4.) Let p be a prime with

$$p \bmod 4 = 3$$

and let $a \in \mathbb{Z}_p^*$ be a square.

Show that the square roots of a are given by $\pm a^{\frac{p+1}{4}}$.

$a = b^2$, square root of a are $\pm b$ $\quad x^2 - a = x^2 - b^2 = (x-b)(x+b)$

$$\left(a^{\frac{p+1}{4}}\right)^2 = b^{2 \cdot \frac{p+1}{4} \cdot 2} = b^{p+1} = b^{p-1} \cdot b^2 = b^2 = a$$

5.) Let $G = E(\mathbb{F}_q)$ denote the group of points of an elliptic curve. (Or just any finite abelian group G.) Proof the following:

(i) For every $P \in G$ there exits some $k \in \mathbb{N}$ such that $k \cdot P = \mathcal{O}$. The smallest $k \in \mathbb{N}$ with $k \cdot P = \mathcal{O}$ is called the order of P and denoted by:

$$o(P) \;:=\; \min\{k \in \mathbb{N} \mid k \cdot P = \mathcal{O}\}$$

(ii) The cyclic subgroup generated by $P \in G$

$$\langle P \rangle \;:=\; \{k \cdot P \mid k \in \mathbb{N}\}$$

contains exactly $o(P)$ many elements, namely:

$$P, \quad 2 \cdot P, \quad 3 \cdot P, \ldots, \quad (o(P)-1) \cdot P, \quad o(P) \cdot P = \mathcal{O}$$

(iii) If $k \cdot P = \mathcal{O}$ then $o(P) \mid k$.

(iv) If $Q \in \langle P \rangle$ then $o(Q) \mid o(P)$.

(v) If $\gcd(k, o(P)) = 1$ then $o(k \cdot P) = o(P)$ and $\langle k \cdot P \rangle = \langle P \rangle$.

(vi) If $\gcd(o(P), o(Q)) = 1$ then $P, Q \in \langle P + Q \rangle$ and $o(P + Q) = o(P) \cdot o(Q)$.

$\mathcal{O}, P, 2P, 3P, \ldots, n \cdot P = k \cdot P$ with $k < n$ take minimal n

$\mathcal{O} = nP - kP = (n-k) \cdot P \quad 1 < n-k \leq n$

because of the minimality of n

$\Rightarrow n - k = n \quad \Rightarrow k = 0$

$\Rightarrow n \cdot P = \mathcal{O}$

$\langle P \rangle = \{\mathcal{O}, P, \ldots, (n-1) \cdot P\}$

$|\langle P \rangle| = o(P)$

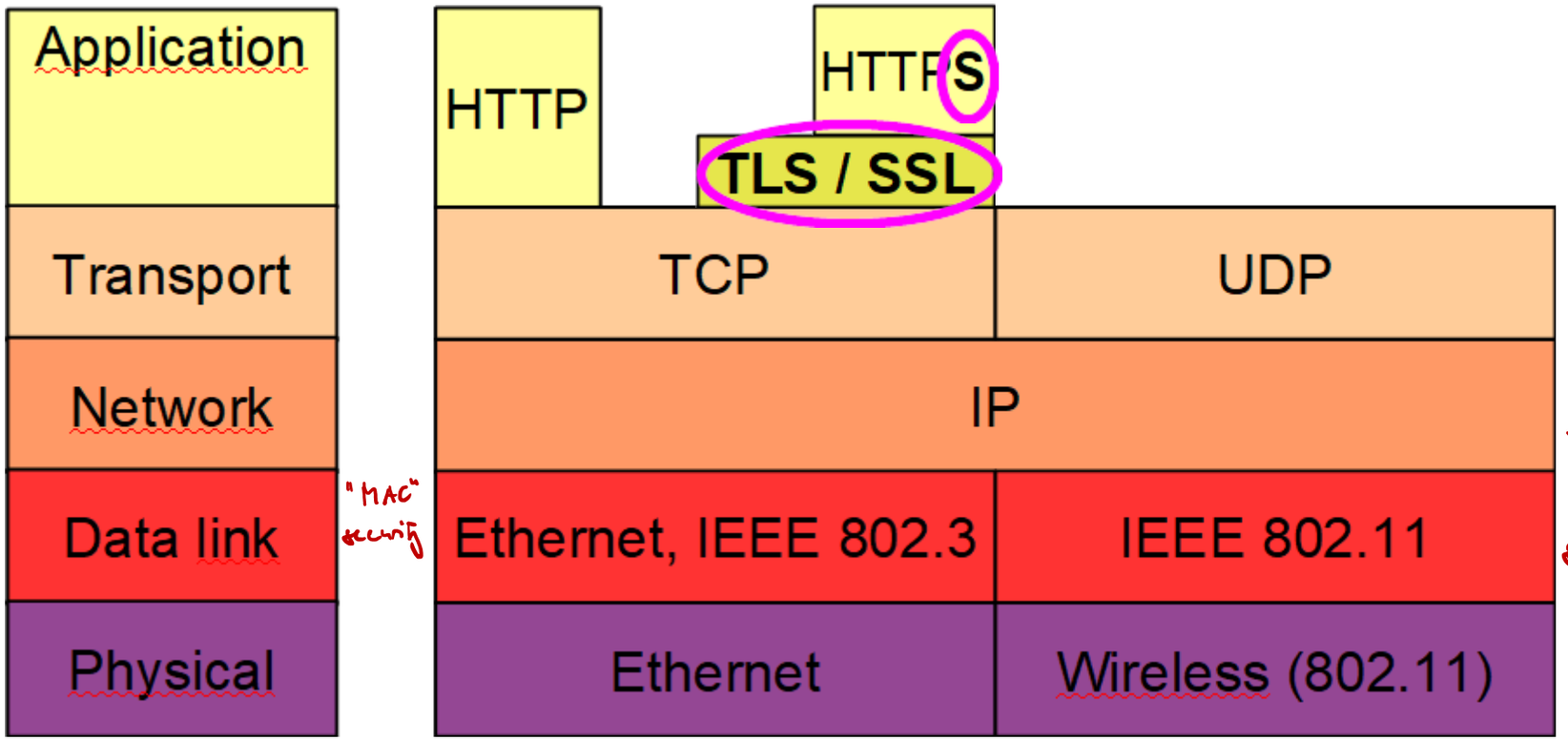
TECHNISCHE HOCHSCHULE
OSTWESTFALEN-LIPPE
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS

Network Security

Transport Layer Security (TLS)

Prof. Dr. Stefan Heiss

Security at Layer 4++

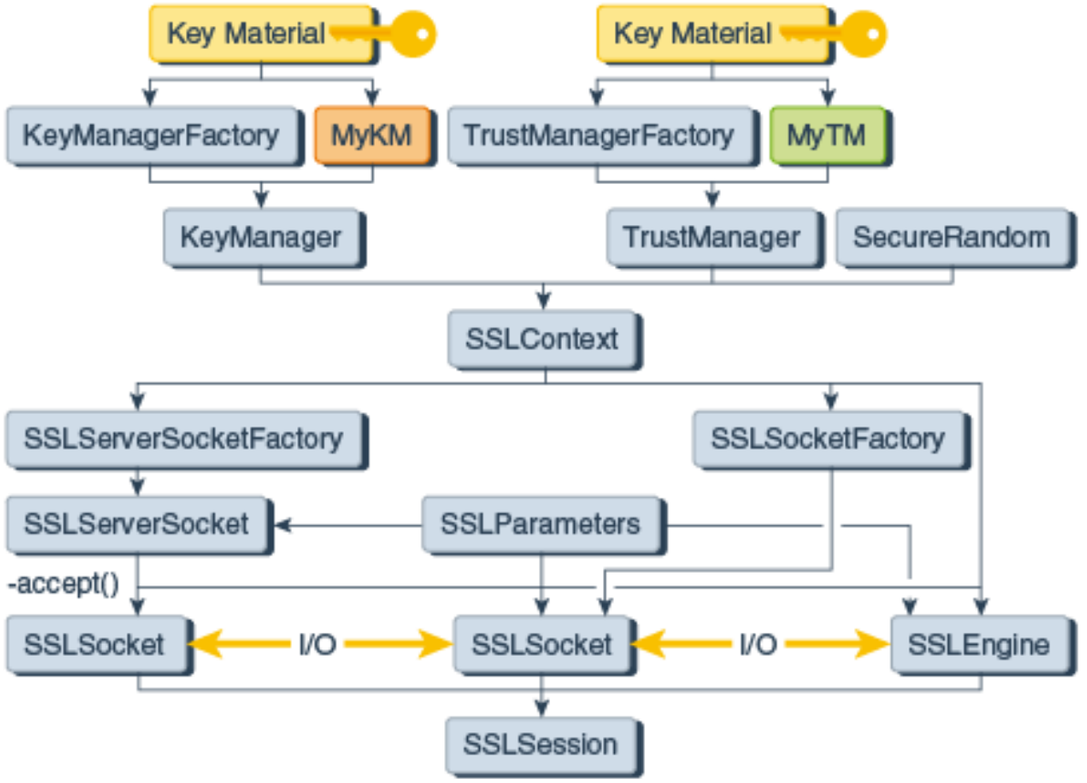


- **Secure Sockets Layer** Protocols
  - **SSL 1.0**:Netscape Communications Corp., 1994
  - **SSL 2.0**:The SSL Protocol, Netscape Communications Corp., Feb. 1995
  - **SSL 3.0**:The SSL 3.0 Protocol, Netscape Communications Corp., Nov. 1996

- **Transport Layer Security** Protocols
  - **TLS 1.0 = SSL 3.1:** The TLS Protocol - Version 1.0, **RFC 2246,** Jan. 1999
  - **TLS 1.1:** The Transport Layer Security (TLS) Protocol - Version 1.1, **RFC 4346,** April 2006
  - **TLS 1.2:**The Transport Layer Security (TLS) Protocol - Version 1.2, **RFC 5246,** August 2008
  - **TLS 1.3:**The Transport Layer Security (TLS) Protocol - Version 1.3, **RFC 8446,** August 2018

Java‘s TLS Implementation

- Java Secure Socket Extension (JSSE)
- JSSE Reference Guide (Chapter 8 in Security Developer’s Guide)
- TLS 1.3 support since Java 11



- **TLS Record Protocol**
  - Privacy: symmetric encryption
  - Integrity: MAC's or AEAD methods
    TLS 1.3
  - Keys are negotiated per connection with the help of the TLS Handshake Protocol
- **TLS Handshake Protocol**
  - Authentication of peer's identities
  - Negotiation of secret session keys

Server usually is authenticated by a certificat / signature (PKI)

Client can rarely authenticated (TLS with Client Authentication)

TLS – Ciphersuites

- **TLS v ≤ 1.2:** <~~Record~~ Handshake Protocol Algos.>**_WITH_**<Record Protocol Algos.>
  - Addition of Kerberos Cipher Suites to Transport Layer Security (TLS) , RFC 2712
  - Addition of SEED Cipher Suites to Transport Layer Security (TLS), RFC 4162
  - Pre-Shared Key Ciphersuites for Transport Layer Security (TLS), RFC 4279
  - **The Transport Layer Security (TLS) Protocol - Version 1.1**, RFC 4346
  - Pre-Shared Key (PSK) Ciphersuites with NULL Encryption for Transport Layer Security (TLS), RFC 4785
  - Using the Secure Remote Password (SRP) Protocol for TLS Authentication , RFC 5054
  - **The Transport Layer Security (TLS) Protocol - Version 1.2**, RFC 5246
  - AES Galois Counter Mode (GCM) Cipher Suites for TLS, RFC 5288
  - TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM) , RFC 5289
  - DES and IDEA Cipher Suites for Transport Layer Security (TLS), RFC 5469
  - Pre-Shared Key Cipher Suites for TLS with SHA-256/384 and AES Galois Counter Mode, RFC 5487
  - ECDHE_PSK Cipher Suites for Transport Layer Security (TLS) , RFC 5489
  - Camellia Cipher Suites for TLS, RFC 5932
  - Addition of the ARIA Cipher Suites to Transport Layer Security (TLS), RFC 6209
  - Datagram Transport Layer Security Version 1.2, RFC 6347
  - Addition of the Camellia Cipher Suites to Transport Layer Security (TLS), RFC 6367
  - AES-CCM Cipher Suites for Transport Layer Security (TLS), RFC 6655
  - AES-CCM Elliptic Curve Cryptography (ECC) Cipher Suites for TLS, RFC 7251
  - ChaCha20-Poly1305 Cipher Suites for Transport Layer Security (TLS), RFC 7905
  - Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier, RFC 8422
  - GOST Cipher Suites for Transport Layer Security (TLS) Protocol Version 1.2, draft-smyshlyaev-tls12-gost-suites

- **TLS v = 1.3:** `<Record Protocol Algos.>`
  - **The Transport Layer Security (TLS) Protocol Version 1.3**, RFC 8446
  - Secure Password Ciphersuites for Transport Layer Security (TLS), RFC 8492
  - ShangMi (SM) Cipher Suites for TLS 1.3, RFC 8998
  - TLS 1.3 Authentication and Integrity only Cipher Suites, draft-camwinget-tls-ts13-macciphersuites
  - GOST Cipher Suites for Transport Layer Security (TLS) Protocol Version 1.3, draft-smyshlyaev-tls13-gost-suites

- Ciphersuite identifiers: Transport Layer Security (TLS) Parameters

TLS 1.3 – Cipher Suites (RFC 8446, Appendix B.4)

A symmetric cipher suite defines the pair of the AEAD algorithm and hash algorithm to be used with HKDF. Cipher suite names follow the naming convention:

```
CipherSuite TLS_AEAD_HASH = VALUE;
```

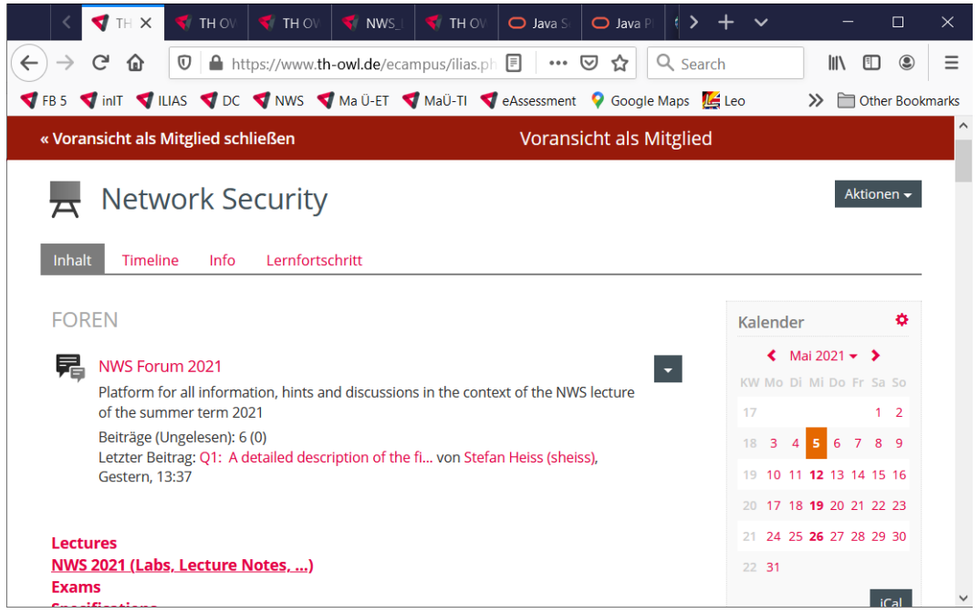
Component	Contents
TLS	The string "TLS"
AEAD	The AEAD algorithm used for record protection
HASH	The hash algorithm used with HKDF
VALUE	The two-byte ID assigned for this cipher suite

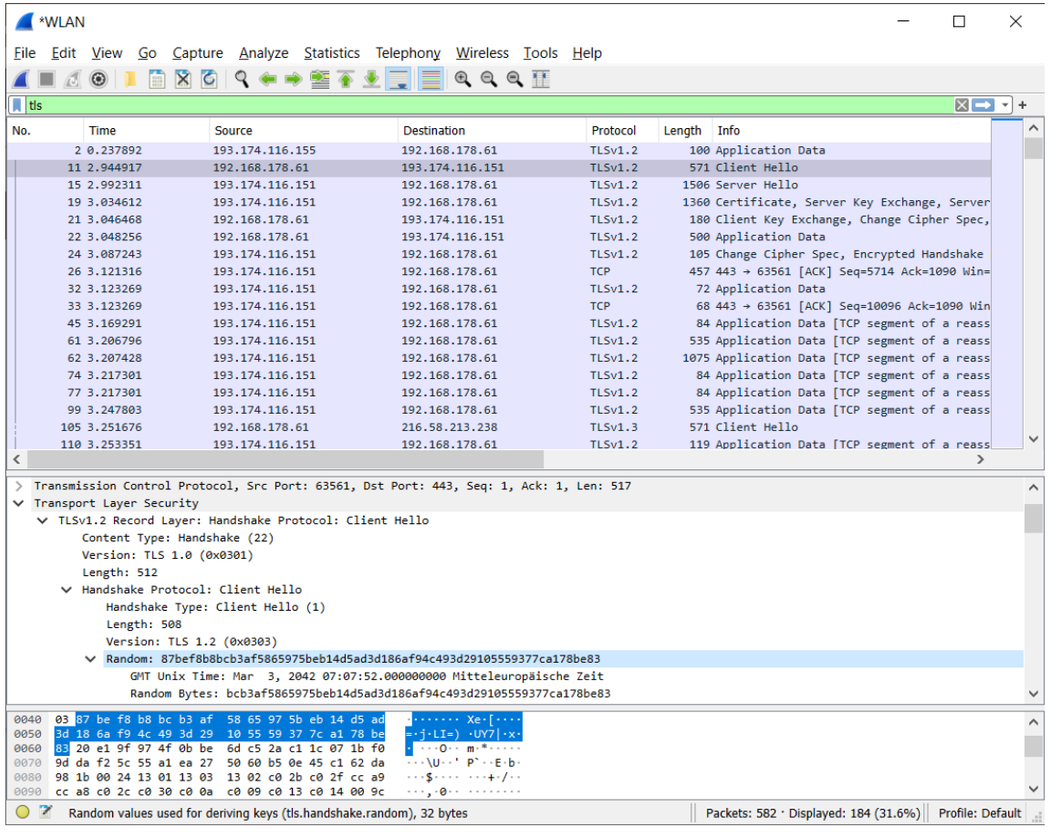
TLS 1.3 – Cipher Suites (RFC 8446, Appendix B.4)

This specification defines the following cipher suites for use with TLS 1.3.

Description	Value
TLS_AES_128_GCM_SHA256	{0x13,0x01}
TLS_AES_256_GCM_SHA384	{0x13,0x02}
TLS_CHACHA20_POLY1305_SHA256	{0x13,0x03}
TLS_AES_128_CCM_SHA256	{0x13,0x04}
TLS_AES_128_CCM_8_SHA256	{0x13,0x05}

Capturing a TLS session

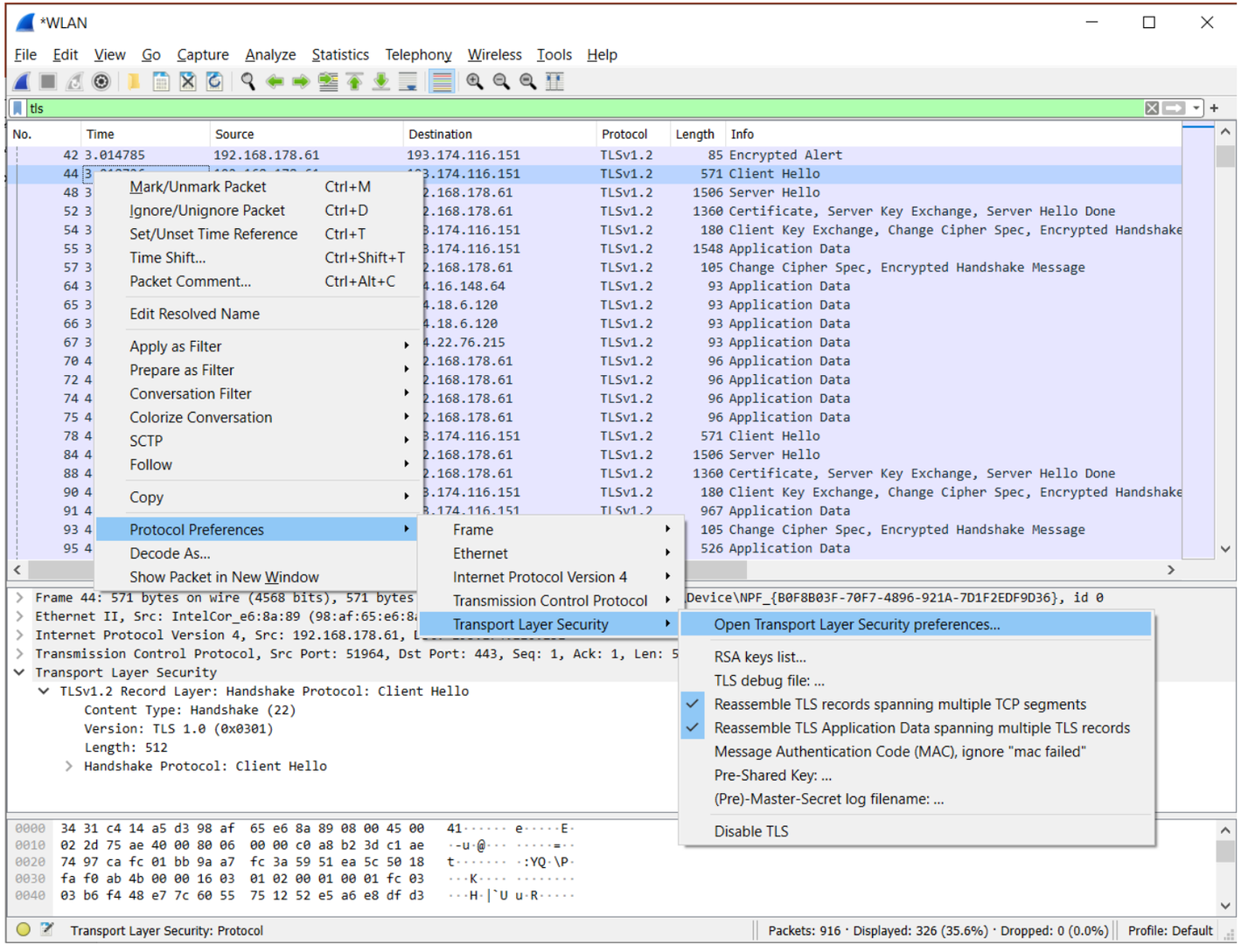


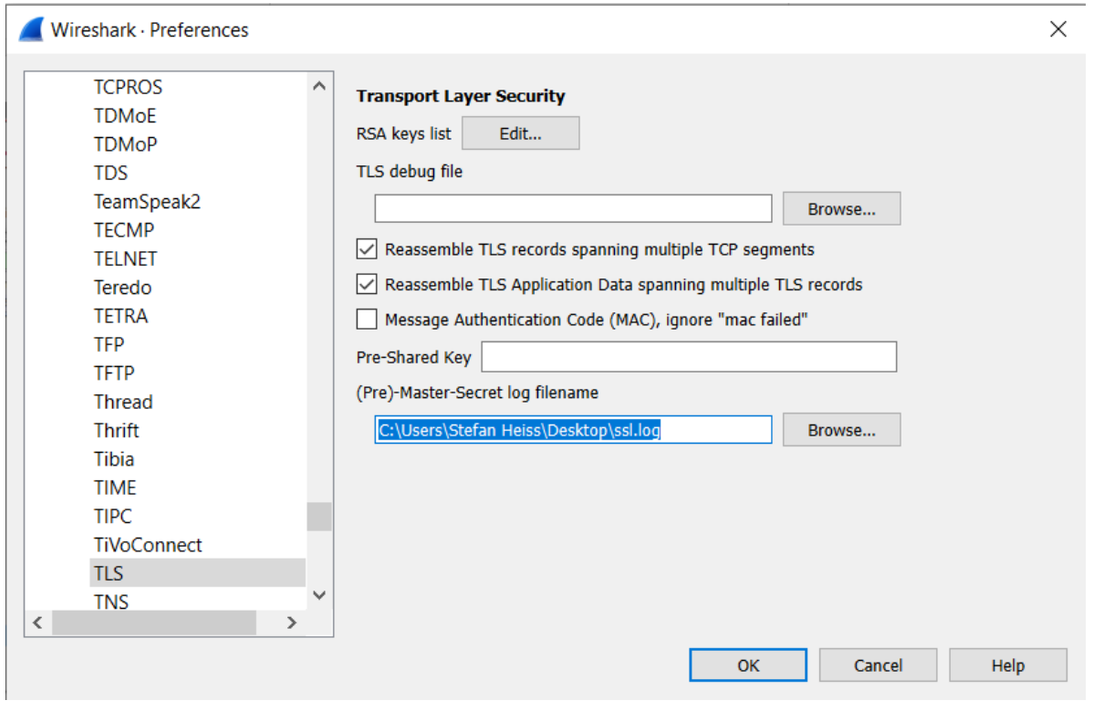


Decrypting a captured TLS session

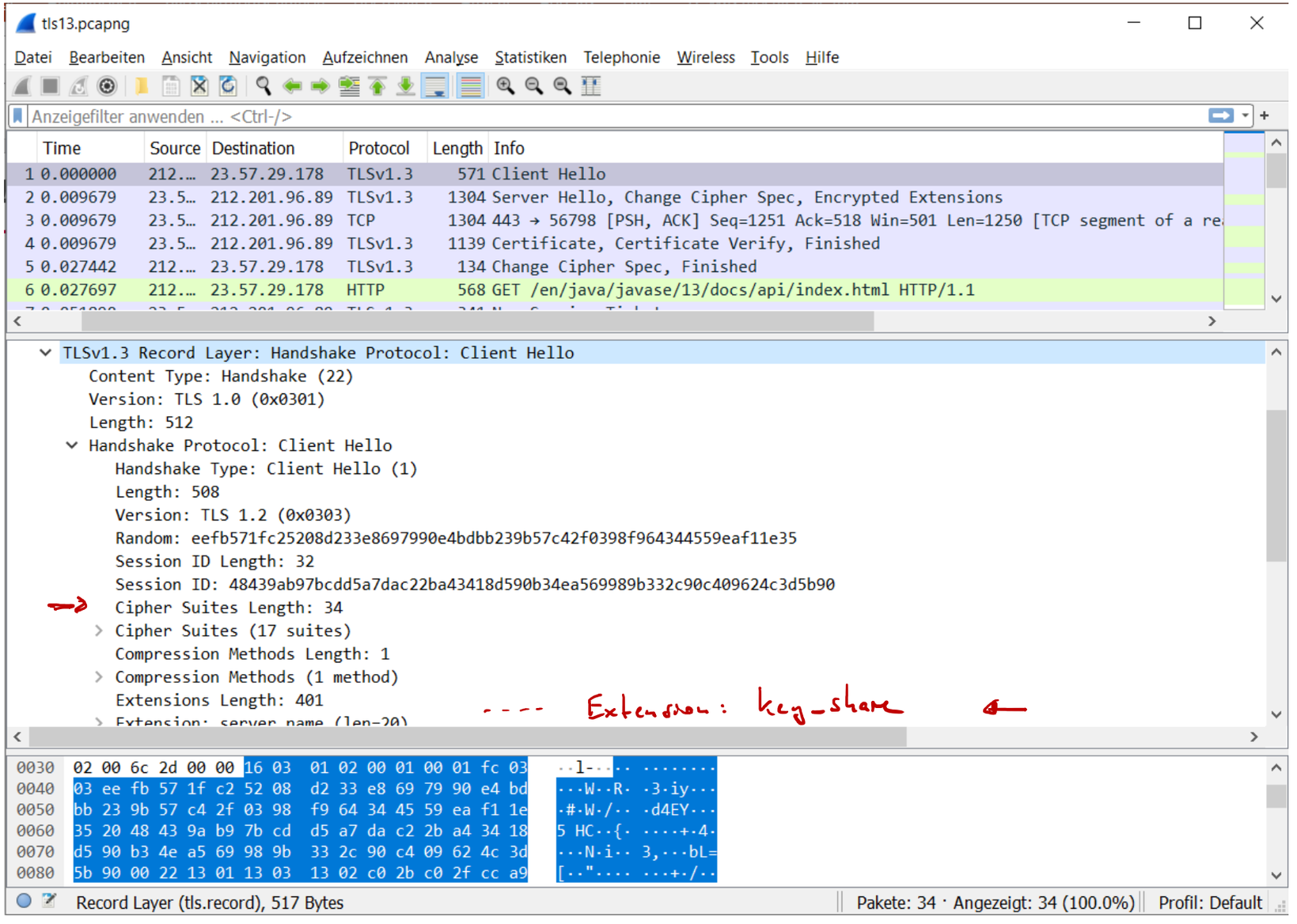
- Use a web browser that allows key logging, see: NSS Key Log Format
- Enable key logging by setting the environment variable SSLKEYLOGFILE to point to a file.
- Start Wireshark to capture a TLS session and filter for “tls”
- To decrypt captured TLS packets right click on a TLS packet and choose
  - Protocol Preferences -> Transport Layer Security -> Open Transport Layer Security preferences…
  - Add the name (complete path) of the file with the key logs at: “(Pre)-Master-Secret log filename”

Decrypting a captured TLS session

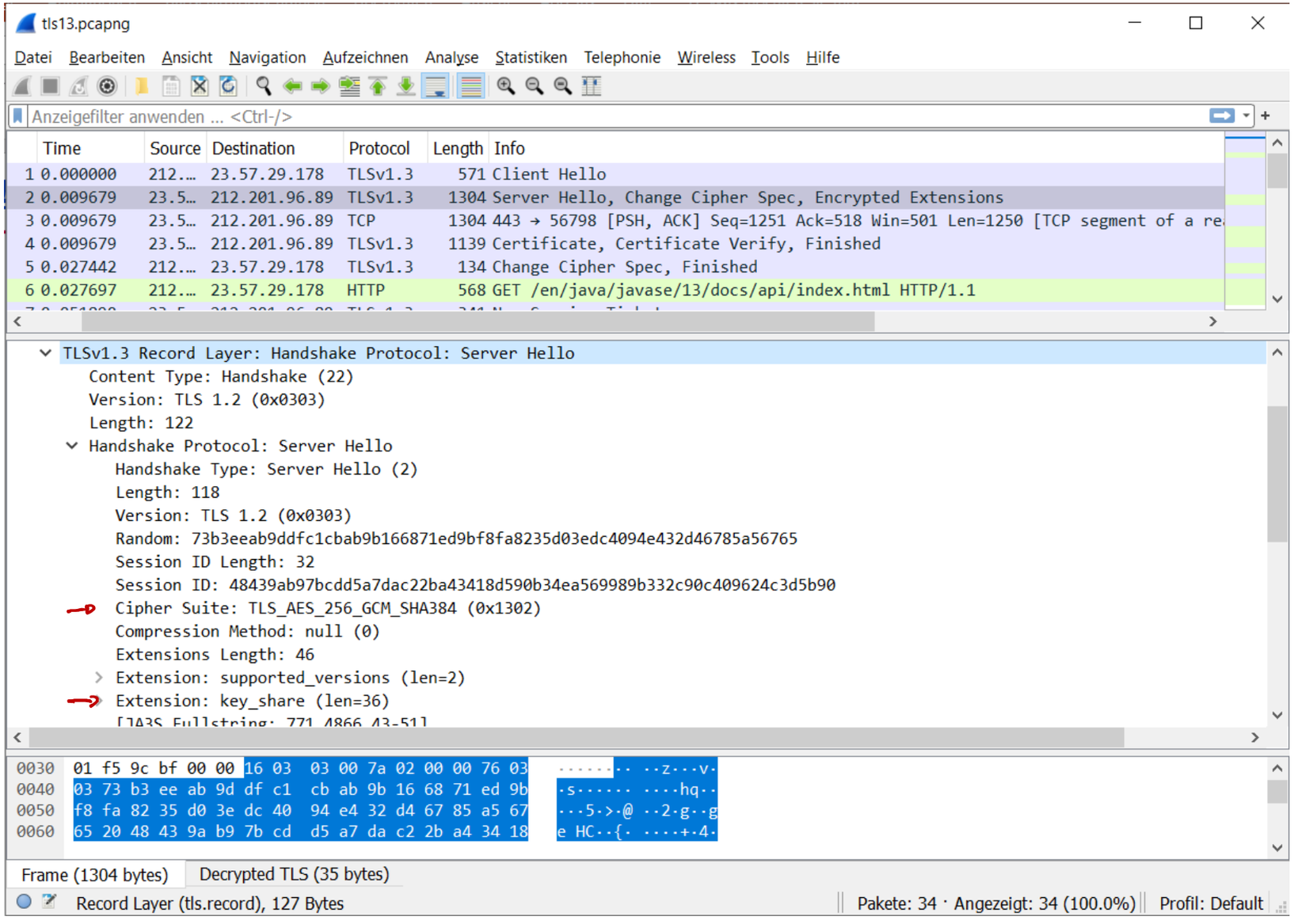




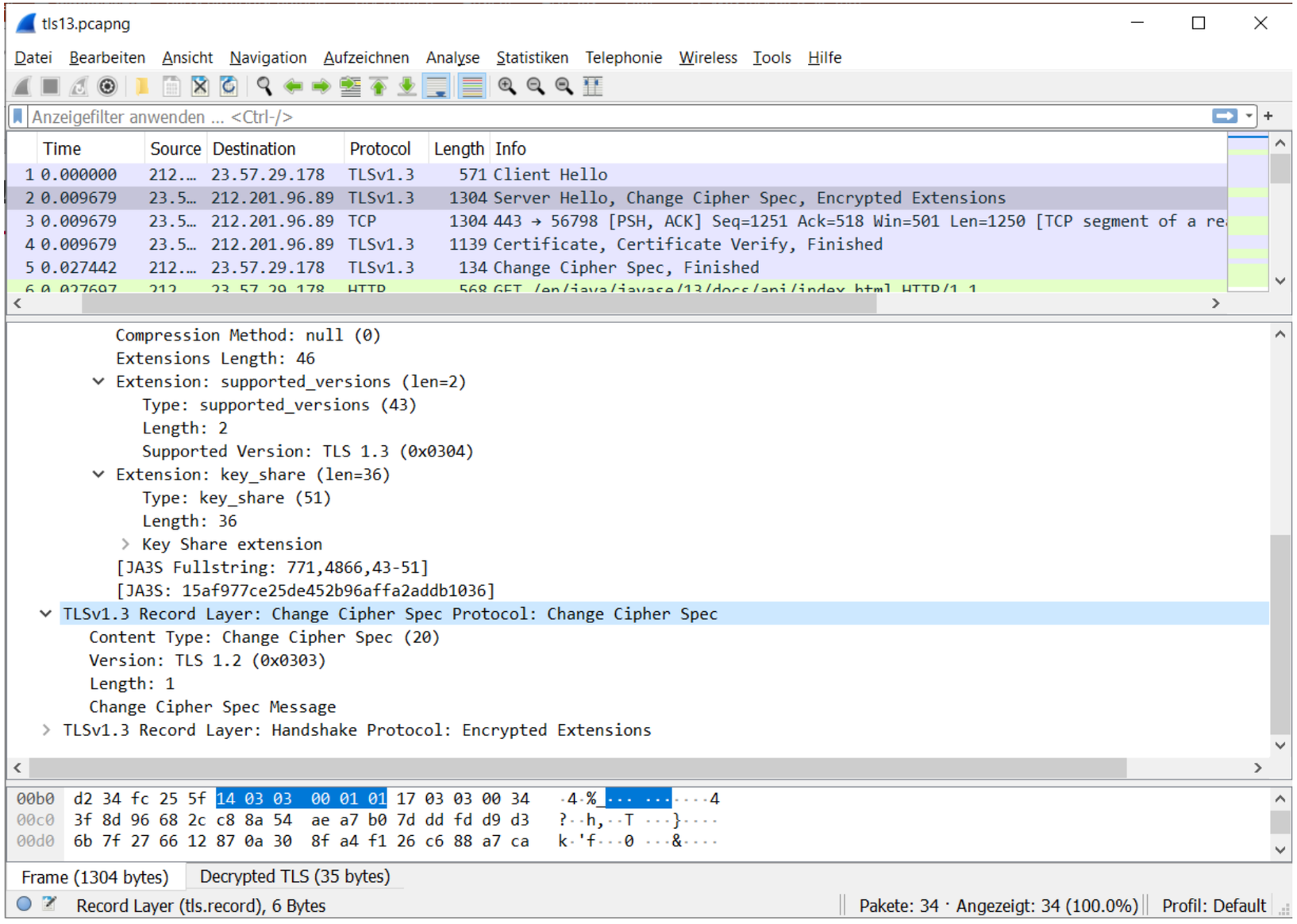
TLS 1.3 Capture – Client Hello



TLS 1.3 Capture – Server Hello

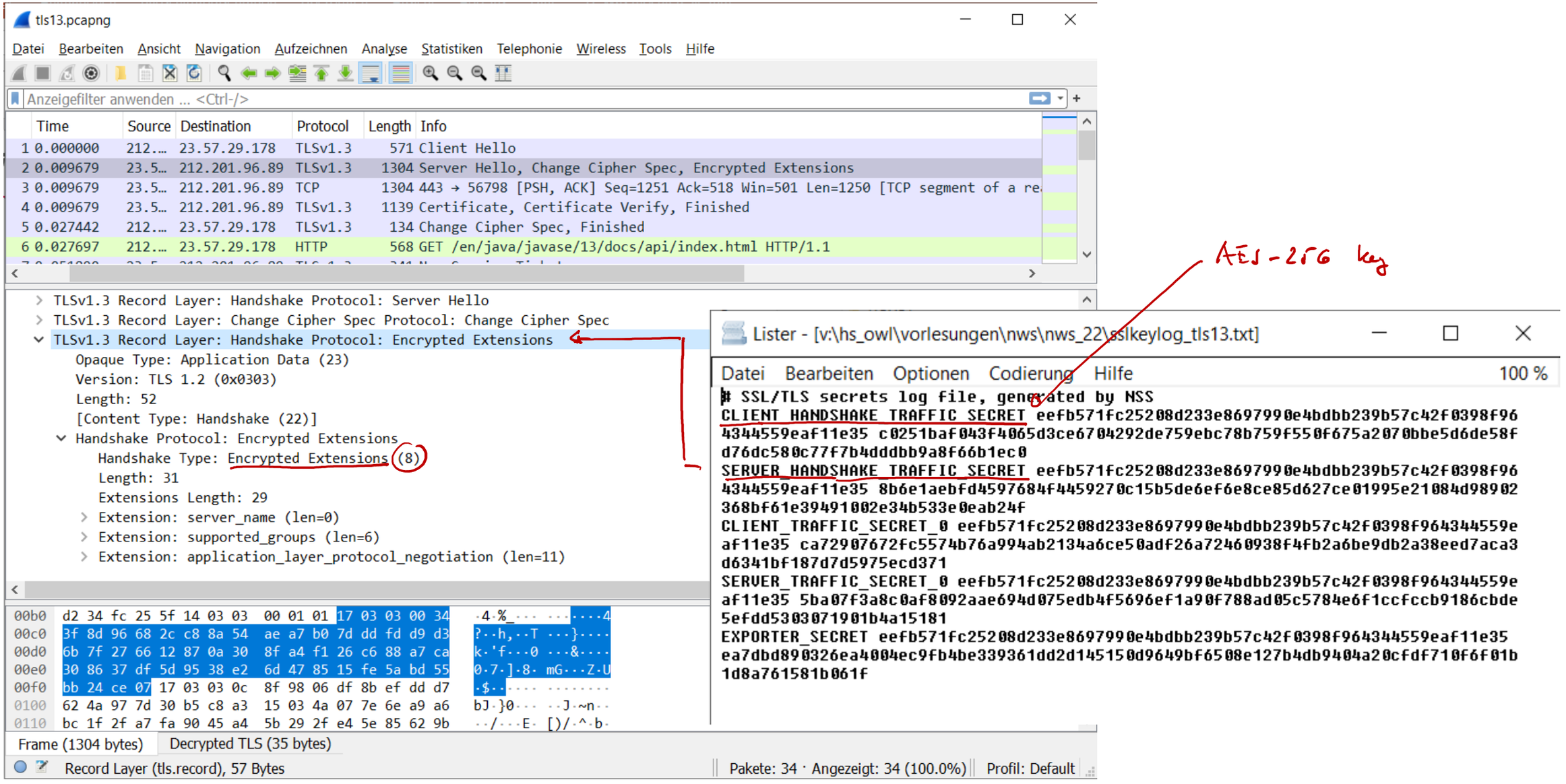


TLS 1.3 Capture – (Server) Change Cipher Spec

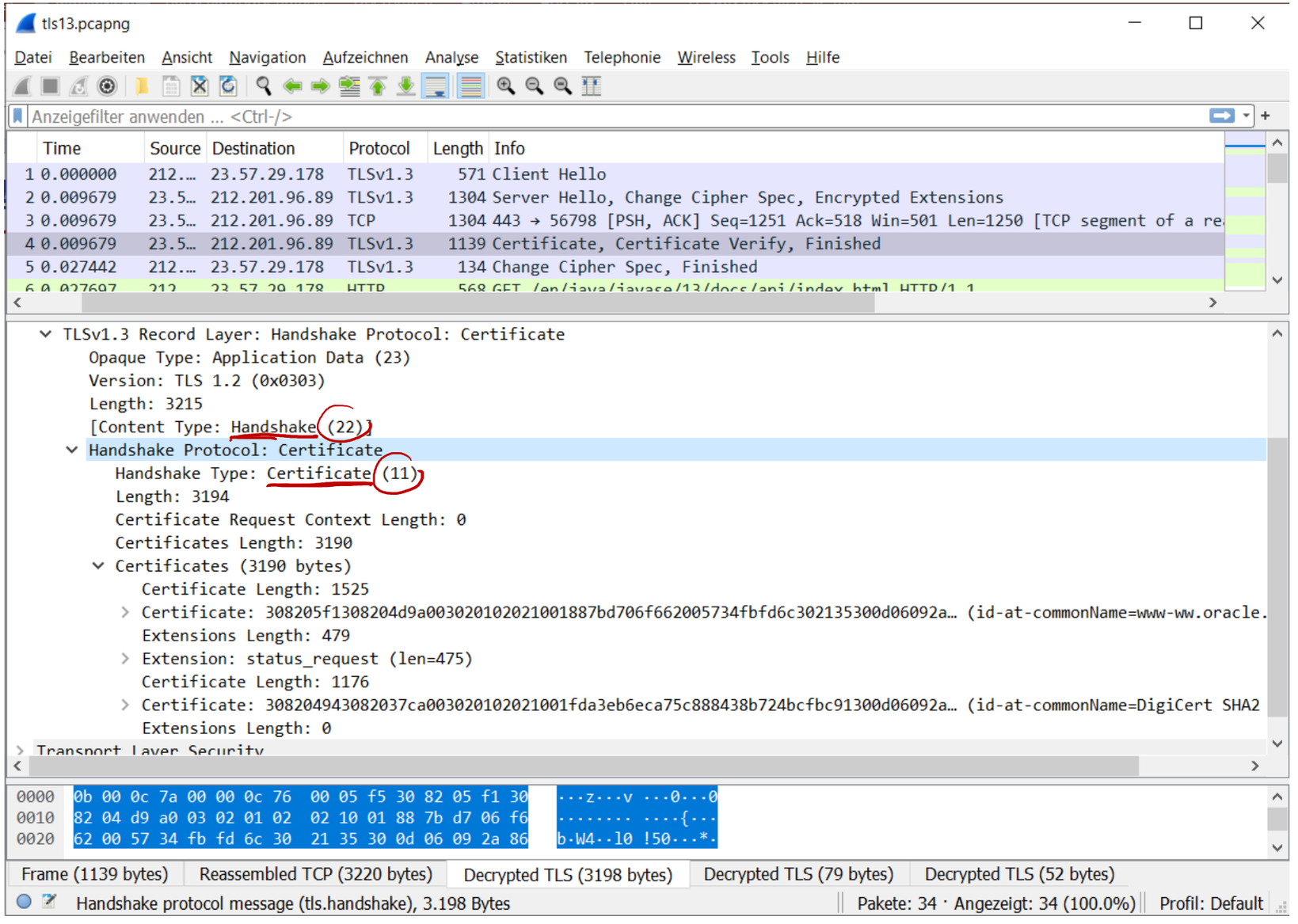


option in TLS v1.3

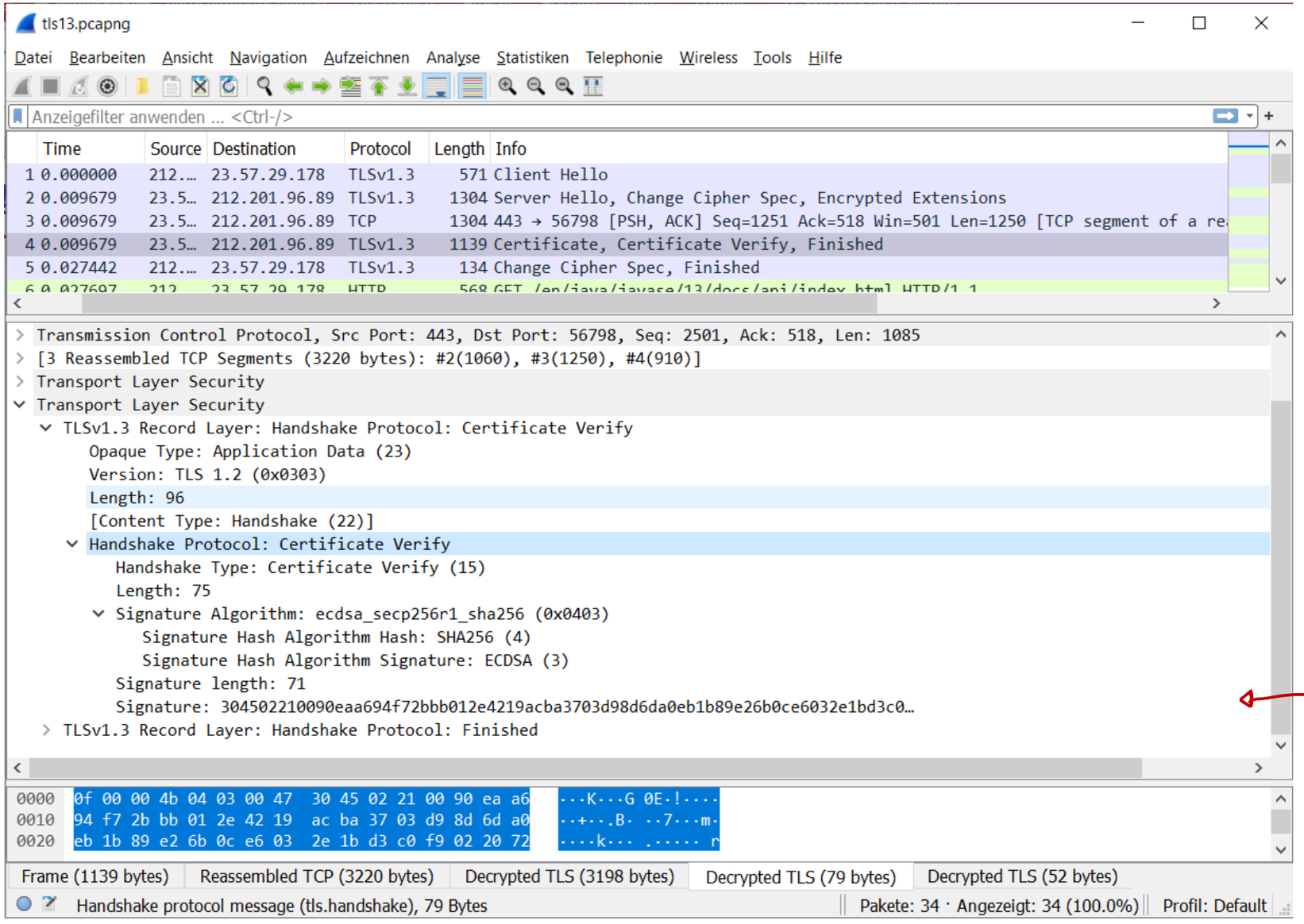
TLS 1.3 Capture – Encrypted Extensions



TLS 1.3 Capture – Certificate

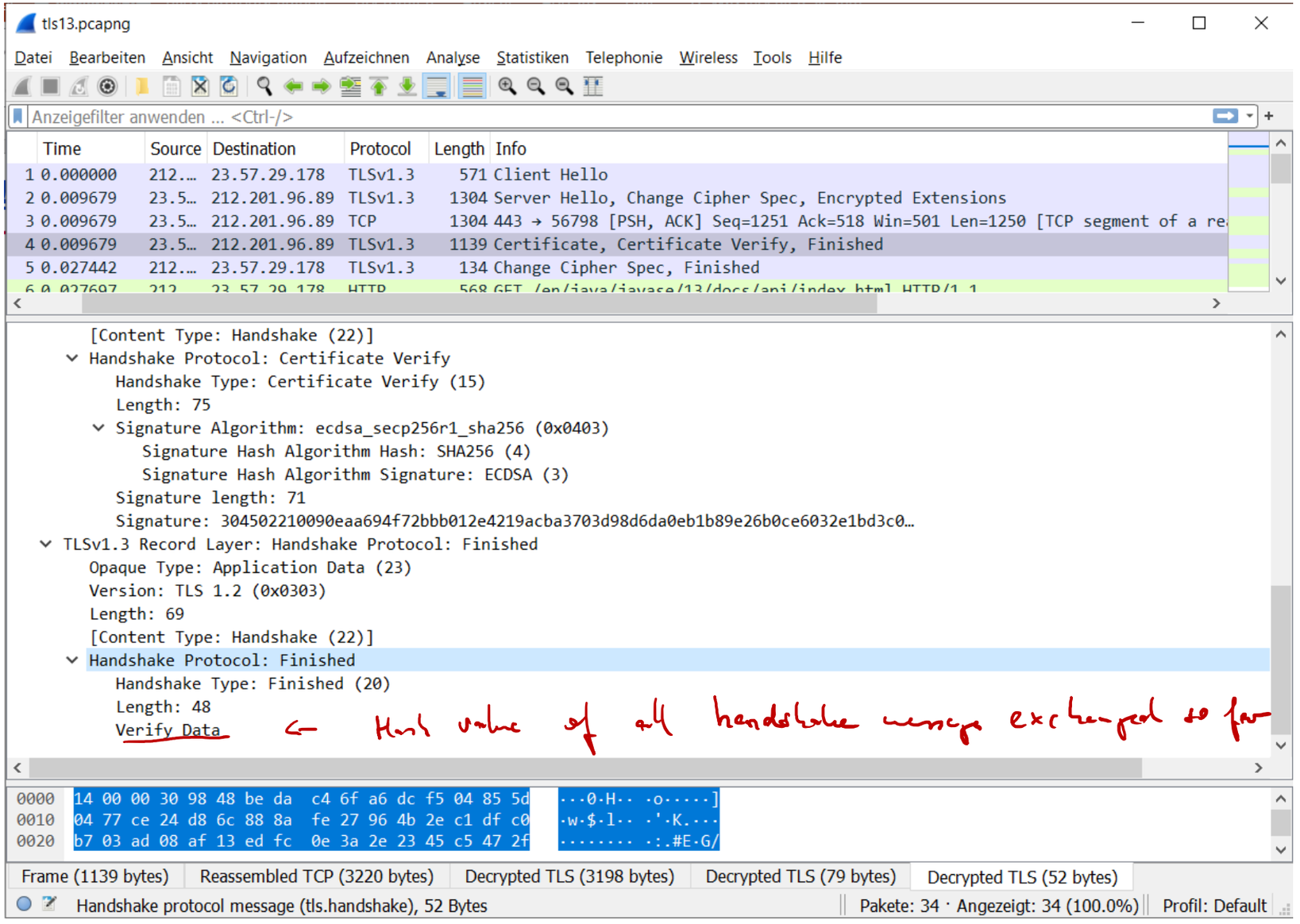


TLS 1.3 Capture – Certificate Verify



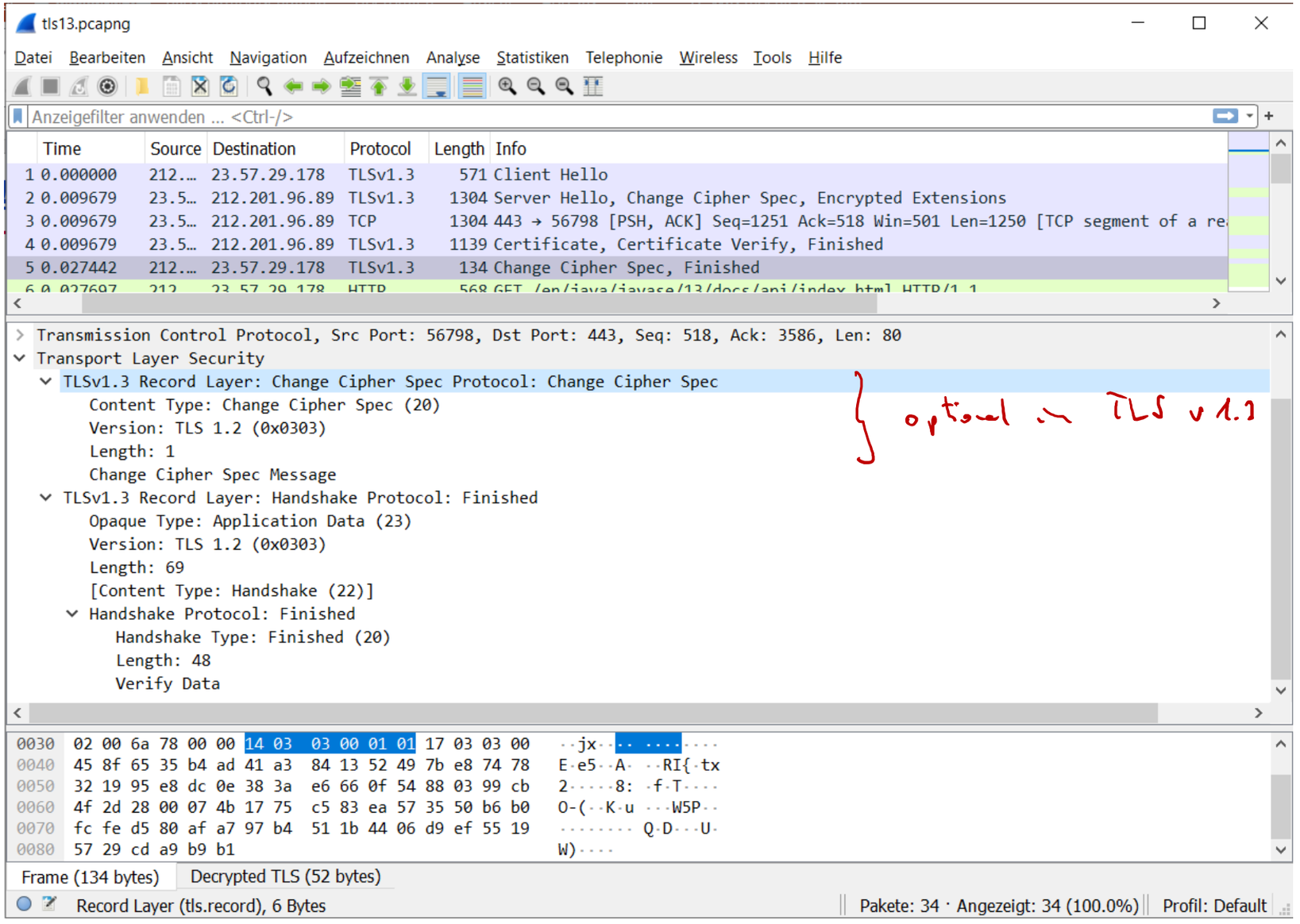
Signature of all Handshake messages exchanged so far, with the server's private key

TLS 1.3 Capture – (Server) Finished

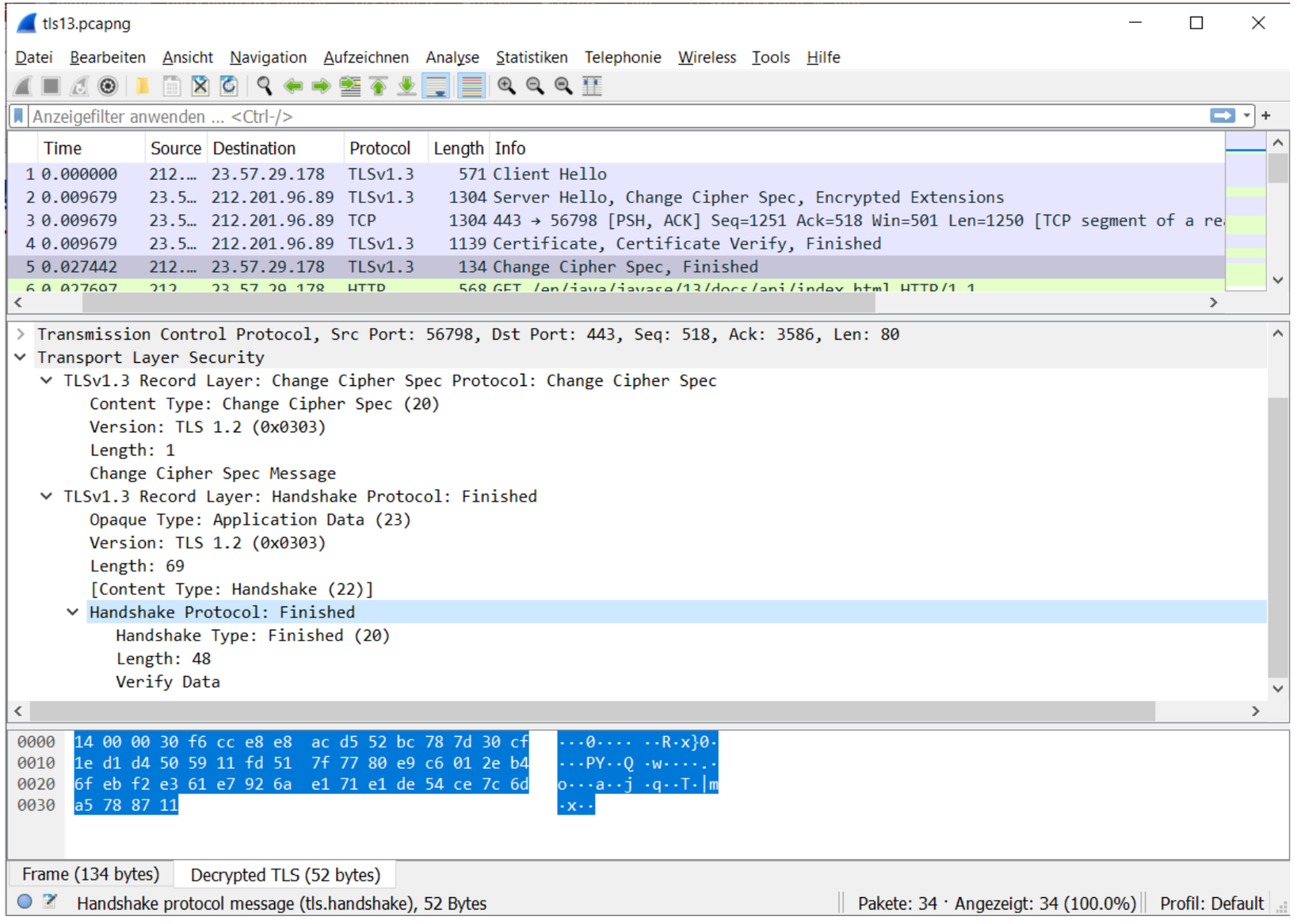


← Hash value of all handshake messages exchanged so far

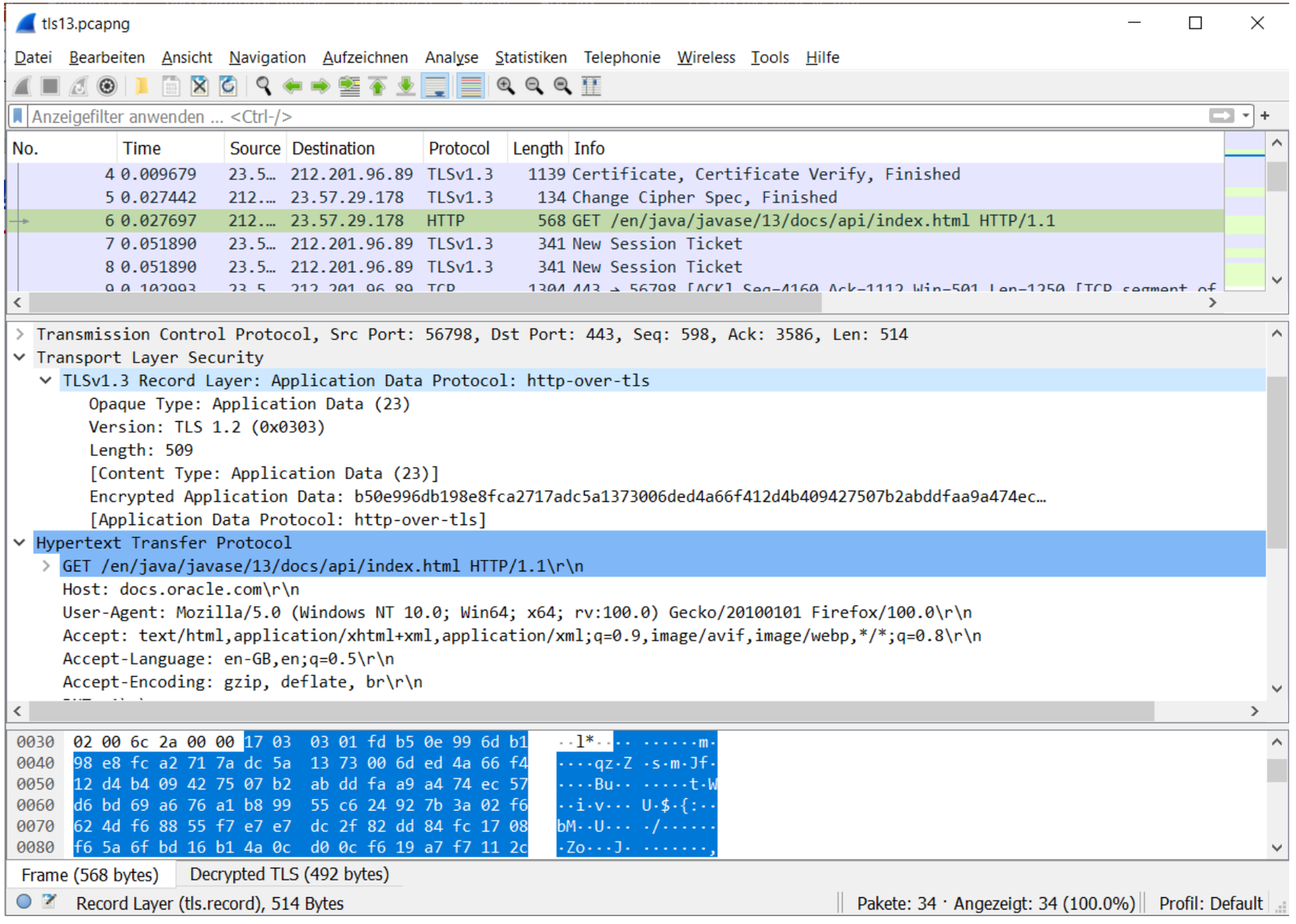
TLS 1.3 Capture – (Client) Change Cipher Spec



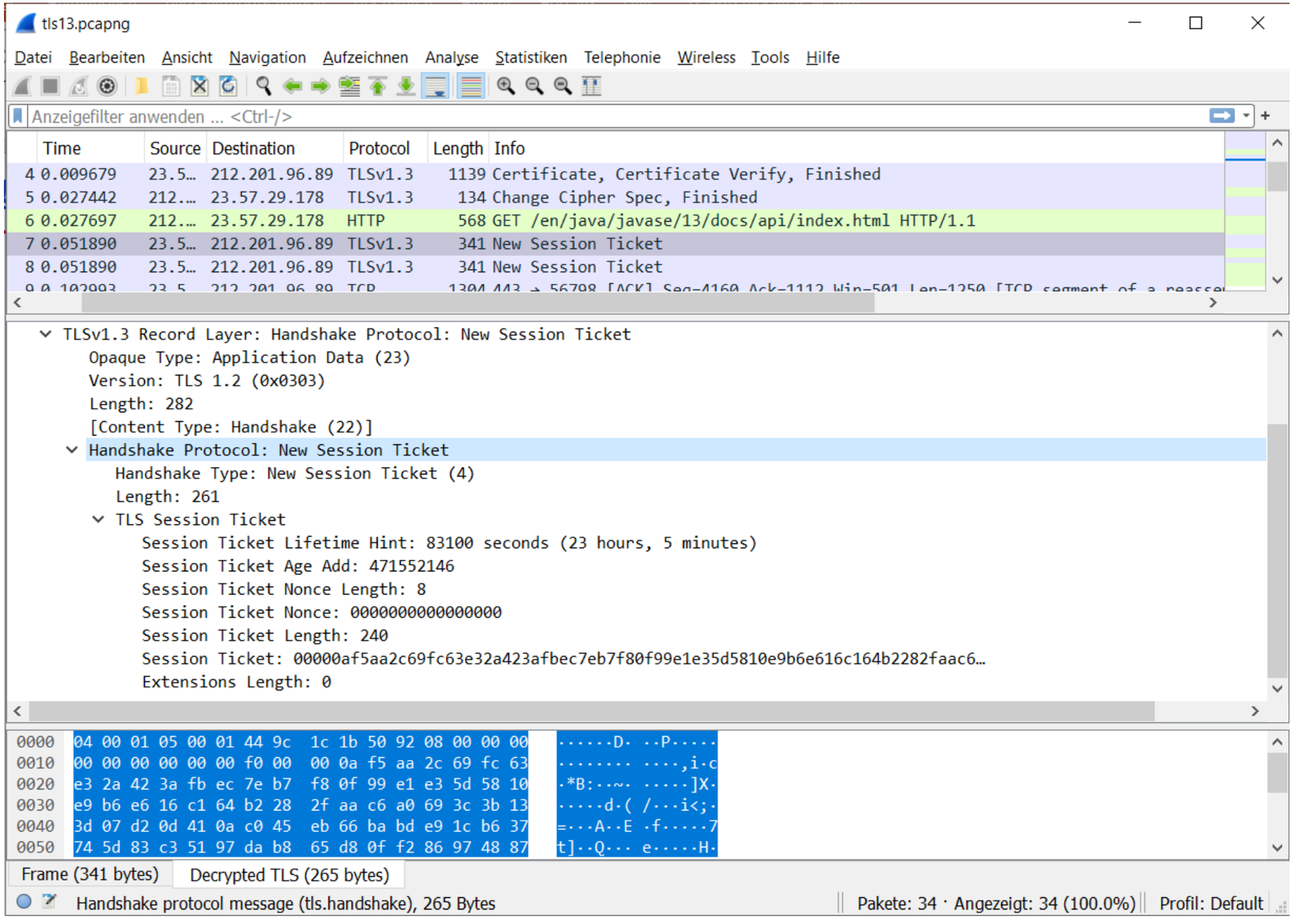
TLS 1.3 Capture – (Client) Finished



TLS 1.3 Capture – Application Data



TLS 1.3 Capture – New Session Ticket



TLS 1.2 – Record Layer (RFC 5246, 6.2)

- **`TLSPlaintext`** records
- **`length`** ≤ 2^{14}

0x14	**`change_cipher_spec`**
0x15	**`alert`**
0x16	**`handshake`**
0x17	**`application_data`**

0x03	0x00	***SSL 3.0***
0x03	0x01	***TLS 1.0***
0x03	0x02	***TLS 1.1***
0x03	0x03	***TLS 1.2***
0x03	0x04	***TLS 1.3***

TLS 1.3 – Record Layer (RFC 8446, 5.1)

- **`TLSPlaintext`** records


```
enum {
    invalid(0),
    change_cipher_spec(20),
    alert(21),
    handshake(22),
    application_data(23),
    (255)
} ContentType;
```

0x14	**`change_cipher_spec`**
0x15	**`alert`**
0x16	**`handshake`**
0x17	**`application_data`**

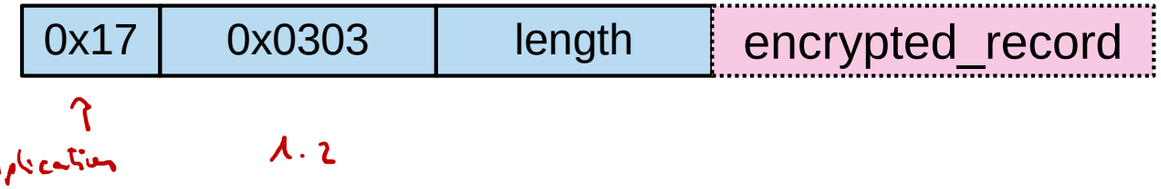
```
struct {
    ContentType type;
    ProtocolVersion legacy_record_version;
    uint16 length;
    opaque fragment[TLSPlaintext.length];
} TLSPlaintext;
```

0x03	0x03	***TLS 1.2***
0x03	0x01	***TLS 1.0***

- TLS 1.3: **`legacy_record_version`**: 0x0303 / 0x0301 (initial ClientHello)

TLS 1.3 – Record Payload Protection (RFC 8446, 5.2)

- **`TLSCiphertext`** records



- In TLS 1.3, as opposed to previous versions of TLS, all ciphers are modeled as "Authenticated Encryption with Associated Data" (AEAD) [RFC5116].

```
struct {
    opaque content[TLSPlaintext.length];
    ContentType type;
    uint8 zeros[length_of_padding];
} TLSInnerPlaintext;

struct {
    ContentType opaque_type = application_data; /* 23 */
    ProtocolVersion legacy_record_version = 0x0303; /* TLS v1.2 */
    uint16 length;
    opaque encrypted_record[TLSCiphertext.length];
} TLSCiphertext;
```

TLS 1.2 – Handshake Protocol (RFC 5246, 7.4)

- **HandshakeType**

0x00	**hello_request**
0x01	**client_hello**
0x02	**server_hello**
0x0B	**certificate** ←
0x0C	**server_key_exchange** 1.2 only
0x0D	**certificate_request**
0x0E	**server_hello_done** 1.2 only
0x0F	**certificate_verify** ←
0x10	**client_key_exchange** 1.2 only
0x14	**finished**

TLS 1.3 – Handshake Protocol (RFC 8446, 4)

- **HandshakeType**

0x01	client_hello
0x02	server_hello
0x0B	certificate
0x0D	certificate_request
0x0F	certificate_verify
0x14	finished

TLS 1.3 – Handshake Protocol (RFC 8446, 4)

- **HandshakeType**

0x01	client_hello
0x02	server_hello
0x04	new_session_ticket
0x05	end_of_early_data
0x08	encrypted_extensions ← !
0x0B	certificate
0x0D	certificate_request
0x0F	certificate_verify
0x14	finished
0x18	key_update
0xFE	message_hash

TLS 1.3 – Handshake Protocol (RFC 8446, 4)

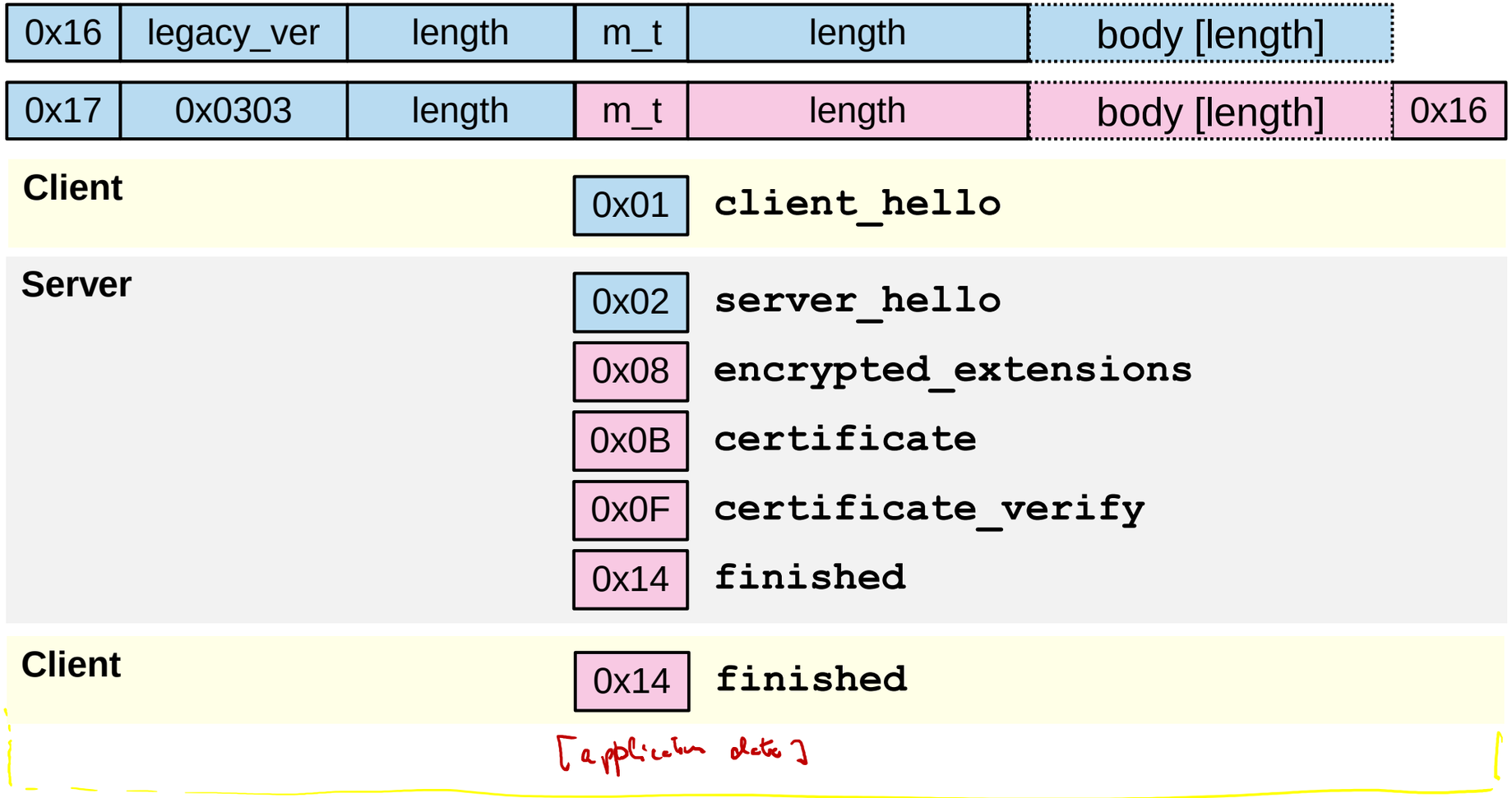
0x16	legacy_ver	length	m_t	length	body [length]

0x17	0x0303	length	m_t	length	body [length]	0x16

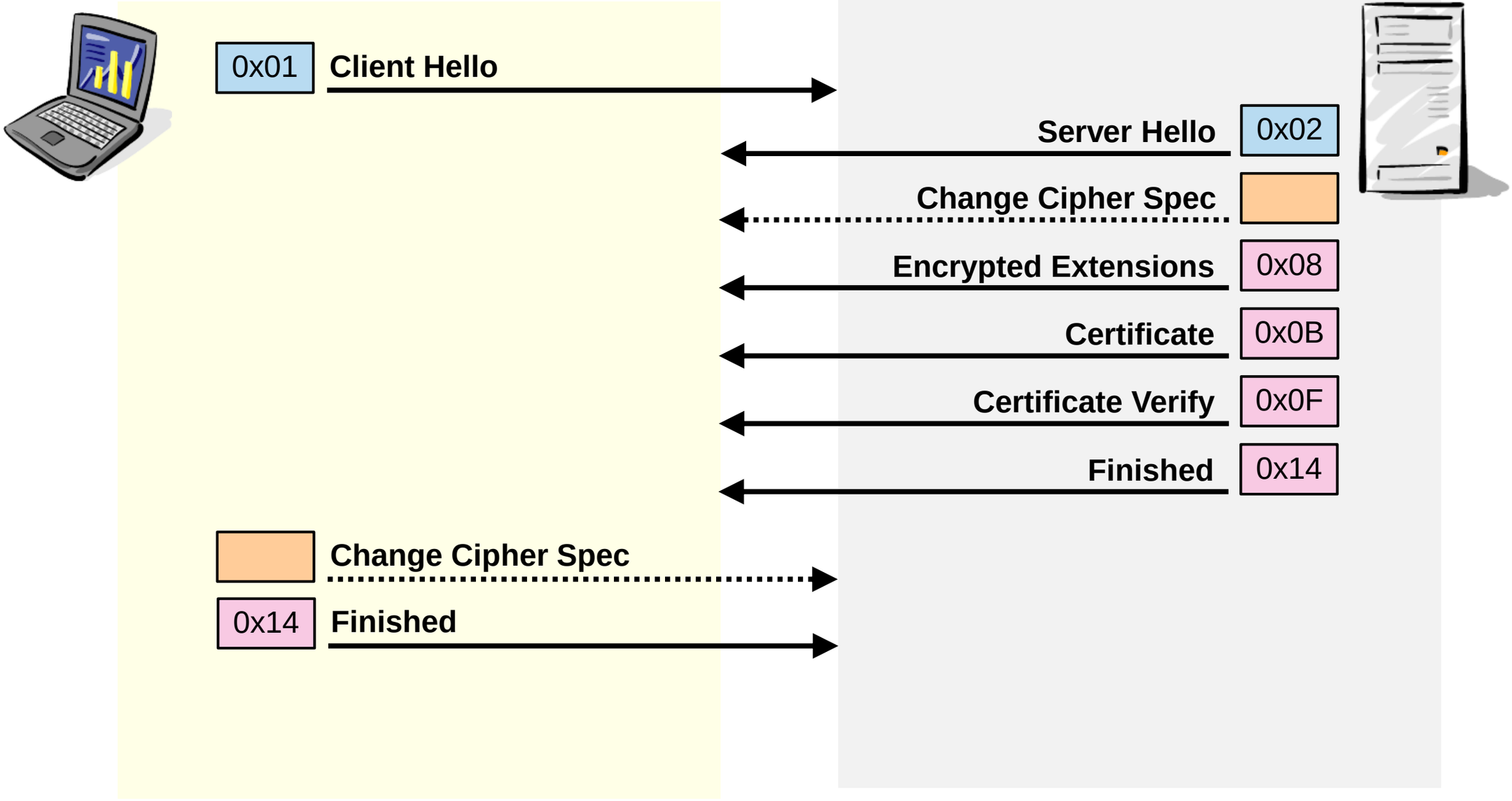
- **HandshakeType**

0x01	client_hello
0x02	server_hello
0x04	new_session_ticket
0x05	end_of_early_data
0x08	encrypted_extensions
0x0B	certificate
0x0D	certificate_request
0x0F	certificate_verify
0x14	finished
0x18	key_update
0xFE	message_hash

TLS 1.3 – Handshake Protocol (RFC 8446, 4)

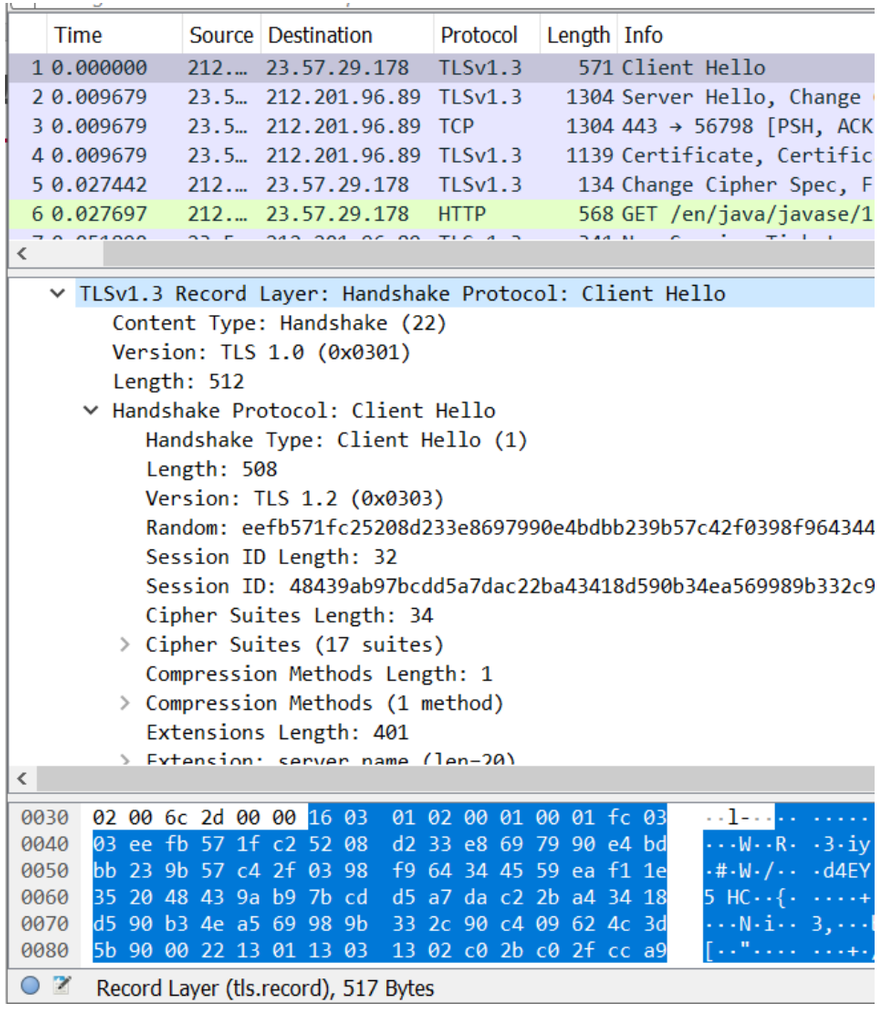


TLS 1.3 – Handshake Protocol



TLS 1.3 – Client Hello (RFC 8446, 4.1.2)

0x16	legacy_ver	length	0x01	length	body [length]



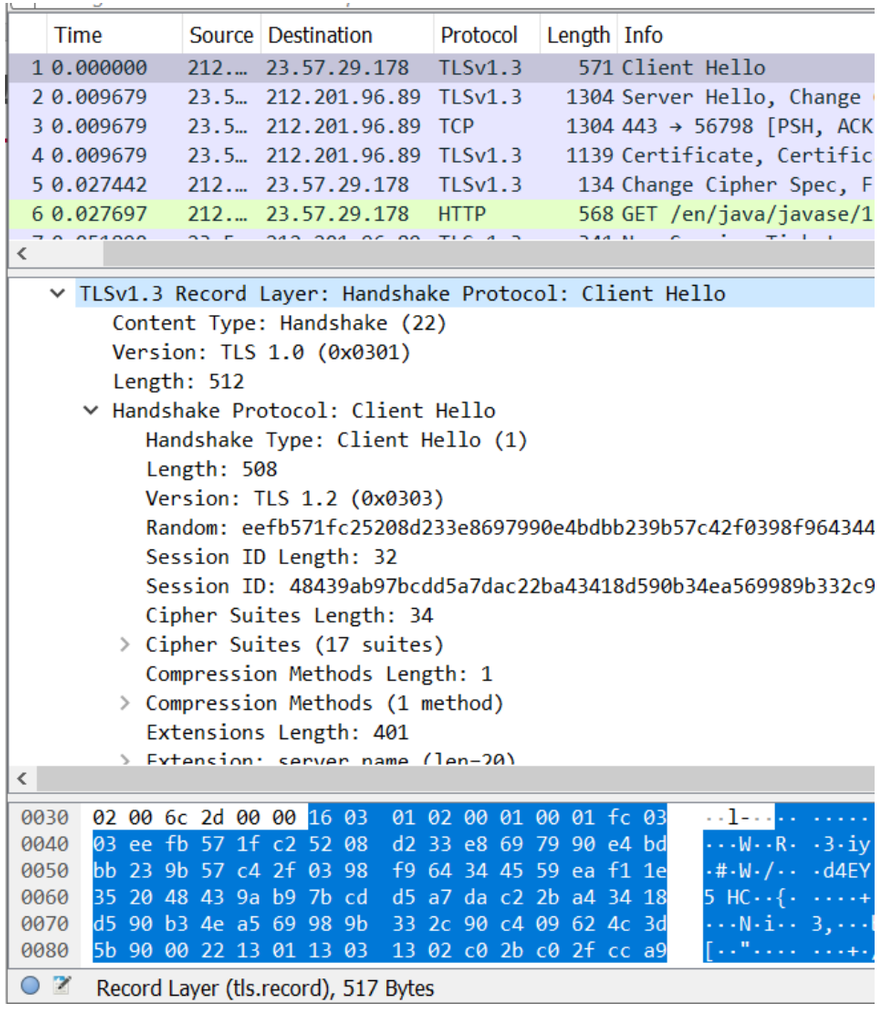

```
uint16 ProtocolVersion;
opaque Random[32];

uint8 CipherSuite[2];     /* Cryptographic suite selector */

struct {
    ProtocolVersion legacy_version = 0x0303;     /* TLS v1.2 */
    Random random;
    opaque legacy_session_id<0..32>;
    CipherSuite cipher_suites<2..2^16-2>;
    opaque legacy_compression_methods<1..2^8-1>;
    Extension extensions<8..2^16-1>;
} ClientHello;
```

TLS 1.3 – Client Hello – Extensions (RFC 8446, 4.2)

0x16	legacy_ver	length	0x01	length	body [length]




```
struct {
    ExtensionType extension_type;
    opaque extension_data<0..2^16-1>;
} Extension;
```

TLS 1.3 – Client Hello – Extensions (RFC 8446, 4.2)

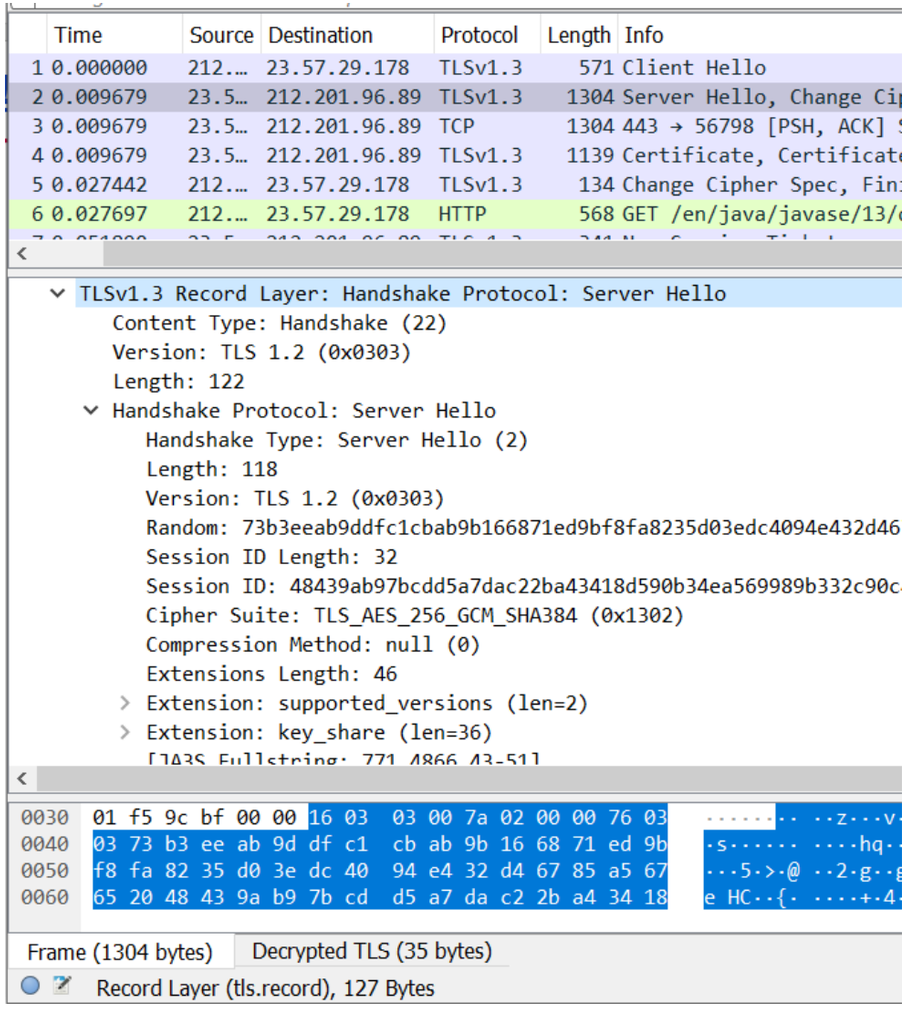
```
enum {
    server_name(0),                             /* RFC 6066 */
    max_fragment_length(1),                     /* RFC 6066 */
    status_request(5),                          /* RFC 6066 */
    supported_groups(10),                       /* RFC 8422, 7919 */
    signature_algorithms(13),                   /* RFC 8446 */
    use_srtp(14),                               /* RFC 5764 */
    heartbeat(15),                              /* RFC 6520 */
    application_layer_protocol_negotiation(16), /* RFC 7301 */
    signed_certificate_timestamp(18),           /* RFC 6962 */
    client_certificate_type(19),                /* RFC 7250 */
    server_certificate_type(20),                /* RFC 7250 */
    padding(21),                                /* RFC 7685 */
    pre_shared_key(41),                         /* RFC 8446 */
    early_data(42),                             /* RFC 8446 */
    supported_versions(43),                     /* RFC 8446 */
    cookie(44),                                 /* RFC 8446 */
    psk_key_exchange_modes(45),                 /* RFC 8446 */
    certificate_authorities(47),                /* RFC 8446 */
    oid_filters(48),                            /* RFC 8446 */
    post_handshake_auth(49),                    /* RFC 8446 */
    signature_algorithms_cert(50),              /* RFC 8446 */
    key_share(51),                              /* RFC 8446 */
    (65535)
} ExtensionType;
```

TLS 1.3 – Client Hello – Extensions (RFC 8446, 4.2)

```
enum {
    server_name(0),                             /* RFC 6066 */
    max_fragment_length(1),                     /* RFC 6066 */
    status_request(5),                          /* RFC 6066 */
    supported_groups(10),                       /* RFC 8422, 7919 */
    signature_algorithms(13),                   /* RFC 8446 */
    use_srtp(14),                               /* RFC 5764 */
    heartbeat(15),                              /* RFC 6520 */
    application_layer_protocol_negotiation(16), /* RFC 7301 */
    signed_certificate_timestamp(18),           /* RFC 6962 */
    client_certificate_type(19),                /* RFC 7250 */
    server_certificate_type(20),                /* RFC 7250 */
    padding(21),                                /* RFC 7685 */
    pre_shared_key(41),                         /* RFC 8446 */
    early_data(42),                             /* RFC 8446 */
    supported_versions(43),                     /* RFC 8446 */
    cookie(44),                                 /* RFC 8446 */
    psk_key_exchange_modes(45),                 /* RFC 8446 */
    certificate_authorities(47),                /* RFC 8446 */
    oid_filters(48),                            /* RFC 8446 */
    post_handshake_auth(49),                    /* RFC 8446 */
    signature_algorithms_cert(50),              /* RFC 8446 */
    key_share(51),                              /* RFC 8446 */
    (65535)
} ExtensionType;
```

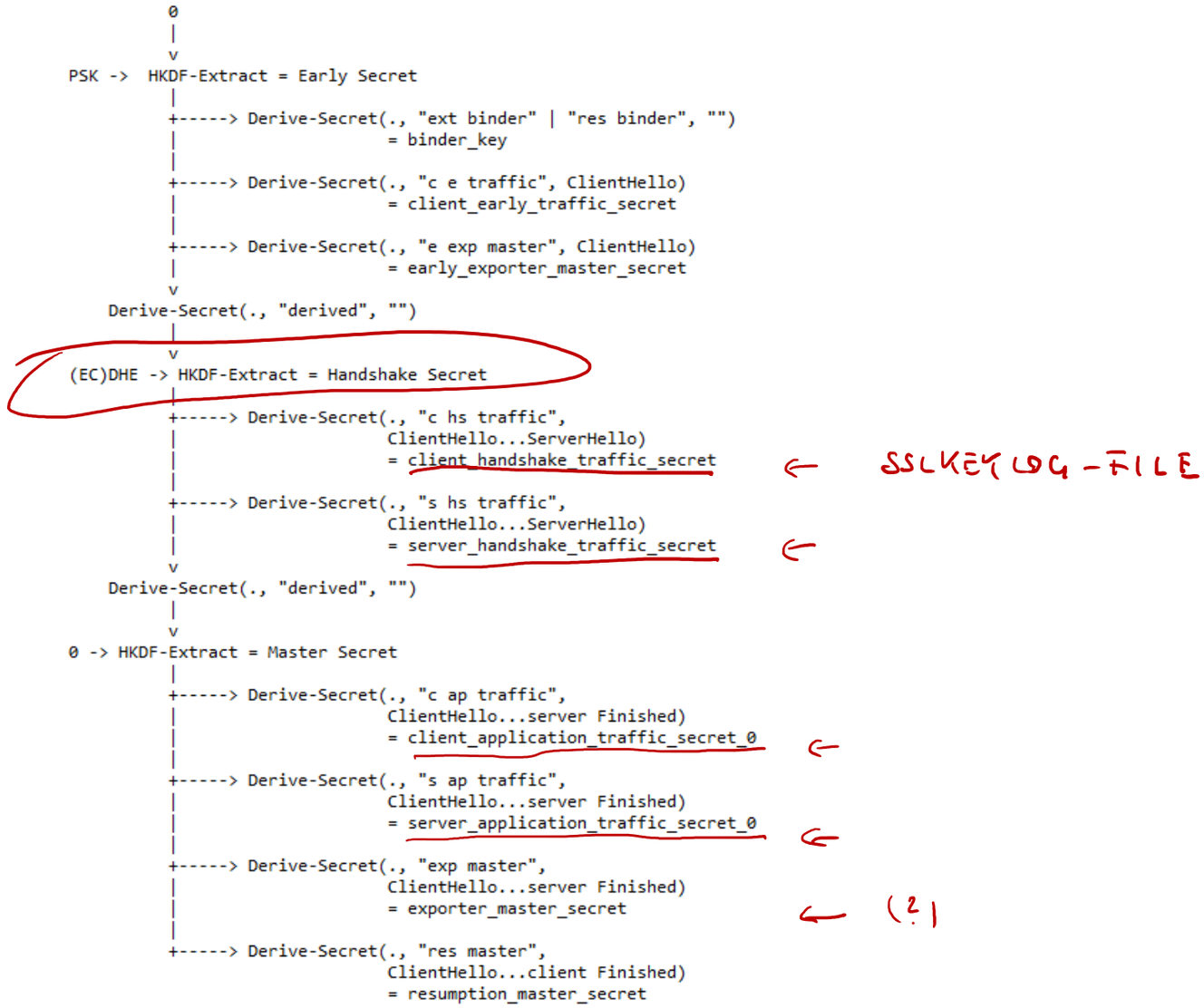
TLS 1.3 – Server Hello (RFC 8446, 4.1.3)

0x16	legacy_ver	length	0x02	length	body [length]




```
struct {
    ProtocolVersion legacy_version = 0x0303;    /* TLS v1.2 */
    Random random;
    opaque legacy_session_id_echo<0..32>;
    CipherSuite cipher_suite;
    opaque legacy_compression_method = 0;
    Extension extensions<8..2^16-1>;
} ServerHello;
```

TLS 1.3 – Key Schedule (RFC 8446, 7.1)



SSLKEYLOG-FILE

(?)

TLS 1.3 – Key Schedule (RFC 8446, 7.1)

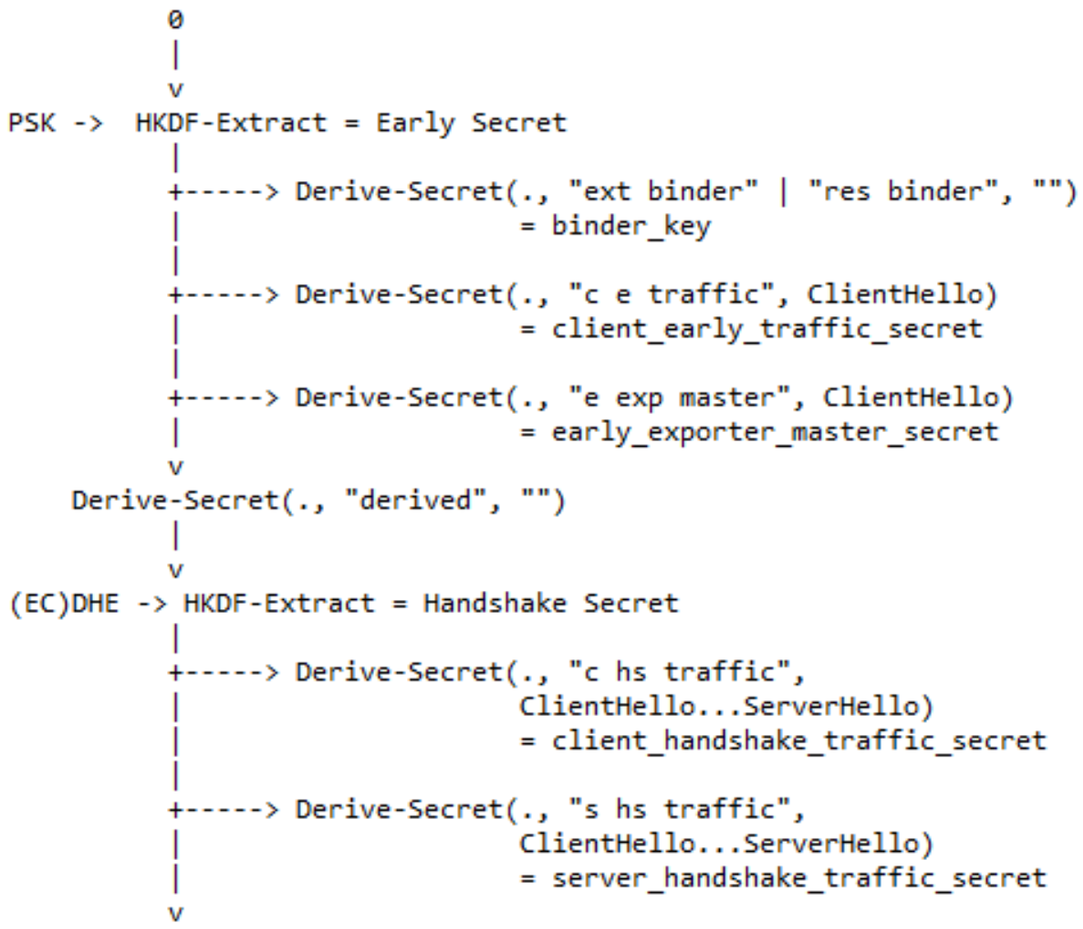
The key derivation process makes use of the **HKDF-Extract** and **HKDF-Expand** functions as defined for **HKDF [RFC5869]**, as well as the functions defined below:

```
HKDF-Expand-Label(Secret, Label, Context, Length) =
    HKDF-Expand(Secret, HkdfLabel, Length)

struct {
    uint16 length = Length;
    opaque label<7..255> = "tls13 " + Label;
    opaque context<0..255> = Context;
} HkdfLabel;

Derive-Secret(Secret, Label, Messages) =
    HKDF-Expand-Label(Secret, Label,
                      Transcript-Hash(Messages), Hash.length)
```

TLS 1.3 – Key Schedule – Traffic Key Calculation (RFC 8446, 7.3)

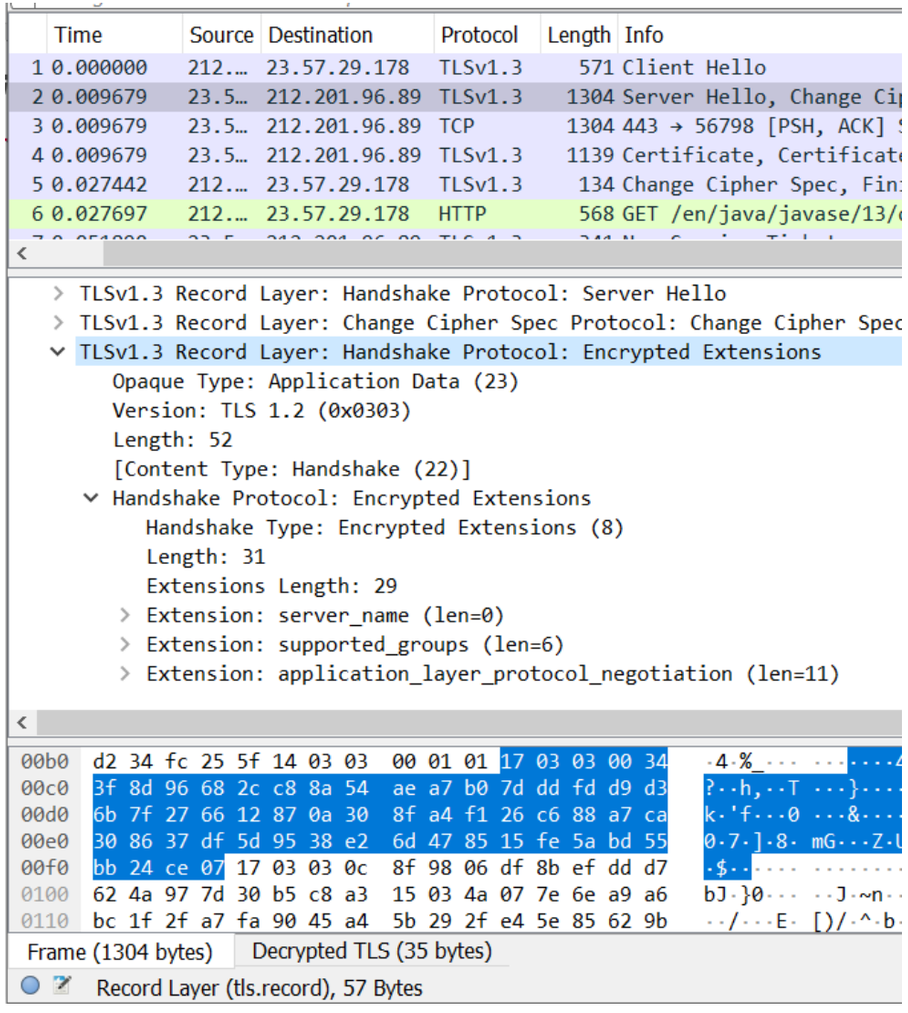



```
[sender]_write_key = HKDF-Expand-Label(Secret, "key", "", key_length)

[sender]_write_iv  = HKDF-Expand-Label(Secret, "iv", "", iv_length)
```

TLS 1.3 – Encrypted Extensions (RFC 8446, 4.3.1)

0x17	legacy_ver	length	0x08	length	body [length]	0x16




```
struct {
    Extension extensions<0..2^16-1>;
} EncryptedExtensions;
```