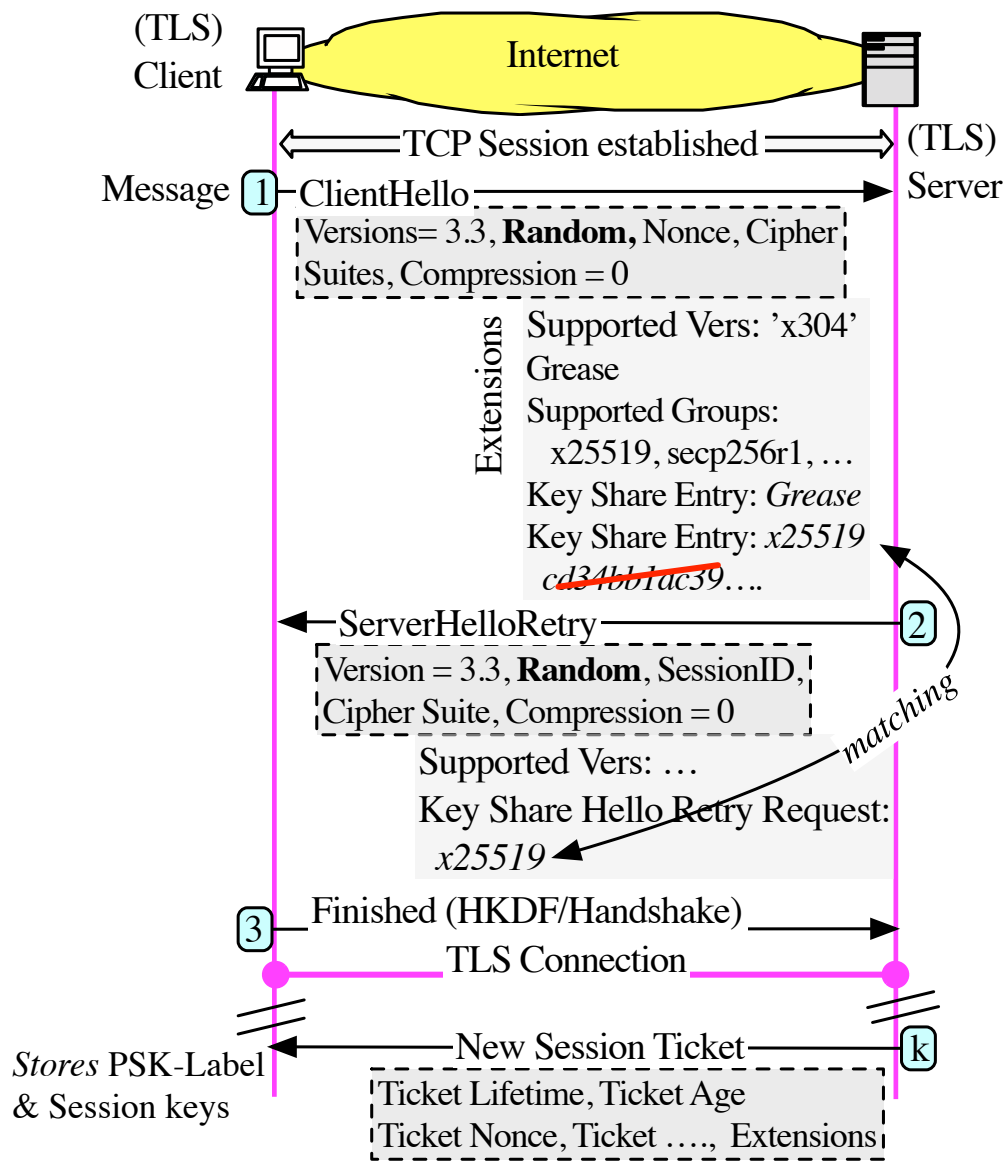

Figure: TLS 1.3 using Pre-shared Keys

TLS 1.3 – 1RTT

Session establishment can be accelerated in case both client and server don't need to negotiate the Cipher, but rather provide a quick focus: `1-RTT`.

In this case, the client indicates to the server to know already the (otherwise transmitted) DH-Params and the chosen curve (for ECC) for a quick negotiation.

↪ However, the server may have changed its DH-Params, thus this procedure would not work out. Rather, in this case, the server sends a `Hello Retry` message telling the new DH-Params. This procedure obviously does not violate PFS, since only protocol artefacts are provided by the client.

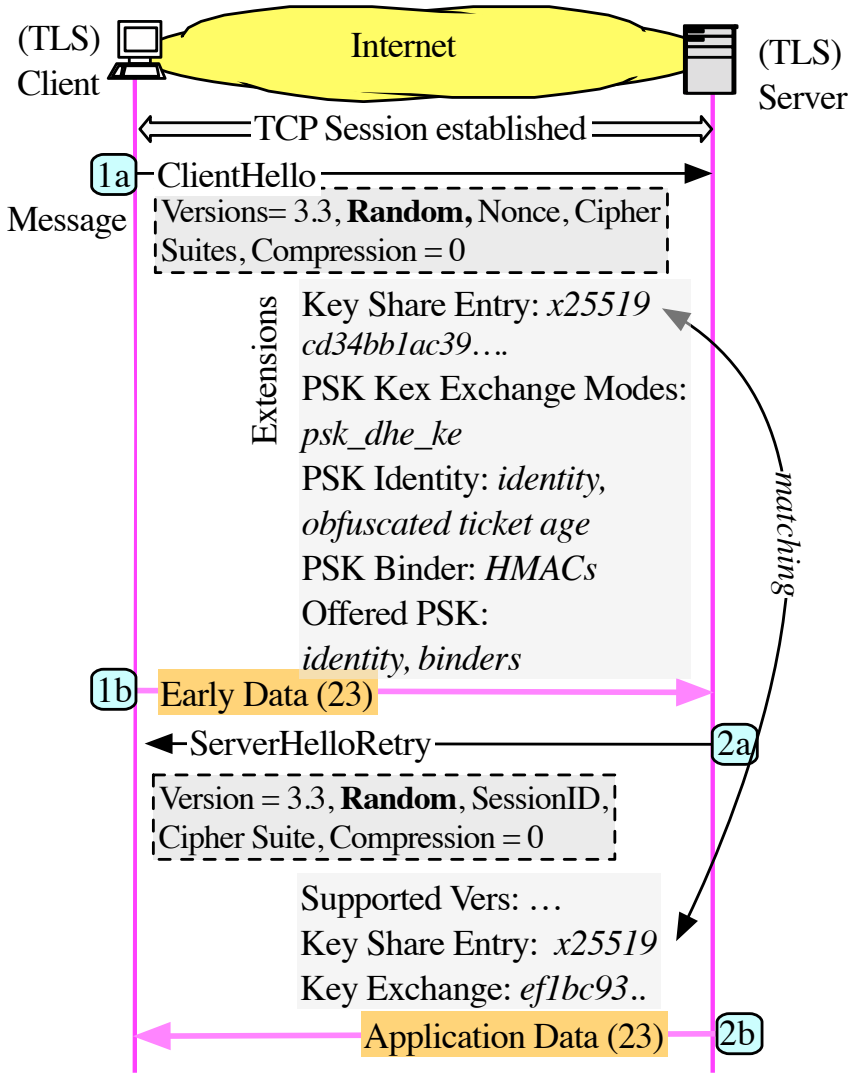

Figure: TLS 1.3 using 1-RTT message exchange

Commercial spot

Figure: Technik der IP-Netze (4th edition)

Implementation

OpenSSL 1.1.0:

- Install **OpenSSL 1.1.1** on OmniOSce – however without overwriting previous Libs.
- How to tell which OpenSSL is installed?

fehQlibs + ucspi-ssl:

- Install **fehQlibs** (under `/usr/local`).
- Install **ucspi-ssl** using fehQlibs and OpenSSL 1.1.1.
- How can you tell that OpenSSL 1.1.1 is in use for a client/server?

testssl:

- Install *Dirk Wetter*'s **ssltest** from `https://github.com/drwetter/testssl.sh`
- Test TLS 1.3. for some sites (including Google and fehcom).

WireShark:

- Install **WireShark** (> 2.5).
- Record a TLS 1.3 connection setup and do an interpretation of the negotiation.

Bibliography

- RFC 8446
- RFC 8447
- `https://tools.ietf.org/html/draft-irtf-cfrg-eddsa-08`