Internet Security - Transport Layer Comparative Study: TLS 1.2 VS TLS 1.3

Informatik Seminar

Degree course: Computer Science

Authors: Anna Doukmak, Rajina Kandiah

Group: 23

Tutor: Simon Kramer

Date: 09.01.2019

Berner Fachhochschule | Haute école spécialisée bernoise | Bern University of Applied Sciences

Abstract

The Transport Layer Security (TLS) is used for secured and encrypted communication over the networks. 2018 was the release date of the new TLS 1.3 which is defined in RFC8446. In this study we are going to figure out, what is new, what the changes to TLS 1.2 exactly are and what has remained the same.

The questions of the particular improvement and the consequences, for example of the configurations or security will be identified. What was removed will also be noted and explained.

The first part of the study will give a brief overview about TLS and the different protocols. This part will give an overall explanation of how the process of the securing and encryption of the communication works.

The main part will be the comparison of TLS 1.3 and TLS 1.2. This will be realised with the comparison of the changes. Namely the handhsake protocol, the session ressumption, the key derivation and supported ciphersuites which will be faced and analysed.

The conclusion will give a general overview based on the gathered information from the comparison.

Contents

1 Introduction

Transport Layer Security (TLS) Protocol is a cryptographic protocol that provides secure transport connection between applications over a computer network, e.g. web server and web browser. Using TLS prevents eavesdropping, tampering and message forgery. It provides privacy and data integrity between communicating applications. The protocol secures transmitted data using encryption. Secure communication is enabled with Authenticated Key Etablishment (AKE) and secure channel with Authenticated Encryption (AE). Using Datagram TLS (DTLS) the stateless, TLS is feasible. It offers server and, optionally, client authentication to confirm the identities of parties involved in the communication.

Furthermore, the integrity check value implemented in the protocol provides integrity for the transferred data. TLS is also known as Secure Socket Layer (SSL), its predecessor.

To use TLS all communicators must know that the other is supporting TLS.

TLS 1.2 was defined in RFC5246 in August 2008. Its successor TLS 1.3 is defined in RFC8446 in August 2018. [3][7]

1.1 Structure of the TLS Protocol

The TLS Protocol is layered between the Application layer and the TCP/IP layer according to the Internet Model (or between Session and Transport layers according to the OSI Model) where it can secure and then send application data to the transport layer [7]. Thus it can support multiple application layer protocols such as HTTP, FTP, SMTP or POP3. The protocols using TLS become respectively HTTPS, FTPS, SMTPS, POP3S and so on.

The TLS Protocol can be split into two layers. The lower layer of the TLS is the Record Protocol. The upper layer is the Handshake layer which consists of the following protocols: Handshake Protocol, ChangeCipherSpec Protocol and Alert Protocol. ChangeCipherSpec has been removed from the 1.3 version. Figure 1.1 illustrates the structure of the TLS Protocol. [5]

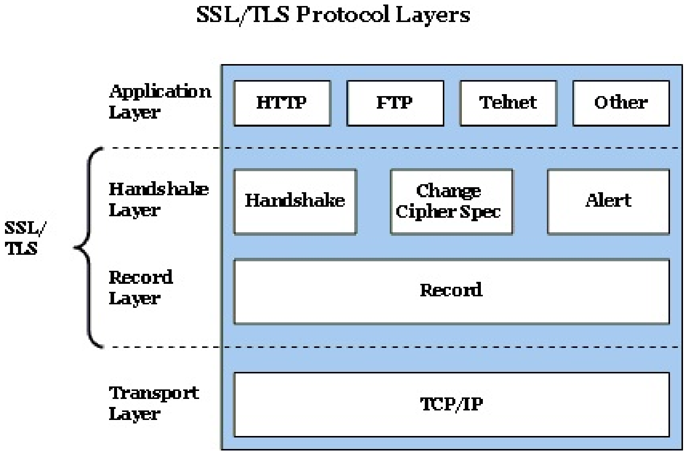


Figure 1.1: TLS 1.2 Protocol Structure [7]

1.2 Handshake Protocol

A handshake is an automated process of negotiation between communicators. The TLS Handshake Protocol is the most widely-used AKE Protocol to negotiate session information between the client and the server. After the establishment of the TCP connection between a client and a server the handshake protocol will be triggered. Thus, the purpose of this protocol is to negotiate and define these parameters:

- to authenticate the server and optionally also the client
- to agree on the version of the TLS
- to have an agreement of the cipher suites
- to arrange with the extension
- to derive authenticated encryption key for the connection
- if required, to verify the certificates
- finally all these points have to be assured between each communicators

To protect data, these parameters are then used by the record layer. As mentioned, usually it is only the server which is authenticated, but once a secure channel is established, the mutual authentications may also be carried out. This is permitted with the use of renegotiation but requires a public key infrastructure (PKI). [7] [9]

1.3 ChangeCipherSpec Protocol

This protocol is responsible for informing about a change of one set of keys to another. It means that a transition in ciphering would be signaled through this protocol. This message is either sent by the server or the client.

This protocol triggers an instruction to the record layer.

The keying material is raw data which is used for encryption between the client and server. On the basis of the exchanged information from the handshake protocol, the keys will be then computed. The ChangeCipherSpec Protocol consists of a byte with value 1. The ChangeCipherSpec Protocol also includes a message to inform other parties in the SSL/TLS session about the change. [7]

The ChangeCipherSpec protocol is removed from the 1.3 version of the TLS protocol. [14]

1.4 Alert Protocol

The peer will be notified by the alert messages about the change in status or error condition (fatal/warning).These messages are also compressed and encrypted. If a fatal error occurs, the session will close immediately. Fatal:

- unexpected_message
- bad_record_MAC
- handshake_failure
- etc.

Warning:

- close_notify
- unsupported_certificates
- certificate_expired
- etc.

The full list of the alerts to notify the peer of normal or error condition is declared in RFC2246 [2]. All these alerts consist of 2 bytes. The first byte defines the kind (eg. fatal or warning) and the second specifies what happened [11] [7]

1.5 Record Protocol

The connection of two independent channels is established through the TLS Handshake, from the client to the server and in the other direction from the server to the client. The Record Protocol is responsible for the data protection on these channels using the authenticated encryption scheme.
The responsibilities of the Record Protocol are:

- fragmentation
- compression
- application of MAC and encryption
- transmisson

But the tasks/responsibilities also depend on the parameters of the keys, namely how the keys were set during the handshake protocol.
The processing of data is illustrated and described below:

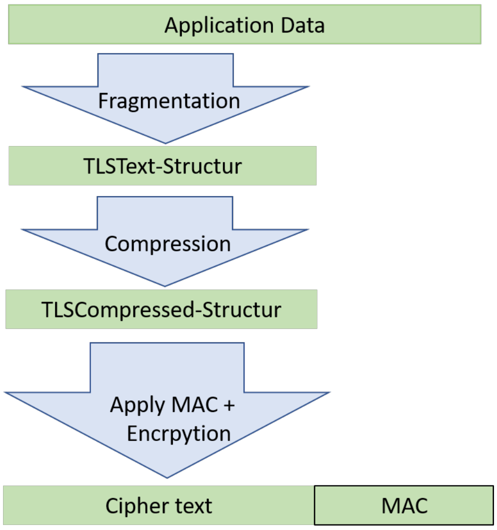


Figure 1.2: TLS Record Protocol

First the Record Protocol layer receives the data from the application layer. Then it fragments the data to a size appropriate for the cryptographic algorithm. The next step is the compression, which is done if specified. The compression algorithm would then be defined in the current session state and would translate the date to a TLSCompressed structure.

Subsequently the message authentication code over the compressed data would be computed. To ensure that the data were not altered during the transmission, the MAC value is added so the receiver can check if the incoming MAC value and the computed one match. Thus, the integrity and confidentiality are ensured.

What should be mentioned is, the first handshake is neither secured by a MAC nor encrypted. Because this is the initial value for the keys, it is how they are first set. After the first handshake these parameters are set and therefore secured and encrypted. [8] [6]

2 Comparative Study

The version 1.2 of the TLS protocol was defined in RFC5246 in August 2008 und has been in use for many years. With the release of TLS 1.3 in August 2018 some security and speed improvements of the protocol have been implemented. The TLS 1.3 is defined in RFC8446. In the following chapters the changes and differences of both TLS releases are examined and discussed.

2.1 Handshake

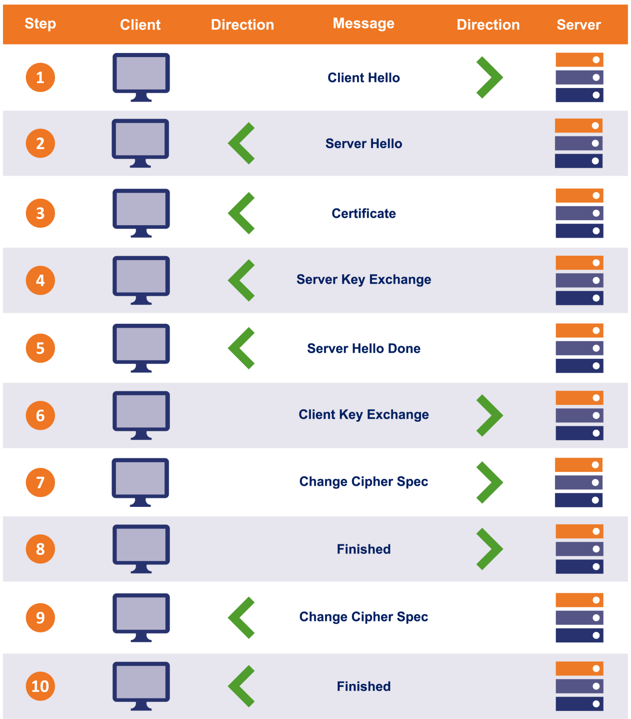
As described in the chapter 1.2 the Handshake begins after the establishment of the TCP connection between client and server. The goals of the handshake protocol are:

- to authenticate the server and, optionally, the client
- negotiate protocol version, cipher suites and extensions
- derive authenticated encryption keys for the secure connection
- ensure agreement on all negotiated parameters

[6]

2.1.1 TLS 1.2 handshake

The figure 2.1 shows the process of the full handshake in the TLS protocol 1.2 for simplicity only with authentication of the server. There will be more steps of the handshake in the case of mutual authentication when the client must also be authenticated by the server.



Step	Client	Direction	Message	Direction	Server
1	🖥		Client Hello	>	🗄
2	🖥	<	Server Hello		🗄
3	🖥	<	Certificate		🗄
4	🖥	<	Server Key Exchange		🗄
5	🖥	<	Server Hello Done		🗄
6	🖥		Client Key Exchange	>	🗄
7	🖥		Change Cipher Spec	>	🗄
8	🖥		Finished	>	🗄
9	🖥	<	Change Cipher Spec		🗄
10	🖥	<	Finished		🗄

Figure 2.1: TLS 1.2 full handshake [13]

Step 1. The 1st round of the handshake begins with the "client hello" message sent from the client to the server. This message includes the following cryptographic information:

- CipherSuites - encryption algorithms supported by the client
- desired maximum of the protocol version
- random value generated on the client side
- session ID

Step 2. Then the server responds with "server hello" message. The message consists of the following information:

- CipherSuites chosen by the server from the "client hello" message

- protocol version supported by the server
- random value generated on the server side
- session ID

Step 3. The server sends its X.509 certificate and its public key.

Step 4. This step is needed if the Diffie-Hellmann key exchange algorithm was negotiated between the client and the server at the steps 1 and 2. In this case the server transmits additional key materials in the "server key exchange" message.

Step 5. With the "server hello done" message the server notifies the client that it finished his steps and waits for the answer of the client.

Step 6. The client verifies the certificate sent by the server. Then it transmits to the server its key material in the key exchange message. At this point the client and the server can compute a pre-master secret from key exchange messages. Using the pre-master secret with the nonces sent in the steps 1 and 2 they can generate the master key. Afterwards the client and the server derive a set of session keys from the master key that will be used to symmetrically encrypt the data.

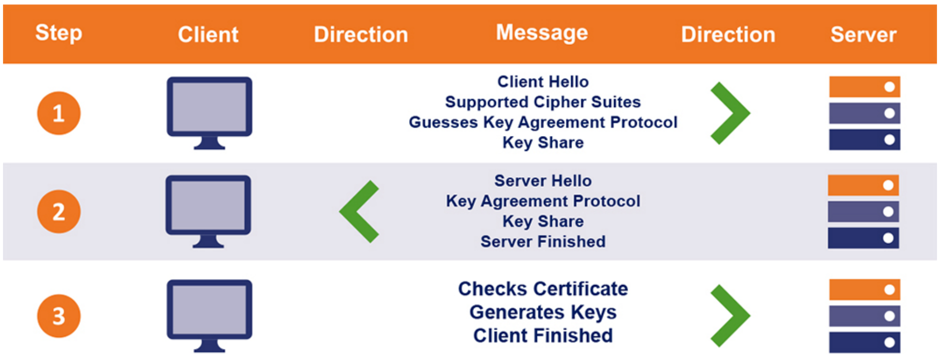
Step 7. Once the client has derived the session key, it notifies the server about the change of the keys for the secure communication.

Step 8. With the message "finished" the client signals to the server that it completed its tasks and is ready for the communication. This message includes MAC of the messages from the previous steps using the calculated master key, so that the server can verify the integrity of the handshake's messages.

Step 9-10. The server makes the same steps as the client in the steps 7-8 to switch the keys for the symmetric encryption and notifies the client with the message "finished". This message includes the MAC of the whole handshake log except "change cipher spec" messages from the steps 7 and 9. [13][6]

Almost the whole communication in the handshake protocol flows in the clear text. Only beginning from step 8, when the client has finished its preparation for the secure communication, will the next messages of the handshake be encrypted.

2.1.2 TLS 1.3 handshake



Step	Client	Direction	Message	Direction	Server
1			Client Hello Supported Cipher Suites Guesses Key Agreement Protocol Key Share	>	
2		<	Server Hello Key Agreement Protocol Key Share Server Finished		
3			Checks Certificate Generates Keys Client Finished	>	

Figure 2.2: TLS 1.3 full handshake [13]

Step 1. The TLS 1.3 handshake begins with the "client hello" message as in the TLS 1.2 handshake. In addition, the client also sends the following information with the message:

- the list of the supported CypherSuites
- key share entries consisting of a named (EC)DH group and an ephemeral public key

Step 2. The server respondes with the "server hello" message. In addition, it sends the following information:

- chosen CypherSuite

- key share entries consisting of the chosen (EC)DH group and its public key
- the certificate of the server
- "certificate verify" message containing a hash of all previous handshake messages signed with the private key of the server (the client afterwards verifies the signature with the public key from the server's certificate)
- "server finished" message

In case the server does not support any of the proposed groups, the server will request to retry the handshake or will abort the connection with a fatal handshake_failure alert.

Step 3. The client checks the certificate of the server. It generates keys from the key share of the server from the previous step. Afterwards the client sends the "client finished" message. [6][13]

After the "server hello" message all handshake messages will be encrypted. [10]

2.1.3 Discussion of TLS 1.2 and 1.3 full handshakes

The TLS 1.2 handshake makes two round trips between the client and the server to complete the handshake. On average this process requires 0.25 to 0.5 seconds.

The "server key exchange" and "client key exchange" messages have been removed from the TLS 1.3 handshake. The key exchange parameters and public keys will be sent in the key share extensions, that are added to the "client hello" and "server hello" messages. This keeps the 1.3 release compatible with the 1.2 version due to the remaining sending-order of messages.

Furthermore, the ChangeCipherSpec protocol has been removed in the 1.3 version.

As a consequence of this missing protocol, the TLS 1.3 handshake makes only one round trip. This change results in reduced latency almost by half. As mentioned in the blog of The SSL Store a delay of half a second results in 20% traffic decline [13]. That is why the performance improvement of the TLS protocol is a crucial change in the protocol.

Moreover, the privacy of the handshake protocol has been improved. In the 1.2 release of the TLS protocol almost all handshake messages are sent in the clear text except the last steps after the client sends "finished" message. On the contrary in the 1.3 version all information will be encrypted as early as possible, namely after "server hello" message.

2.2 Session resumption

The full handshake costs time, increases server load and causes connection latency. For performance improvement the session resumption can be used. With the "client hello" and "server hello" messages the session ID can be saved and used to resume the previously established TLS session. Under this session ID the client and the server store the master key and other details of the connection. In the next session they can transmit the session ID and thus reuse the connection parameters. The following session data can be cached:

- master secret
- protocol version
- cipher suits
- compression method
- certificate

Because the session resumption is efficient it is supported by default in all major web browsers and web servers. But if it is not correctly implemented that can lead to vulnerabilities.

2.2.1 Session resumption in TLS 1.2

The client and the server can set up a new connection by reusing the master key from the recent session cached on both ends in the one round-trip handshake. This kind of handshake is called abbreviated TLS handshake. The figure 2.3 illustrates the process of the abbreviated handshake for the session resumption in the TLS 1.2.

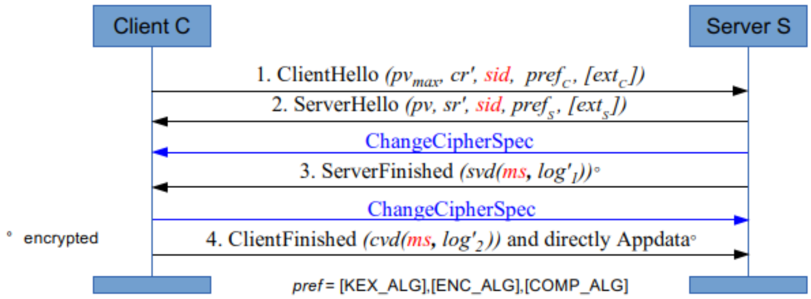

Client C
Server S
1. ClientHello (pv_max, cr', sid, pref_C, [ext_C])
2. ServerHello (pv, sr', sid, pref_S, [ext_S])
ChangeCipherSpec
3. ServerFinished (svd(ms, log'_1))°
ChangeCipherSpec
° encrypted
4. ClientFinished (cvd(ms, log'_2)) and directly Appdata°
pref = [KEX_ALG],[ENC_ALG],[COMP_ALG]

Figure 2.3: Abbreviated Handshake for Session Resumption in TLS 1.2 [6]

As session ID, which is sent by the client in the "client hello" message to the server, can be used from the information of connection parameter. A new random value (nonce) should also be transmitted by the client. If the session has been cached the server responds in the "server hello" message with a new nonce and the same CipherSuites as in the previous handshake. Without waiting for the client's answer, the server immediately notifies the client about the changes of the key with the ChangeCipherSpec message and finishes its part of the handshake with MAC of the abbreviated handshake log in the encrypted "server finished" message. The client answers with its notification about the change of the key and completes the handshakes process with the "client finished" message containing a MAC of the whole abbreviated handshake log with the exception of ChangeCipherSpec messages. Then both parties use the master key from the previous handshake and the new nonce to derive a set of session keys for the symmetric encryption of their communication.

The session resumption is also possible with the session ticket. In this case the client transmits the session ticket in the "client hello" message instead of the session ID. The session ticket is a blob file created by the server and stored on the client side. This file contains all necessary detail of a connection and is encrypted with the key only known to the server. After the decryption of the ticket by the server the master key will be resumed for the derivation of session keys.

Figure 2.4 demonstrates the efficiency of the session resumption. The abbreviated handshake is almost twice as fast as the full handshake.

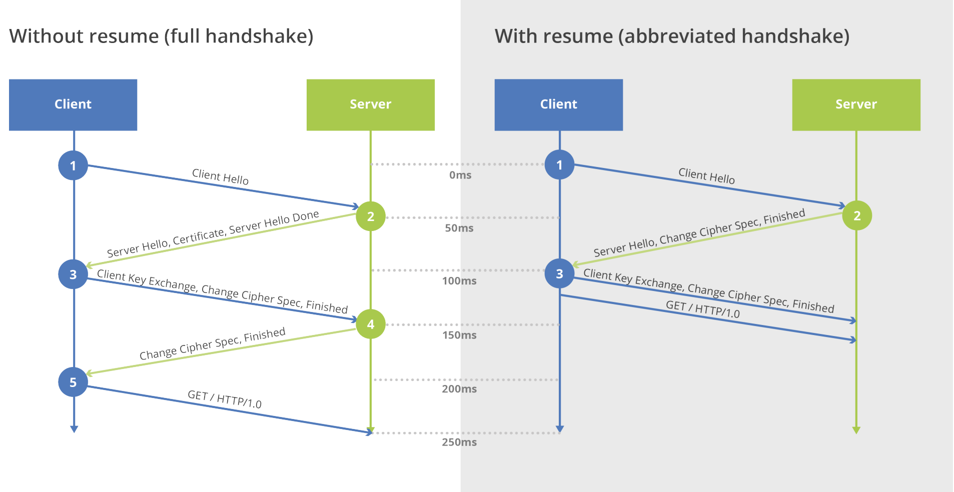

Without resume (full handshake)
With resume (abbreviated handshake)
Client
Server
Client
Server
1
Client Hello
2
Server Hello, Certificate, Server Hello Done
3
Client Key Exchange, Change Cipher Spec, Finished
4
Change Cipher Spec, Finished
5
GET / HTTP/1.0
0ms
50ms
100ms
150ms
200ms
250ms
1
Client Hello
2
Server Hello, Change Cipher Spec, Finished
3
Client Key Exchange, Change Cipher Spec, Finished
GET / HTTP/1.0

Figure 2.4: Full TLS handshake vs. abbreviated TLS handshake [4]

2.2.2 Session resumption in TLS 1.3

There are differences in how the session resumption works in the release 1.3 of TLS. Figure 2.5 illustrates the process of the handshake for the session resumption in TLS 1.3.

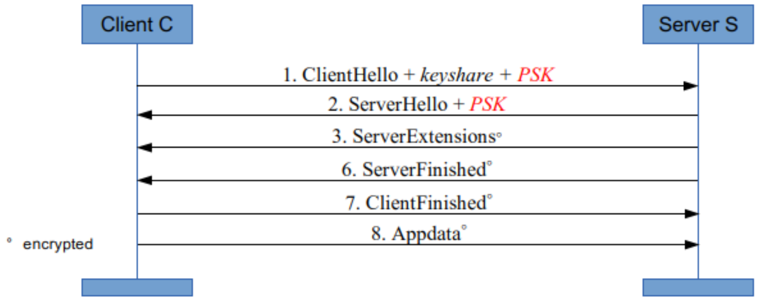


Figure 2.5: Handshake for Session Resumption in TLS 1.3 [6]

A new method to resume the session is implemented in TLS 1.3. It is called pre-shared key (PSK) mode and replaces the usage of the session identifier or session ticket. An PSK identity is sent to the client by the server after the full handshake is completed. The PSK identity is a blob file containing database lookup keys or self-encrypted and self-authenticated values. It corresponds to a key derived from the handshake.

A PSK, that has been created during the handshake of the previous connection can be presented by the client on the next visit. If it is necessary to resume the session, the client transmits PSK identities to the server. Then the server responds with the chosen PSK identity in the "server hello" message. The security context of this identity will be used in the new connection and the unique key derived from the previous connection can be used in the new one. The client can transmit the data to the server already in the first step together with the PSK. The data will be encrypted with the key derived from this PSK. This is the case of "Zero round-trip" handshake. [15]

The second PSK exchange mode is available in the session resumption in TLS 1.3. This mode additionally includes a key agreement using ephemeral Diffie-Hellman and thereby making this mode of the session resumption forward secure. A shared ECDHE/DHE key will be mixed into the derived master keys of the previous session. Thus an attacker can not steal ticket keys to recover the communication data of the previous session. [12]

2.2.3 Discussion of the session resumption in TLS 1.2 and 1.3

Though the performance of the session resumption in TLS 1.2 is very fast, it can lead to some security problems. Using the same session ID during the resumption the same master secret is used. In case of compromised master key all resumed sessions will be revealed. But setting the lifespan the server remembers a client session helps to reduce this risk. The vast majority of browsers clear TLS session information after an hour [1].

Using 0-RTT handshake in TLS 1.3 does not provide perfect forward secrecy. It also does not protect against replay attack in contrary to 1-RTT where the server transmits a random value in each handshake. [10] In this case the security level is basically the same as in the session resumtion in TLS 1.2 if the server never rotates keys or delete cache entries after their expiration [12].

2.3 Cipher suites

A cipher suite contains the key exchange and authentication algorithms for the handshake, the encryption and message authentication code (MAC) algorithm for the record protocol. The cipher suite contains different combinations of these algorithms.

2.3.1 Cipher suites in TLS 1.2

The cipher suites for key exchange and the authentication in TLS 1.2 are described.

TLS_RSA (Static RSA)

The key exchange is based on the key transport with RSA public-private keys. If this cipher suite is used, the following computations and communications are made during the handshake:

- for the pre-master secret, the protocol version is concatenated with the 46 bytes random value generated by the client
- the "server key exchange" message in step 4 of the full handshake is not used
- the client transmits in the "client key exchange" message in the step 6 the pre-master secret encrypted with the public key of the server taken from its certificate
- the server which decrypts the message, gets the pre-master secret
- both parties derive the master secret from it and a set of session keys

The keyEncipherment option must be enabled in the key usage extension of the certificate of the server. This cipher suite does not offer perfect forward secrecy. If an adversary gets the private key of the server, he can decrypt the pre-master secret from the client, compute all session keys and can then decrypt the whole communication with them.

DH_RSA/DH_DSS

The fixed Diffie-Hellman is used for the key exchange. The certificate of the server contains the DH public key and is signed with RSA or DSS. The keyAgreement flag must be enabled in the key usage extension of the certificate.

The following computations are made during the handshake:

- the "server key exchange" message in step 4 of the full handshake is not used
- the client transmits in the "client key exchange" message in step 6 - its value for DH exchange is: $kex_c = g^c \bmod p$
- the client and the server compute the pre-master secret as follows: $pms = g^{cs} \bmod p$
- both parties derive the master secret from the pre-master secret and a set of session keys

DHE_RSA/DHE_DSS

The key exchange is based on the Ephemeral Diffie-Hellman and the server authentication - on the RSA/DSS signature.

The following computations are made during the handshake:

- the "server key exchange" message is transmitted by the server in step 4
- the "server key exchange" message contains the parameters of a RSA group and an ephemeral public key computed by the server
- the "server key exchange" message is signed with a private key of the server
- the client verifies the signature of the message with a public key from the certficate of the server
- the client transmits in the "client key exchange" message in step 6 its ephemeral public key, which is compatible with the parameters sent by the server
- the client and the server compute the pre-master secret
- both parties derive the master secret from the pre-master secret and a set of session keys

The digitalSignature flag must be enabled in the key usage extension defined in the certificate of the server.

ECDHE_RSA

The key exchange is based on the Ephemeral Diffie-Hellman using elliptic curves and the server authentication - on the RSA/DSS signature.

The following computations are made during the handshake:

- the "server key exchange" message is transmitted by the server in step 4
- the "server key exchange" message contains the parameters of an elliptic curve and an ephemeral public key computed by the server
- the "server key exchange" message is signed with the private key of the server
- the client verifies the signature of the message with the public key from the certficate of the server
- the client transmits in the "client key exchange" message in step 6 its ephemeral public key, which is compatible with the parameters sent by the server
- the client and the server compute the pre-master secret
- both parties derive the master secret from the pre-master secret and a set of session keys

The digitalSignature flag must be enabled in the key usage extension of the server's certificate.

RSA_anon

The key exchange is based on the Anonymous Diffie-Hellman. In this case neither the server nor the client is authenticated, both parties remain anonymous. But the communication with this cipher suite is vulnerable to the man-in-the-middle attacks.

The following computations are made during the handshake:

- the server sends the "server key exchange" message in step 4 with the following value: $kex_s = g^s \bmod p$
- the client transmits in the "client key exchange" message in step 6 its value for DH exchange: $kex_c = g^c \bmod p$
- the client and the server compute the pre-master secret as follows: $pms = g^{cs} \bmod p$
- both parties derive the master secret from the pre-master secret and a set of session keys

Cipher suites for encryption

In the last years many attacks have been performed against commonly used cipher suites and their modes of operation in TLS 1.2. For instance the stream cipher RC4 is used though its weaknesses have been proved and it has been recognized as insecure. There are some attacks against CBC mode of AES encryption if it is not correct implemented. The list of stream and block encryption algorithms are presented in Tabletab:comparison.

2.3.2 Cipher suites in TLS 1.3

The cipher suite concept has been changed to separate the authentication and key exchange mechanisms. For the key exchange in TLS 1.3 only DHE and ECDHE are used providing the perfect forward secrecy. Five elliptic curve groups and five DHE groups from Galois field are predefined. That eliminates the risk of the unsecure implementation of TLS by the administrators and developers.

The following signature algorithms remain:

- EdDSA (Edwards-curve Digital Signature Algorithm)
- ECDSA (Elliptic Curve Digital Signature Algorithm) and RSA

The list of supported symmetric algorithms has been reduced to Authenticated Encryption with Associated Data (AEAD) algorithms. The weak cryptographic algorithms have been deprecated, such that using static RSA in the key exchange, RC4 stream cipher, MD5 and SHA-1 hash functions. The compression of the plaintext before the applying of MAC is removed. The whole list of the algorithms used in the release 1.3 is shown in table 2.1. [10]

2.4 Key derivation

For the key derivation in TLS 1.2 HMAC with SHA256 as pseudo-random function (PRF) is used. The PRF takes the pre-master key as an input for the master key derivation.
The key derivation funcitons habe been re-desired in TLS 1.3. The HMAC-based Extract-and-Expand Key Derivation Function (HKDF) is used as an underlying primitive [6]. It replaces PRF based on HMAC.

2.5 Comparison: TLS 1.2 vs 1.3

The Table 2.1 illustrates a brief summarized comparison of both versions of TLS.

Characteristic	**TLS 1.2**	**TLS 1.3**
Round trip time of the full handshake	2-RTT	1-RTT
Encryption of handshake messages	at the end, from ”client finished” message	at the beginning, after ”server hello” message
Performance of the full handshake	0.25-0.5 s	0.2-0.3 s
Round trip time of the handshake for session resumption	1-RTT	0-RTT
Session resumption method	session ID or session ticket	pre-shared key identity
Key exchange algorithms	RSA RSA/DHE ECDH Secure Remote Password Pre-shared key	DHE ECDHE
Signature algorithms	RSA DSS ECDSA	RSA ECDSA EdDSA
Symmetric encryption algorithms	AES GSM, CCM, CBC ChaCha20-Poly1305 3DES RC4 Camellia et al.	AES GCM AES CCM ChaCha20-Poly1305
Hash algorithms	MD5 SHA-256 SHA-384	SHA-256 SHA-384
Key derivation	PRF based on HMAC	HKDF-Extract and HKDF-Expand functions
Compression	Yes	No
Renegotiation	Yes	No
Known attacks	BEAST CRIME TIME BREACH POODLE Lucky 13 FREAK Logjam DROWN	

Table 2.1: Comparison of TLS 1.2 and TLS 1.3

3 Migration from TLS 1.2 to TLS 1.3

To support the usage of the TLS 1.3 protocol the following changes should be made in the Apache configuration file by administrators of a web server:

1. Enable TLS 1.3 in the optionSSLProtocol. Example of the configuration:

   ```
   SSLProtocol TLSv1.3
   ```

2. Disable if necessary other versions of TLS. Example:

   ```
   SSLProtocol TLSv1.3 -TLSv1.2
   ```

3. Configure the supported ciphersuites in the option SSLCipherSuite. The following ciphersuites are available in TLS 1.3:

   - TLS-AES-128-GCM-SHA256
   - TLS-AES-256-GCM-SHA384
   - TLS-CHACHA20-POLY1305-SHA256
   - TLS-AES-128-CCM-SHA256

   Example of the configuration:

   ```
   SSLCipherSuite TLS-AES-128-GCM-SHA256:TLS-AES-256-GCM-SHA384:
   TLS-CHACHA20-POLY1305-SHA256:TLS-AES-128-CCM-SHA256
   ```

   Respectively the configuration for Nginx:

   ```
   ssl_protocols TLSv1.3;
   ssl_ciphers "TLS-AES-128-GCM-SHA256:TLS-AES-256-GCM-SHA384:
   TLS-CHACHA20-POLY1305-SHA256:TLS-AES-128-CCM-SHA256";
   ```

4 Conclusion

As conclusion of the gathered information about TLS, its use, the comparison of the new and the prior version and with the experiences of the migration from the older to the newer version, the following can be state:

TLS 1.3 has significant changes with lots of simplifications, which results in a improved security and better performance during communications and transactions.

- PFS: TLS 1.3 now provides perfect forward secrecy (PFS),beside in 0-RTT, which means that each session is protected separately, even if any transmission is interrupted. So, there will be no common key.
- 0-RTT: The entire round-trip can be avoided because the client now has the possibility to reconnect to a server to which it has been connected before. This will improve the load time.
- handshake: The full handshake in TLS 1.2 costs time, increases server load and causes connection latency. This is now minimized.
- Strong ciphers: TLS 1.3 makes only use of the strong ciphers (AES-GCM and ChaCha-Poly), therefore it will be easier to configure applications that are already secured by default.

Therefore it is very useful and important to update to TLS 1.3.

Declaration of primary authorship

We hereby confirm that we have written this thesis independently and without using other sources and resources than those specified in the bibliography. All text passages which were not written by me are marked as quotations and provided with the exact indication of its origin.

Place, Date: Biel, 09.01.2019

Last Names, First Names: Anna Doukmak Rajina Kandiah

Signatures: ....

Glossary

(EC)DH (Elliptic curve) Diffie-Hellman.

AE Authenticated Encryption.

AEAD Authenticated Encryption with Associated Data.

AKE Authenticated Key Etablishment.

DH Diffie-Hellman.

DHE Diffie-Hellman Ephemeral.

DSS Digital Signature Standard.

DTLS Datagram Trasport Layer Security.

ECDHE Elliptic curve Diffie-Hellman Ephemeral.

ECDSA Elliptic Curve Digital Signature Algorithm.

EdDSA Edwards-curve Digital Signature Algorithm.

FTP File Transfer Protocol.

FTPS File Transfer Protocol Secure.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

IP Internet Protocol.

MAC Message Authentication Code.

OSI Open Systems Interconnection.

PKI Public Key Infrastructure.

POP3 Post Office Protocol version 3.

POP3S Post Office Protocol version 3 Secure.

PSK Pre-Shared Key.

RC4 Rivest Cipher 4.

RFC Request for Comments.

RSA RivestShamirAdleman cryptosystem.

RTT Round Trip Time.

SMTP Simple Mail Transfer Protocol.

SMTPS Simple Mail Transfer Protocol Secure.

SSL Secure Socket Layer.

TCP Transmission Control Protocol.

TLS Trasport Layer Security.

Bibliography

[1] C. Cimpanu, "Advertisers can track users across the internet via tls session resumption," October 2018, accessed on 09.01.2019. [Online]. Available: https://www.zdnet.com/article/advertisers-can-track-users-across-the-internet-via-tls-session-resumption/

[2] T. Dierks and C. Allen, "The tls protocol version 1.0," January 1999, accessed on 06.01.2019. [Online]. Available: https://tools.ietf.org/html/rfc2246

[3] T. Dierks and E. Rescorla, "The transport layer security (tls) protocol version 1.2," August 2008, accessed on 02.01.2019. [Online]. Available: https://tools.ietf.org/html/rfc5246

[4] P. R. Donahue, "hange the (s)channel! deconstructing the microsoft tls session resumption bug," Cloudflare, February 2016, accessed on 07.01.2019. [Online]. Available: https://blog.cloudflare.com/microsoft-tls-downgrade-schannel-bug/

[5] I. Grigorik, "High performance browser networking - chapter 4: Transport layer security (tls)," 2018, accessed on 07.01.2019. [Online]. Available: https://hpbn.co/transport-layer-security-tls/

[6] G. Hassenstein, "Security basics and applied cryptography: Chapter 12 - transport layer security," Berner Fachhochschule, 2018.

[7] Microsoft, "Overview of ssl/tls encryption," August 2009, accessed on 02.01.2019. [Online]. Available: https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc781476(v%3dws.10)

[8] Microsoft2, "Tls record protocol," May 2018, acessed on 04.01.2019,. [Online]. Available: https://docs.microsoft.com/en-us/windows/desktop/secauthn/tls-record-protocol

[9] Microsoft3, "Tls handshake protocol," May 2018, acessed on 04.01.2019,. [Online]. Available: https://docs.microsoft.com/en-us/windows/desktop/secauthn/tls-handshake-protocol

[10] E. Rescorla, "A readable specification of tls 1.3," August 2018, accessed on 06.01.2019. [Online]. Available: https://davidwong.fr/tls13/

[11] W. Stallings, "Ssl: Foundation for web security - the internet protocol journal - volume 1, no. 1," June 1998, accessed on 06.01.2019. [Online]. Available: https://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-18/ssl.html

[12] T. Taubert, "The future of session resumption," February 2017, accessed on 09.01.2019. [Online]. Available: https://timtaubert.de/blog/2017/02/the-future-of-session-resumption/

[13] J. Thakkar, "Tls 1.3 handshake: Taking a closer look," The SSL Store, March 2018, accessed on 05.01.2019. [Online]. Available: https://www.thesslstore.com/blog/tls-1-3-handshake-tls-1-2/

[14] Wikipedia, "Cipher suite," 2013, accessed on 07.01.2019. [Online]. Available: https://en.wikipedia.org/wiki/Cipher_suite

[15] J. Willeke, "Tls session resumption," LDAP Wiki, April 2018, accessed on 07.01.2019. [Online]. Available: https://ldapwiki.com/wiki/TLS%20Session%20Resumption

List of Figures

List of Tables