Transport Layer Security (TLS) protocol

Milan Welser

FEL, ČVUT

Winter 2023/2024

1 Table of Contents

1 Introduction

1.1 Project focus

The main focus of this project revolves around the Transport Layer Security (TLS) protocol. Specifically, to understand the basic principles of the protocol, its functionality, guarantees and options. Furthermore, to understand its safety and security, possible vulnerabilities, and risks.

1.2 TLS purpose

"The TLS protocol provides communications privacy over the Internet. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forger." (Dierks & Allen, RFC 2246 - TLS 1.0, 1999)

In other words, TLS ensures a protected channel for data transfer between two peers (client and server)

The TLS is a layered protocol, it requires a transport protocol to run below it, mostly Transmission Control Protocol (TCP), placing it just above the transport layer. TLS provides security to any two communicating peers using an application protocol (HTTP, IMAP, VoIP), making it mostly application independent. (McKay & Cooper, 2019)

1.3 TLS history

TLS predecessor was the Secure Sockets Layer (SSL) protocol. SSL version 1.0 was never released publicly. Version 2.0 of SSL was released in 1995 with known security issues, which were later addressed in 1996 with SSL 3.0 release.

From the original document specification for TLS 1.0:
"This document describes TLS Version 1.0, which uses the version {3, 1}. The version value 3.1 is historical: TLS version 1.0 is a minor modification to the SSL 3.0 protocol, which bears the version value 3.0."

Due to this historical reason, the used versions for TLS 1.0, 1.1, 1.2 and 1.3 are 3.1, 3.2, 3.3 and 3.4, respectively. Even the brief description of SSL 3.0 is the same as the TLS 1.0 quoted above from RFC 2246 and it has remained the same even in TLS 3.0. Although TLS 3.0 is technically only a minor version increase (3.3 to 3.4), it is a very major upgrade with some radical changes.

(Dierks & Allen, RFC 2246 - TLS 1.0, 1999)

2 TLS in principle

2.1 Main principle

The main principle of the TLS protocol is to provide a secure channel for communication between two peers (client and server). This is done by first establishing a secure channel by exchanging a key using an asymmetrical cipher, after which the peers can share a mutual secret key for further symmetrical communication. The asymmetrical cipher is too slow for encrypting all communication, but it ensures a secure parameter negotiation and most importantly, a secure transfer of a shared symmetrical key. Server authentication is also included in the initial part of the “handshake”.

2.2 TLS Record Protocol

The TLS protocol is the main means of TLS to transmit data. It is done by fragmenting it into manageable blocks, protecting them and transmitting them, so that these records can be reassembled at the other end and data used by the application.

TLS records are typed to distinguish between TLS subprotocols. Overall, 4 types exist, one for every subprotocol of TLS – *handshake*, *application_data*, *alert* and *change_cipher_spec*. Handshake protocol is used to negotiate the session and its parameters, the alert protocol is used to notify the other peer of any errors and the change cipher spec protocol is used to change the cryptographic parameters during the session up until version 1.2. In TLS 1.3 however, it is used only for compatibility purposes due to problems with 1.3 handshakes.

(Rescorla, 2018)

2.3 TLS Handshake Protocol

2.3.1 Parameters and cipher suite

The handshake protocol is used to negotiate the session and its security parameters. The client and server exchange a series of messages, negotiating parameters such as the TLS version, cipher suite, means of authentication and optionally other extensions of the TLS protocol.

The cipher suite usually specifies the specific algorithms to be used during the communication. This entails:

- a key exchange algorithm
- an encryption algorithm to be used for all communication after the handshake to ensure confidentiality
- hash algorithm to ensure data integrity

(McKay & Cooper, 2019)

An example of a cipher suite for TLS 1.2 would be TLS_DH_RSA_WITH_AES_128_CBC_SHA256.

DH signifies a Diffie-Hellman key exchange.

RSA signifies that the server shall use an RSA certificate for authentication.

AES_128_CBC is the symmetrical cipher to be used, with a 128bit key in Cipher Block Chaining (CBC) mode.

(Ciphersuite info - TLS_DH_RSA_WITH_AES_128_CBC_SHA256)

The goal of the handshaking protocol is to guarantee the following desirable properties of the connection: Data confidentiality, data integrity, authentication, non-replayability and even length concealment.

2.3.2 Confidentiality

"An attacker should not be able to determine the plaintext contents of a given record." (Rescorla, 2018)

Data confidentiality is ensured by the chosen symmetrical encryption algorithm, transmitted thanks to the asymmetrical key exchange method.

2.3.3 Integrity

"An attacker should not be able to craft a new record which is different from an existing record which will be accepted by the receiver."

The integrity provider is specified by the chosen cipher suite. Earlier versions of TLS include a keyed MAC (message authenticity code) for which the hash functions from the cipher suite are used. Later TLS versions use an AEAD encryption algorithm which provides both confidentiality and integrity for the data.

(Dierks & Rescorla, RFC 5246 - TLS 2.0, 2008)

2.3.4 Authentication

TLS only requires the server to be authenticated, the client authentication is optional.

During the handshake process, the server sends a certificate signed by a Certificate Authority (trusted by the client!) to the client, who verifies the signatures using public keys belonging to the certificate. Authentication can also be done implicitly by using the public server key during creation of the "master secret" – the key for symmetrically encrypted communication.

(McKay & Cooper, 2019)

2.3.5 Non-replayability

Non-replayability, also called order protection, ensures that the server will not accept an intercepted record sent by an attacker out of order or even multiple times. This is done simply by including an increasing sequence number of the record.

(Rescorla, 2018)

3 TLS versions overview

3.1 Deprecated protocol versions

3.1.1 SSL Mention

Both SSL 2.0 and SSL 3.0 are deprecated (2011 and 2015), quoting several weaknesses, such as use of MD5, vulnerabilities to exploits and overall broken functionality.

(Turner & Polk, 2011) (Barnes, Thomson, Pironti, & Langley, 2015)

According to NIST guidelines, no SSL version shall be used even in citizen or business-facing applications. Both are considered unsafe and not equipped to facilitate a secure communication channel.

3.1.2 TLS 1.0 and TLS 1.1

Both TLS 1.0 and 1.1, released in 1999 and 2006, respectively, have been deprecated since 2021. The main reason for the deprecation was problematic usage of SHA-1 hash algorithm or the concatenation of SHA1 and MD5 hashes. Due to the weakness of SHA-1, the handshake protocol is vulnerable to a downgrade attack (forcing the server to use an older version of TLS). The authentication part is also at risk as it relies on the SHA-1 hash as well, allowing the attacker to impersonate a server. Since there were no stronger hash functions offered to select as part of TLS 1.0 and 1.1, both were deprecated. Both versions are considered unsafe. NIST guidelines prohibit use of TLS 1.0 for US government servers and use of TLS 1.1 is highly discouraged.

Also, DES and IDEA cipher suites are removed from TLS 1.2. They were supported only because of TLS 1.0 and 1.1.

(Moriarty & Farrel, 2021)

3.2 Currently supported versions

3.2.1 TLS 1.2

TLS 1.2 was released in 2008. The combination of MD5 with SHA-1 is replaced with a stronger SHA-256 hash function (a reason for which earlier TLS versions were deprecated), specified through cipher suites.This version also introduced TLS Extensions, AES ciphers and Authenticated Encryption with Associated Data (AEAD) options. Overall, TLS 1.2 supports a very wide array of cipher suites with many key exchange algorithms as well as many bulk encryption algorithms with many operating modes. This is both an advantage and a disadvantage, as the wide support allows it to function securely today, but only with proper setup and mitigations (countermeasures) to avoid attacks and exploits.

(Dierks & Rescorla, RFC 5246 - TLS 2.0, 2008)

3.2.2 TLS 1.3

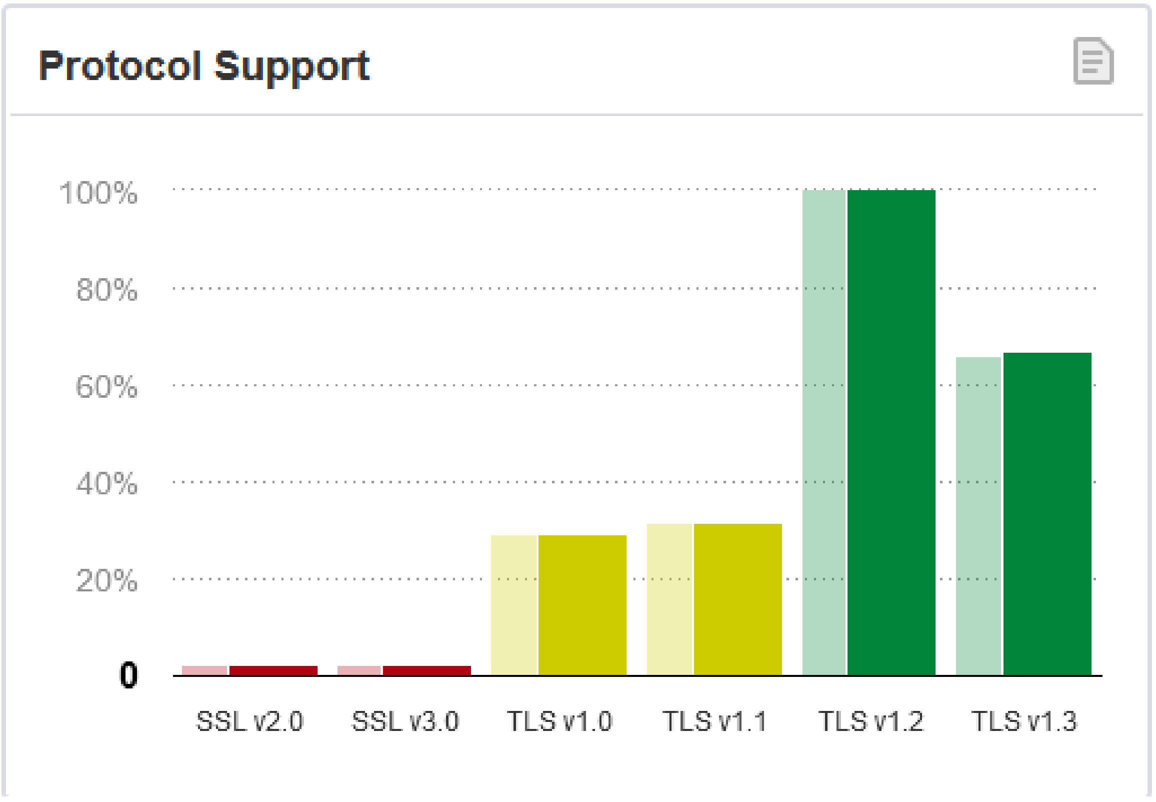
Released in 2018, TLS 1.3 is the probably the largest upgrade to the Transport Layer Security protocol. The number of supported algorithms has been radically reduced. For key exchange, static (reused) key algorithms are no longer supported and use of ephemeral (newly generated for every session) keys is required, ensuring forward-secrecy. Another important change was in the handshaking protocol. Only messages up until “ServerHello” are unencrypted. (Rescorla, 2018)

Overall, TLS 1.3 is the more secure of the two supported TLS versions. According to NIST, all agencies shall support TLS 1.3 by January 2024 and even though TLS 1.2 support is still recommended, however the guidelines advise that TLS 1.2 can be disabled if its not required for the operation of said server. (McKay & Cooper, 2019)

3.2.3 Current support and use of TLS

According to SSL Labs, who monitor SSL/TLS support across 150 000 SSL/TLS enabled most popular websites, up to 66.9% of monitored sites support use of TLS 1.3 and over 99% of monitored sites support TLS 1.2, as can be seen on Figure 1.

Figure 1 - SSL/TLS support



We can also see that 29% of sites support TLS 1.0 and 31% of sites support TLS 1.1. This is very much discouraged, as both versions use SHA1 hash function which is susceptible to attacks due to its low collision resistance. (SHAttered, n.d.)

4 TLS 1.2

4.1 Supported Cipher Suites

TLS 1.2 supports a wide array of cipher suites. According to ciphersuite.info, the total number of possible suites is 339. Only 12 of those are considered recommended and 15 on top of that considered secure. The remaining 312 are considered at least weak, some even Insecure.

(Ciphersuite.info - TLS 1.2)

The cipher suite consists of 4 elements. The key exchange method, the authentication method (certificate), the bulk encryption method (symmetrical) and the hashing method (An exception is RSA key exchange which uses RSA for both key exchange and authentication, listing only 3 elements).

4.1.1 Recommended cipher suites

Key Exchange - Recommended (very strong) cipher suites usalmost exclusively ECDHE (Elliptic Curve Diffie-Hellman Ephemeral) key exchange or occasionally the ECCPWD password based key exchange method (which includes authentication) (D. Harkins, 2019)

Authentication – ECDSA (Elliptic Curve Digital Signature Algorithm), PSK (Pre-Shared-Key, the parties shared the symmetric keys in advance and so they are authenticated) or ECCPWD.

Bulk Encryption – AES, ARIA and CAMELLIA are used with both 128 and 256 key sizes, however only in GCM mode. A stream cipher is also used, called CHACHA20-POLY1305.

HASH –SHA with at least 256 key size

4.1.2 Strong cipher suites

The main difference from the strongest cipher suites seems to be the use of RSA for authentication as well as the use of CCM modes for symmetrical encryption, both of which seem to be rated slightly weaker, but still secure.

4.1.3 Weak cipher suites

Key Exchange - There are no ephemeral key exchange methods among the cipher suites rated as weak. This removes the benefit of forward secrecy – if a key of one session is compromised, it may compromise the security of other sessions as well.

Bulk Encryption – The weaker cipher suites seem to also use the CBC mode for encryption. The CBC is considered weak due to the rising amount of padding attacks and difficulty of correct implementation. The change of rating aims to discourage further use. (Qualys, 2019)

4.1.4 INSECURE cipher suites

The insecure cipher suites use old, deprecated or even no security methods in some respects. Some are vulnerable to old exploits.

Key Exchange – sometimes no ephemeral key exchange methods are used

Authentication – Some of the suites use no means of authentication, exposing the communication to Man In The Middle attacks.

Bulk Encryption –RC4 encryption is used, which has been deprecated since 2015 due to security concerns. (Popov, 2015), some don’t even use any encryption (NULL) or use other deprecated encryptions (IDEA)

HASH - deprecated MD5 and SHA1 hash functions are used,

4.2 TLS 1.2 handshake

In a typical TLS 1.2 handshake only the server is authenticated. The client authentication is optional.

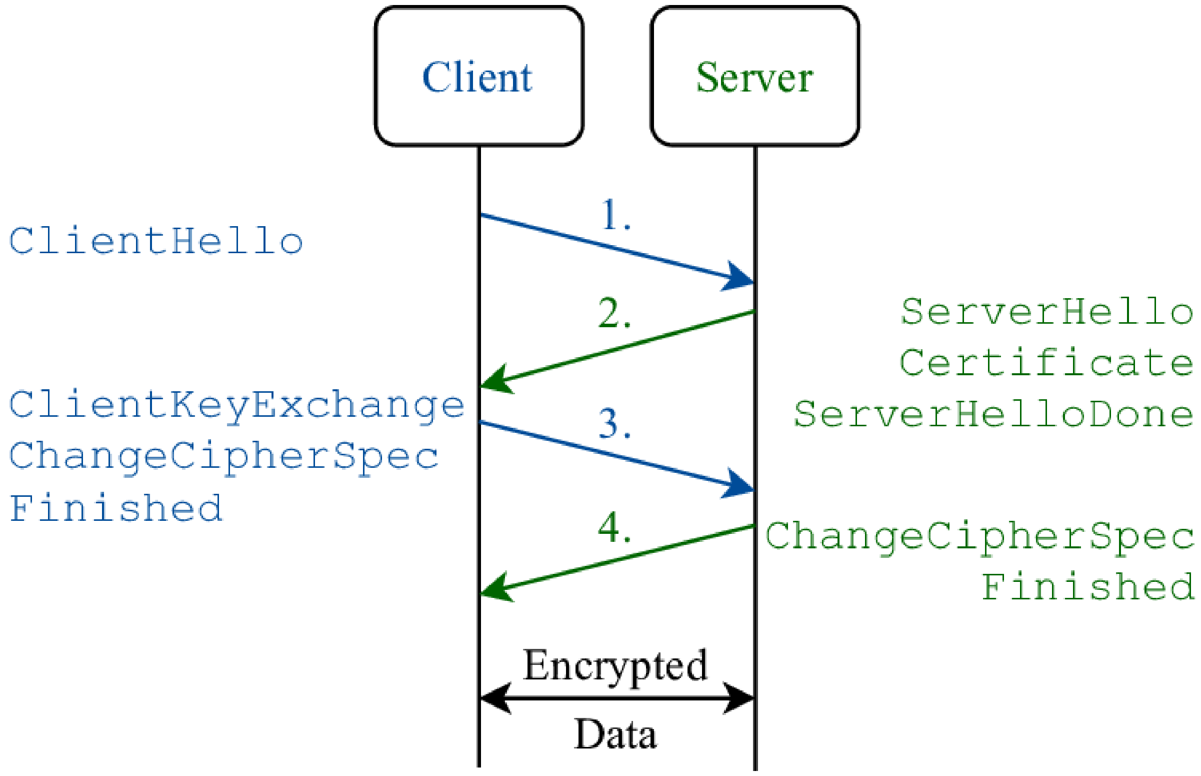


Figure 2 - TLS 1.2 Handshake - https://www.researchgate.net/figure/A-typical-TLS-12-handshake_fig1_352212988

Specifically:

1. **Client sends a "Client Hello" message**
   This contains the highest possible TLS version for the client, client random data (to be later used for premaster-secret),a list of available cipher suites, list of compression methods and a list of TLS extensions.
   Optionally, it can also contain a session id
2. **Server responds with "Server Hello" message**
   This contains the chosen protocol version, server random data(for premaster-secret), selected cipher suite, selected compression (if any) and a list of TLS extensions
3. **Server also sends the Server Certificate** message, containing the certificate, hostname of the server, the public

key belonging to the certificate, usually together with the certificate chain.
This message could be omitted if the chosen authentication method does not require a certificate

- **Server Key Exchange Generation**
  Based on the chosen cipher suite, the server generates a keypair of a private-and a public key, using for example the ECDHE method.

4. **Server sends a Key Exchange message**
   Which contains the just generated public key using ECDHE.
5. **Server sends a “Server Hello Done” message**
   To indicate it finished its first part of the handshake.

- **Client Key Exchange Generation**
  The client generates a keypair in the same manner as the server based on the agreed upon ECDHE

6. **Client sends Key Exchange message to server**
7. **Client Encryption Keys Calculation**
   Client creates the “premaster secret” by multiplying his private key and server’s public key. Client encrypts this premaster secret using the server’s certificate public key and sends it to server.

- **Both client and server calculate the “master secret”.**
  Only the server can decrypt the premaster secret because only the server has the private key belonging to the certificate. Combining the client and server random data from hello messages, the string “master secret” and the actual premaster secret. Required keys are derived from the master secret using a pseudorandom function (f.e. HMAC-SHA256, based on the specified HASH function in the suite). The keys are client and server MAC keys, client and server write keys and based on the cipher suite client and server Initiation Vector.

8. **“Client Change Cipher Spec” message**
   Client indicates following messages will be encrypted using the calculated client write key
9. **Client Handshake Finished**
   Client calculates verification data from all handshake records (Change Cipher Spec is ignored), encrypts with client write key and sends to server. Server can decrypt using the client write key and verify the data
10. **Server Change Cipher Spec**
    Server also indicates following messages will be encrypted using the server write key
11. **Server Handshake Finished**
    Server also calculates verification data from all handshake records (including the extra record from 9.), encrypts using server write key and sends to client.
12. **Client decrypts and verifies data, the application data transmission can begin**

5 TLS 1.3

5.1 Supported cipher suites

RFC 8446 specifies only 5 cipher suites for use with TLS 1.3. They are:

- TLS_AES_128_GCM_SHA256 (Recommended)
- TLS_AES_256_GCM_SHA384 (Recommended)
- TLS_CHACHA20_POLY1305_SHA256 (Recommended)
- TLS_AES_128_CCM_SHA256 (Strong)
- TLS_AES_128_CCM_8_SHA256 (Strong)

(Ciphersuite.info - TLS 1.3)

The ciphersuite.info also mentions 2 cipher suites with SM4 encryption algorithm, however this is of Chinese origin and its security is unknown, so any use is not recommended.

5.2 Changes from TLS 1.2

Overall, the TLS 1.3 update focuses more on simplicity and security than backwards compatibility. The cipher suites are missing Key Exchange and Authentication info. From TLS 1.3, cipher suites only contain information for the bulk encryption algorithm and the hash function. The choice of a key exchange mechanism and an authentication mechanism are chosen separately and independently of each other and the cipher suite. We can see that supported algorithms are AES 128 or 256 and CHACHA20.

For authentication, TLS 1.3 uses only forward secrecy options:

- RSA authentication
- ECDSA authentication (elliptic curve Digital Signature Algorithm)
- EdDSA (Edwards-Curve Digital Signature Algorithm)
- Symmetric pre-shared key authentication

(Ciphersuite.info - TLS 1.3, n.d.)

For key exchange, TLS 1.3 supports 3 modes:

- (EC) DHE (Ephemeral Diffie Helman over finite fields or elliptical curves)
- PSK-only (Pre-Shared Key)
- PSK with (EC) DHE

TLS 1.3 also cancels custom DH groups and mandates using only “supported groups” for key exchange.

(Ciphersuite.info - TLS 1.3, n.d.)

5.3 Handshake

The TLS 1.3 handshake is majorly different from the previous TLS version. The TLS1.2 handshake requires at least 2 round trips (2RTT). A RTT (round trip time) is the time it takes for a network request to go from the starting point to the destination and back again. (Cloudflare, What is round-trip time? | RTT definition)

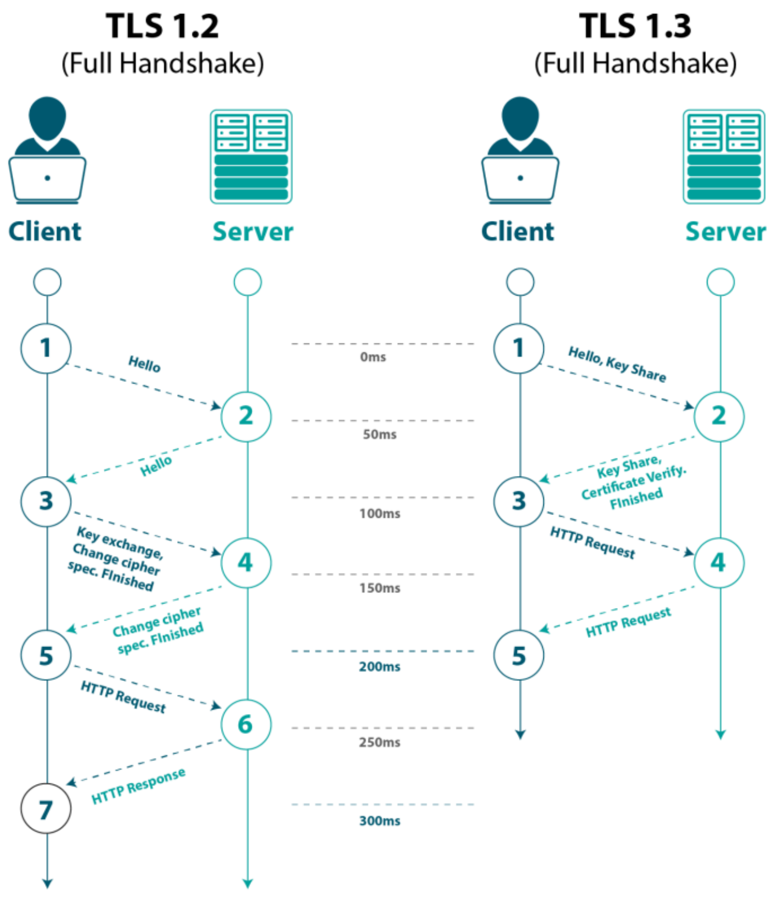


Figure 3- Handshake comparison

With TLS 1.3, the handshake can be completed within 1RTT. This means that everything after the Client Hello message and Server Hello message is already encrypted. (Ciphersuite.info - TLS 1.3, n.d.)

5.3.1 0RTT

One of the main features of TLS 1.3 is the ability to establish the TLS connection within 0RTT (not including the DNS and TCP round trips before TLS can start). This is especially important for HTTP connections.

(Cloudflare, Cloudflare - 0RTT)

The 0 round trip time is achieved thanks to the Pre-Saved Key (PSK) mode for repeated/renewed connections. This comes at a cost of some security properties, specifically weaker anti-replay defense.

6 Conclusion

The goal of this project was to get a better understanding of the Transport Layer Security protocol, its working principles as well and existing versions including their support. The most important part was to research the options of symmetrical and asymmetrical ciphers, authentication methods, known as cipher suites and to find out which are secure, and which are weak or insecure. This report contains most of the findings and researched principles.

7 References

Barnes, R., Thomson, M., Pironti, A., & Langley, A. (2015, June). *RFC 7568 - SSL 3.0 deprecated*. Retrieved from https://datatracker.ietf.org/doc/html/rfc7568

Ciphersuite info - TLS_DH_RSA_WITH_AES_128_CBC_SHA256. (n.d.). Retrieved from https://ciphersuite.info/cs/TLS_DH_RSA_WITH_AES_128_CBC_SHA256/

Ciphersuite.info - TLS 1.2. (n.d.). Retrieved from https://ciphersuite.info/cs/?sort=sec-desc&singlepage=true&tls=tls12

Ciphersuite.info - TLS 1.3. (n.d.). Retrieved from https://ciphersuite.info/cs/?tls=xtls13&sort=sec-desc&singlepage=true

Cloudflare. (n.d.). *Cloudflare - 0RTT*. Retrieved from https://blog.cloudflare.com/introducing-0-rtt/

Cloudflare. (n.d.). *Cloudflare - TLS handshake*. Retrieved from https://www.cloudflare.com/en-gb/learning/ssl/what-happens-in-a-tls-handshake/

Cloudflare. (n.d.). *What is round-trip time? | RTT definition*. Retrieved from https://www.cloudflare.com/en-gb/learning/cdn/glossary/round-trip-time-rtt/

D. Harkins, E. (2019, February). *RFC 8492 - TLS-PWD*. Retrieved from https://datatracker.ietf.org/doc/html/rfc8492

Dierks, T., & Allen, C. (1999, January). *RFC 2246 - TLS 1.0*. Retrieved from https://datatracker.ietf.org/doc/html/rfc2246

Dierks, T., & Rescorla, E. (2008, August). *RFC 5246 - TLS 2.0*. Retrieved from https://datatracker.ietf.org/doc/html/rfc5246

McKay, K. A., & Cooper, D. A. (2019, August). *Guidelines for the Selection, Configuration and Use of Transport Layer Security (TLS) Implementations*. Retrieved from https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r2.pdf

Moriarty, K., & Farrel, S. (2021, March). *RFC 8996 - Deprecating TLS 1.0 and 1.1*. Retrieved from https://datatracker.ietf.org/doc/html/rfc8996

Popov, A. (2015, February). *RFC 7465 - RC4 deprecated*. Retrieved from https://datatracker.ietf.org/doc/html/rfc7465

Qualys. (2019, April). *Zombie POODLE and GOLDENPOODLE Vulnerabilities*. Retrieved from https://blog.qualys.com/product-tech/2019/04/22/zombie-poodle-and-goldendoodle-vulnerabilities#comment-303228

Rescorla, E. (2018, August). *RFC 8446 - TLS 3.0*. Retrieved from https://datatracker.ietf.org/doc/html/rfc8446

SHAttered. (n.d.). Retrieved from https://shattered.io/

Turner, S., & Polk, T. (2011, March). *RFC 6176 - SSL 2.0 deprecated*. Retrieved from https://datatracker.ietf.org/doc/html/rfc6176