MTAT.07.017
Applied Cryptography

Transport Layer Security (TLS)

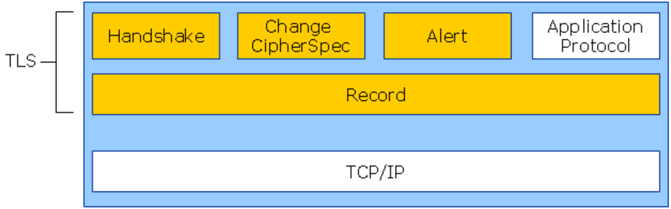
University of Tartu

Spring 2018

Transport Layer Security

"TLS is cryptographic protocol that provides communication security over the Internet."

- Provides confidentiality, integrity and server authentication
- Most successful and widely used cryptographic protocol (!!!)
- Any application protocol can be encapsulated in TLS



TLS History

- SSL 1.0 – never publicly released
- SSL 2.0 – Netscape (1995)
- SSL 3.0 – Netscape (1996)
- TLS 1.0 (SSL 3.1) – RFC 2246 (1999)
- TLS 1.1 – RFC 4346 (2006)
- **TLS 1.2** – RFC 5246 (2008)
- TLS 1.3 – RFC draft (2016)

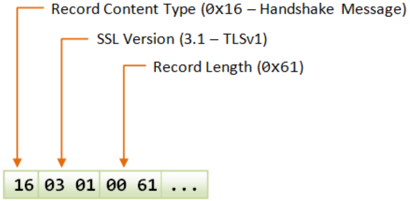
No fundamental changes between versions

`http://www.ietf.org/rfc/rfc5246.txt`

TLS Record Layer

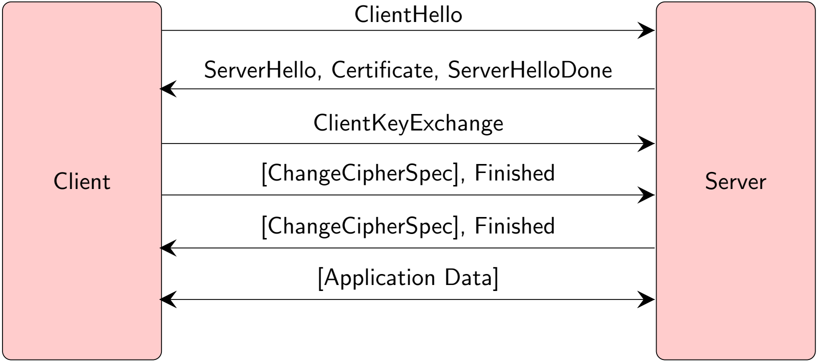
[Type] [Version] [Length] [Data]

- Type: content type of encapsulated data:
  - Handshake message (0x16)
  - Change cipher spec message (0x14)
  - Alert message (0x15)
  - Application data (0x17)
- Protocol version: 0x0303 (for TLS v1.2)
- Length: length of the data (2 bytes)
- Data: encapsulated content
  - Can contain several same type messages



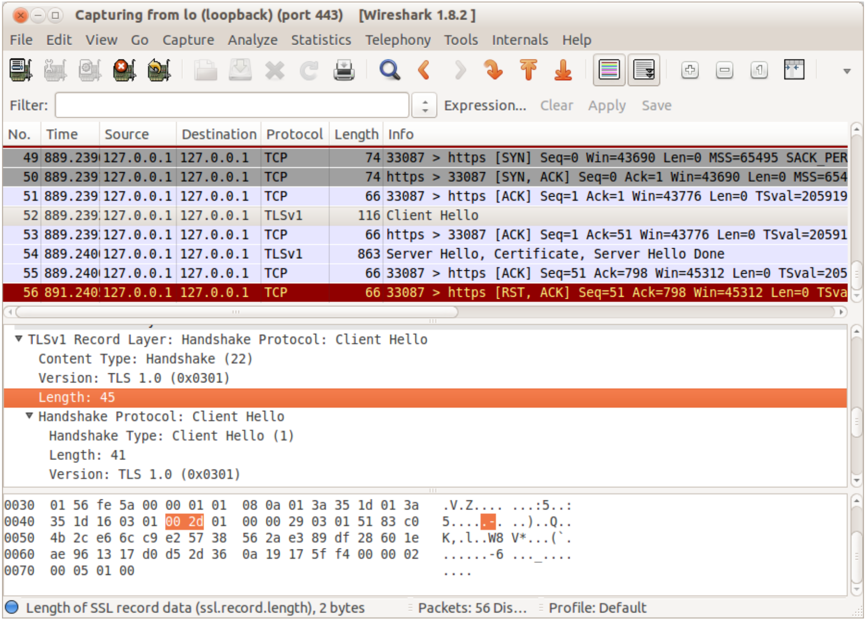
Header is never encrypted!

TLS Handshake



- Client verifies server's X.509 certificate
- Client extracts from the certificate server's public key
- Client encrypts random symmetric key using public key
- Only the server can decrypt symmetric key
- Now the client and server share the same symmetric key
- Symmetric key used for actual data encryption/authentication

Dissecting TLS with Wireshark



Alert Message

Signals about TLS related issues to other party

`[Level] [Description]`

- Level (1 byte):
  - Warning (0x01)
  - Fatal (0x02)
- Description (1 byte):

```
close_notify(0),
unexpected_message(10),
bad_record_mac(20),
decryption_failed(21),
handshake_failure(40),
bad_certificate(42),
unsupported_certificate(43),
certificate_revoked(44),
certificate_expired(45),
illegal_parameter(47),
unknown_ca(48),
access_denied(49),
decrypt_error(51),
user_canceled(90),
...
```

```
▼ TLSv1 Record Layer: Alert (Level: Fatal, Description: Certificate Unknown)
    Content Type: Alert (21)
    Version: TLS 1.0 (0x0301)
    Length: 2
  ▼ Alert Message
      Level: Fatal (2)
      Description: Certificate Unknown (46)
```

Change Cipher Spec Message

```
▼ 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
```

Signals to other party that from now on the negotiated cipher suite will be used to protect outgoing messages
[0x01]

Application Data

```
▼ TLSv1.1 Record Layer: Application Data Protocol: http
    Content Type: Application Data (23)
    Version: TLS 1.1 (0x0302)
    Length: 448
    Encrypted Application Data: 3a37312ac35ea3809f392b2b76174849218d83f179d6d305...
```

Contains (most likely encrypted) application data in a form as required by the application protocol (e.g., HTTP request/response etc.)

`[Application Data]`

Handshake Message

Negotiates TLS protocol security parameters
`[Type] [Length] [Body]`

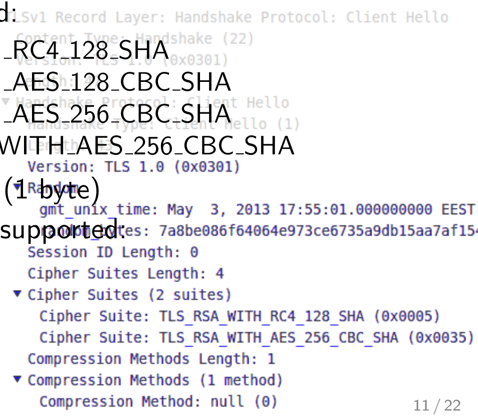
- Type: message type:

```
hello_request(0), client_hello(1), server_hello(2),
certificate(11), server_key_exchange (12),
certificate_request(13), server_hello_done(14),
certificate_verify(15), client_key_exchange(16),
finished(20)
```

- Length: length of the body (3 bytes)
- Body: message body
  - Can be split over several records

Handshake Message: `client_hello`

- Highest TLS version supported (2 bytes)
- Client randomness (32 bytes)
  - Timestamp in first 4 bytes
- Session ID length (1 byte) + session ID
- Cipher suites length (2 bytes)
- List of cipher suites supported:
  - `0x0005` – TLS_RSA_WITH_RC4_128_SHA
  - `0x002f` – TLS_RSA_WITH_AES_128_CBC_SHA
  - `0x0035` – TLS_RSA_WITH_AES_256_CBC_SHA
  - `0x0039` – TLS_DHE_RSA_WITH_AES_256_CBC_SHA
- Compression methods length (1 byte)
- List of compression methods supported:
  - `0x00` – null (mandatory)
  - `0x01` – DEFLATE (gzip)
- Extensions (optional)



Handshake Message: `server_hello`

```
▼ TLSv1 Record Layer: Handshake Protocol: Server Hello
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 74
  ▼ Handshake Protocol: Server Hello
      Handshake Type: Server Hello (2)
      Length: 70
      Version: TLS 1.0 (0x0301)
    ▼ Random
        gmt_unix_time: May  3, 2013 17:55:01.000000000 EEST
        random_bytes: 6dbfaec346d39439ac083b8c0df9f485b327c3
      Session ID Length: 32
      Session ID: 6a9137403d3be6f28e95ca0b0053734f6edad2813
      Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)
      Compression Method: null (0)
```

- TLS version selected (2 bytes)
- Server randomness (32 bytes)
    - Timestamp in first 4 bytes
- Session ID length (1 byte) + session ID
- Cipher suite selected (2 bytes)
- Compression method selected (1 byte)
- Extensions (optional)

Handshake Message: `certificate`

```
▼ TLSv1 Record Layer: Handshake Protocol: Certificate
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 2951
  ▼ Handshake Protocol: Certificate
      Handshake Type: Certificate (11)
      Length: 2947
      Certificates Length: 2944
    ▼ Certificates (2944 bytes)
        Certificate Length: 694
      ▶ Certificate (id-at-commonName=ubuntu)
        Certificate Length: 983
      ▶ Certificate (id-at-commonName=ESTEID-SK 2007,id-
        Certificate Length: 1258
      ▶ Certificate (id-at-commonName=Juur-SK,id-at-orga
```

- Length of certificate list (3 bytes)
- List of certificates
    - Certificate length (3 bytes)
    - DER encoded certificate

- The first is server's certificate
- Other certificates are optional
    - Usually intermediate CA certificates

Handshake Message: `server_hello_done`

```
▼ 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
```

- Empty message body
- Tells that there will be no more messages from the server in this protocol round

Handshake Message: client_key_exchange

```
▼ TLSv1 Record Layer: Handshake Protocol: Client Key Exchange
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 262
  ▼ Handshake Protocol: Client Key Exchange
      Handshake Type: Client Key Exchange (16)
      Length: 258
    ▼ RSA Encrypted PreMaster Secret
        Encrypted PreMaster length: 256
        Encrypted PreMaster: 34f527956711a0e5100b30571910486042
```

Contains (two byte length-prefixed) encrypted 48 byte random "pre-master secret"

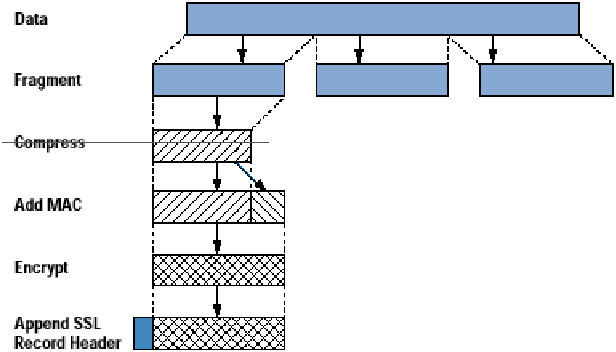
- Encrypted using public key from the server certificate
- Encrypted according to PKCS#1 v1.5
- First two bytes in premaster secret contain TLS version
  - Must be checked by the server
  - Prevents some attacks (?)
- Next 46 bytes are truly random bytes

Handshake Message: `finished`

```
▼ TLSv1.1 Record Layer: Handshake Protocol: Encrypted Handshake Message
    Content Type: Handshake (22)
    Version: TLS 1.1 (0x0302)
    Length: 64
    Handshake Protocol: Encrypted Handshake Message
```

- The first encrypted message
- Serves to verify if encryption works
- Contains hash of concatenation of all previous handshake messages (excluding TLS record header)
    - Must be verified by other party to detect downgrade attacks

Encryption



- Plaintext compression leaks information (CRIME attack)
- How many symmetric keys are needed?
  - MAC & encrypt (+ IV for block ciphers)
    - Separate keys for both directions

How to derive these keys from 48 byte pre-master secret?

Key Derivation

- TLS defines PRF() (pseudo-random function)
  - Uses SHA-256()
  - Produces infinitely long pseudo-random output
- 48 byte "master secret" is derieved:

```
PRF(premaster + "master secret" + client_random + server_random, 48)
```

- From "master secret" is derieved key block in the size needed:

```
PRF(master_secret + "key expansion" + server_random + client_random, 136)
```

- Key block is split into keys needed:

```
client_mac_key = key_block[:20]
server_mac_key = key_block[20:40]
client_enc_key = key_block[40:56]
server_enc_key = key_block[56:72]
client_iv = ...
...
```

MAC Calculation

```
HMAC_digest(key, seq + type + version + length + data)
```

- digest: digest algorithm from negotiated cipher suite
- key: client/server MAC key
- seq: client/server Sequence number (8 bytes)
  - Starts from 0
  - Incremented for every encrypted TLS record
- type: TLS record content type
- version: TLS protocol version (2 bytes)
- length: length of the content (2 bytes)
- data: content

Type, version and length are fields from TLS record header!

This way integrity for TLS record header is also provided!

Task

Implement TLS 1.2 client that can retrieve server's certificate.

```
$ ./tls_getcert.py https://www.eesti.ee/ --certificate server.pem
--> client_hello()
<--- handshake()
        <--- server_hello()
        [+] server randomness: 0A801E9E809F15D7BCDB3A4F9640A3395480E7EF41FC9E6BD9B9438ECD67
        [+] server timestamp: 1975-08-02 02:43:26
        [+] TLS session ID: B268B48206AC28679B315CAB6CF5D0EEB5B0E50A973097EF1AFE20C23E8520
        [+] Cipher suite: TLS_RSA_WITH_AES_256_CBC_SHA
<--- handshake()
        <--- certificate()
        [+] Server certificate length: 1636
        [+] Server certificate saved in: server.pem
<--- handshake()
        <--- server_hello_done()
--> alert()
[+] Closing TCP connection!

$ openssl x509 -in server.pem -text | grep 'Subject:'
Subject: C=EE, ST=Harjumaa, L=Tallinn, O=Estonian Information System Authority, CN=*.eesti
```

3	0.004	172.17.57.208	195.80.123.145	TCP	66	56804 > https [ACK] Seq=1 Ack=1 Win=14720 Len=0 TSval=6
4	0.004	172.17.57.208	195.80.123.145	TLSv1	118	Client Hello
5	0.008	195.80.123.145	172.17.57.208	TLSv1	4410	Server Hello
6	0.008	172.17.57.208	195.80.123.145	TCP	66	56804 > https [ACK] Seq=53 Ack=4345 Win=23296 Len=0 TSv
7	0.011	195.80.123.145	172.17.57.208	TLSv1	530	Certificate, Server Hello Done
8	0.011	172.17.57.208	195.80.123.145	TCP	66	56804 > https [ACK] Seq=53 Ack=4809 Win=26240 Len=0 TSv
9	0.016	172.17.57.208	195.80.123.145	TLSv1	73	Alert (Level: Fatal, Description: Certificate Unknown)

Task: Other Test Cases

```
$ ./tls_getcert.py https://www.ut.ee/
--> client_hello()
<--- handshake()
        <--- server_hello()
        [+] server randomness: 5AE09A3AD38FEBD7F6541A8E9E54813303D397CBFBD3CF667D615EFE73D3
        [+] server timestamp: 2018-04-25 18:09:46
        [+] TLS session ID: 87500CF65DA6B1DD5BA9C675697802254AA383EFD29A403E76C8EC33B723628
        [+] Cipher suite: TLS_RSA_WITH_RC4_128_SHA
<--- handshake()
        <--- certificate()
        [+] Server certificate length: 1604
<--- handshake()
        <--- server_hello_done()
--> alert()
[+] Closing TCP connection!

$ ./tls_getcert.py https://danskebank.ee/
--> client_hello()
<--- handshake()
        <--- server_hello()
        [+] server randomness: 5AE0993AA7777BD76CF82BF906842C6B3A71B6E3E6378528CEDEA00CBA53
        [+] server timestamp: 2018-04-25 18:05:30
        [+] TLS session ID: C37DDFE7B056C2FBD56BDFC93CC5934F78BB7C2668DA5A106E836F5D927FC63
        [+] Cipher suite: TLS_RSA_WITH_AES_256_CBC_SHA
        <--- certificate()
        [+] Server certificate length: 1916
        <--- server_hello_done()
--> alert()
[+] Closing TCP connection!
```

Task: Hints

- Compare your parsed output with output from Wireshark
  - Use capture filters `'host 1.2.3.4 and port 443'`
- NB! One TLS record can contain several handshake messages
- Unix timestamp can be obtained using `int(time.time())`
- Unix timestamp can be printed using:
  `datetime.datetime.fromtimestamp(int(time.time())).strftime('%Y-%m-%d %H:%M:%S')`
- http://blog.fourthbit.com/2014/12/23/traffic-analysis-of-an-ssl-slash-tls-session