Nationaal Cyber Security Centrum
Ministerie van Justitie en Veiligheid

Transport Layer Security (TLS)

Security guidelines version 2025-05

Introduction

Transport Layer Security (TLS) is a standardised protocol for the setup and use of an encrypted connection between two computer systems or applications: a client and a server. The secure connection guarantees the confidentiality and integrity of the data exchanged between the client and the server. TLS enables you to safely browse the internet, exchange confidential e-mails, and to connect to your digital workplace remotely by means of a TLS-based VPN solution. TLS is also used to secure OT networks and to let the systems and applications communicate securely among themselves within your own network environment.

TLS, the successor to SSL (Secure Sockets Layer), and in some contexts still referred to as SSL, has been developed and advanced for more than 30 years and allows for specific options and settings to be configured. You can, for example, choose which TLS versions you wish to support, which algorithms you wish to use for bulk encryption and/or whether you want to apply TLS compression. This adaptability allows you to use TLS for your application in an appropriate manner. Unfortunately, not all options are equally safe.

This publication offers recommendations on how to set up a TLS configuration that protects your application in an appropriate manner. To this end, the next sections present you with:

- Background information relevant in the context of this publication. Among other things, this chapter describes the general operation of TLS, the options and settings relevant to realise a secure TLS connection, and frequently-used terms (Section 1).
- A step-by-step plan you can use to arrive at a secure TLS configuration (Section 2).
- The TLS settings for a secure TLS configuration (Section 3), and
- The preconditions you must meet for an overall secure operation of TLS (Section 4).

The security guidelines for TLS were first published by the NCSC in 2014 and have since been kept up to date. Are you already well acquainted with the previous version of these guidelines, dating back to 2021, and are you interested in what has changed? In *Appendix A: Change with respect to the previous version*, you will find the main changes. We furthermore refer you to *Appendix B: List of cipher suites*, for a list with frequently-used cipher suites which you can use as part of your TLS configuration.

Constituents

These guidelines are written for security professionals who play a role in the configuration or management of a TLS implementation and seek tools to secure this in an appropriate manner.

This publication has been drawn up in association with the following organisations:

- General Intelligence and Security Service (AIVD)
- The Ministry of Health, Welfare and Sport
- Standardization Forum, and
- Cryptography in Context.

We would also like to thank the following individuals and organisations for their critical views and valuable contributions to this revised publication:

- ABN AMRO
- Arne Padmos, Adyen
- ASN Bank
- Dutch DPA
- Centric
- Cloudflare
- KPN
- Logius
- Ministry of General Affairs
- Dutch Banking Association (NVB)
- Northwave Cyber Security
- Pi Rogaar, independent expert
- Peter-Jan Kortlever, KNAB
- SURF
- The Information Security Service for Municipalities (IBD)
- Z-CERT

Table of contents

1 Background

This section provides a general description of the TLS protocol and the main settings, options and terms relevant from a security perspective. Use this section as you see fit to understand the recommendation in the rest of this publication.

1.1 What is TLS?

Transport Layer Security (TLS) is a protocol to setup a secure connection between two computer systems or applications: a *client* and a *server*. The secure connection guarantees the confidentiality and integrity of the data exchanged between the client and the server. The protocol is used for many well-known applications, including the protection of web traffic via https (i.e. the 'browser padlock icon'), secure e-mail traffic (IMAP and SMTP after STARTTLS) and certain types of Virtual Private Networks (VPN). A connection between a client and a server protected by TLS is called a *TLS session*.

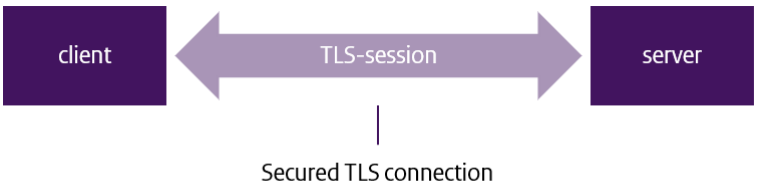

Figure 1: Illustration of a TLS session

A TLS session is established only if the client and the server support corresponding settings. As soon as the client initiates a connection with the server, a TLS session starts in the *handshake phase*. During this phase, the client and the server can authenticate one another and coordinate the settings to be used during the TLS session. The selected settings are then used in the *application phase* to securely exchange data. The client and the server need not start a new TLS session for each connection attempt. Indeed, TLS offers the possibility of *resuming* a TLS session. Should a client wish to a resume session, this can be done by means of a new handshake, after which the application phase of the TLS session is resumed with the previously chosen settings.

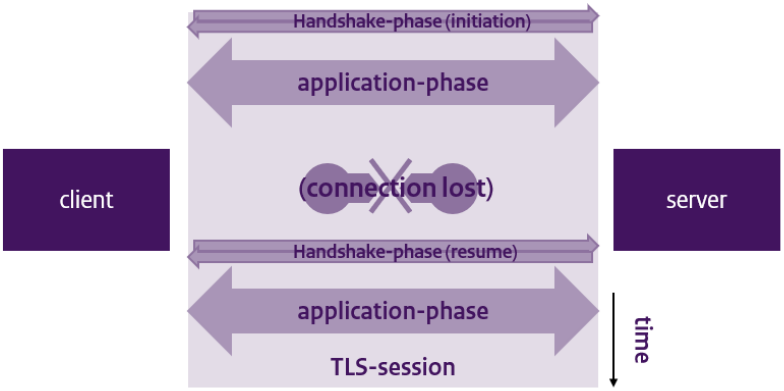

Figure 2: Illustration of the different phases in a TLS session.

1.1.1 Handshake phase

In the handshake phase, the client and the server *set up a new TLS session* or *resume a previously set-up TLS session.*

a) **New TLS session:** If a TLS session has yet to be set up between the client and the server, or if there is the need to start a new session, for example because a previous session has expired, the client and the server exchange multiple messages to agree on the settings to be used during the TLS session. A number of important elements from this messaging, which will be further explained later on in this section, are:

   a. **Authentication:** In setting up a new connection, the server and/or client can authenticate one another to be certain that they are communicating with the intended party.

   b. **Alignment:** The client and the server agree on the settings they intend to use during the application phase. This alignment is done roughly as follows:

      1) The client sends a list with settings it supports and indicates the preferred settings.
      2) If the client and the server support the same settings, the server then chooses the set of settings to be used during the TLS session and informs the client about the decision.

   c. **Key exchange:** The client and the server jointly determine a secret key which is to be used during the application phase.

   Once the above is completed successfully, a TLS session is established and the application phase can begin. The settings used are stored and linked to a *session ID* or *session ticket*, to resume the session at a later date.

b) **Resuming the TLS session:** In many applications, it is common for the client and the server to make multiple connections or renew connections once an initial TLS connection is established. If the client want to resume a session, it will send the previously mentioned *session ID* or *session ticket* along with the handshake. If the session is still valid, meaning that the settings are still supported and the session has not had a timeout yet, this session is resumed with the previously aligned settings. The handshake phase will then run more quickly as it requires less alignment.

1.1.2 Application phase

During the application phase, the settings and the secret key, which had been agreed on during the handshake phase, are used to exchange data between the client and the server in a secure manner. The client and the server encrypt their data with an encryption algorithm, referred to in this publication as *bulk encryption*, and the exchanged secret key.

1.2 Settings

The TLS protocol has been developed and enhanced for more than 30 years now. Different TLS versions were released in those years. Newer versions tend to be more secure, but also stand out because they are faster and support different settings. Below, we describe the settings relevant to the security rating of a TLS session. For each of these settings, Section 3 indicates the extent to which they lend themselves to the setup of a secure TLS configuration.

1.2.1 Versions

New versions of the TLS protocol are released in the event of significant developments in order to safeguard the compatibility and the interoperability between the client and the server. There are currently seven different TLS versions:

- Secure Sockets Layer (SSL) 1.0, 2.0 and 3.0: SSL has been developed by Netscape Communications Corporation and is the forerunner of TLS.
- TLS 1.0, 1.1, 1.2 and 1.3: Developed by the Internet Engineering Task Force (IETF). The IETF offers TLS as an open standard.

The most recent TLS version is TLS 1.3 [1]. The differences with previous versions of TLS are explained in more detail later on in this section, but put in brief, TLS 1.3 is characterised by:

- Mandatory server authentication;
- The use of a more efficient handshake that is run more quickly;
- Only supporting algorithms that are considered safe by current standards;
- Another definition of the word *cipher suite;*
- Dropping the options 'TLS compression' and 'renegotiation'; and
- Adding the 'Zero Round-Trip Time (0-RTT)' option.

1.2.2 Cryptographic algorithms

The security of a TLS session largely bases itself on the use of a number of cryptographic algorithms. These algorithms in TLS are used for the purpose of:

- **Authentication** (handshake phase): checking the digital identity of the server and/or client;
- **Key exchange** (handshake phase): exchanging a secret key that will be used during the application phase for bulk encryption;
- **Bulk encryption** (application phase): encrypting data during a TLS session; and
- **Hashing** (handshake phase and application phase): ensuring the integrity and authenticity of the data exchanged between the client and the server.

1.2.2.1 *Authentication*

Authentication means checking the identity of the server and/or the client. Depending on the TLS version in use, it is an optional feature for the client and/or the server to authenticate themselves:

- In TLS 1.2 and previous versions, it is optional for both the client and the server to authenticate themselves.
- In TLS 1.3, the server always has to authenticate itself; the client's authentication is optional.

Given the fact that client authentication and server authentication is performed in the same way in TLS, the explanation below does not refer to client and server. Instead, we describe how a *party seeking authentication* authenticates itself to a *verifying party*, which is the party responsible for verifying the identity of the other party.

Authentication in TLS virtually always takes place on the basis of digital (X.509) certificates[1]. This section provides background information relevant to understanding the settings that are inherently part of TLS: the optional use of OCSP stapling and the algorithm for TLS authentication. The management and use of certificates in a broader sense is a complex and more expansive topic that falls outside the scope of this publication.[2]

In addition to authentication via digital certificates, it is also possible to authenticate the client and the server out-of-session by means of a *pre-shared key*. More information about this can be found in Section 1.4.4.

To make use of authentication with certificates, the party seeking authentication must first request a certificate authority (CA) to generate a certificate. This certificate contains different data relevant within the context of authentication, including the public name of the party seeking authentication and its private and public keys. The CA signs this certificate by means of a digital signature to ensure its integrity and authenticity. Reliable authentication requires that both the verifying party and the party seeking authentication trust the CA.

---

[1] *X.509 is a standard that specifies the format of public digital certificates [25].*

[2] *More information on the use and management of certificates is provided in the NCSC publication Zorgeloos certificaatbeheer (Factsheet on secure management of digital certificates, in Dutch). This fact sheet is outdated at the time of writing and is being revised. The general guidelines in this publication can still be used at present.*

At the time of the handshake, the party seeking authentication sends the certificate to the verifying party. Authentication subsequently takes place based on the following steps:

1) **Certificate check:** The verifying party checks the certificate submitted to it by:
   a. Checking the **digital signature** of the certificate. This check offers the verifying party certainty that the CA issued this certificate and that the certificate is authentic.
   b. Checking to see whether the certificate is still valid and that it has not been prematurely **revoked**. The verifying party can do this by means of:

      1. **The Certificate Revocation List (CRL):** Each CA keeps a list of certificates that have been revoked (the CRL). The verifying party periodically downloads the CRL and checks whether or not the certificate is included in this.

      2. **Online Certificate Status Protocol (OCSP):** The verifying party requests the certificate's status from an *OCSP responder*, a service often provided by a CA. The OCSP responder informs the verifying party of the revocation status of the requested certificate.

      3. **OCSP stapling:** OCSP stapling means that the party seeking authentication requests the status of its own certificate from an OCSP responder. The OCSP responder replies to the request with a digitally signed revocation status. The party *seeking authentication* can send this *stapled* OCSP response to the verifying party along with the certificate during the handshake phase. The verifying party uses this to verify the status of the certificate.

      In TLS, the party seeking authentication has the option of applying OCSP stapling. The verifying party is free to decide how it wants to verify the certificate's revocation status.

2) **TLS authentication:** The verifying party and the party seeking authentication make use of a digital signature during the handshake phase to complete the authentication process. In this process, the public key, which is part of the digital certificate, is used to obtain assurances that the verifying party is communicating with the intended party seeking authentication.

1.2.2.2 *Key exchange*

The algorithm for the key exchange specifies the manner in which the client and the server agree to the key that is to be used for bulk encryption.

The term *forward secrecy* is relevant for key exchange. Some algorithms use the server's public key to agree upon the session key. This key is part of the certificate and is usually not changed very often. Should an attacker get a hold of the server's secret *private key* which is linked to this public key, it would allow the attacker to decrypt and manipulate *all* current and past communication between the server and its clients. Algorithms offering *forward secrecy* do not run this risk as they make use of a temporary, *ephemeral* key that only applies to that session.

1.2.2.3 *Bulk encryption*

Bulk encryption forms the heart of the application phase. Bulk encryption encrypts the data by means of a symmetrical encryption algorithm, i.e. *cipher*. This algorithm encrypts the data via the previously exchanged key. Virtually all bulk encryption algorithms in this publication are *block ciphers*. These are mostly described in TLS in the form of:

```
Cipher-key length-mode of operation
```

In which:

- `Cipher`: Is the name of the encryption algorithm, for example AES or CAMELLIA
- `Key length`: The length of the key used by the cipher, for example 128-bit or 256-bit.
- `Mode of operation`: The mode in which the block cipher is used, for example CBC (*cipher-block chaining*), CCM (*counter with cipher block chaining message authentication code*) and GCM (*Galois/Counter Mode*).

Block ciphers in CCM and GCM mode, as well as the ChaCha20-Poly1305 algorithm, are *AEAD* algorithms (*authenticated-encryption with associated data*). These algorithms ensure the confidentiality, integrity and authenticity of the encrypted data. Block ciphers in CBC mode only ensure the confidentiality of data. They make use of a hash function, specifically an *HMAC, hash-based message authentication code,* to also ensure integrity and authenticity.

1.2.2.4 *Hashing*

A cryptographic hash function is a mathematical feature that calculates a unique digital fingerprint based on a given input. Hash functions are one-way functions, meaning the input data is not traceable from the computed digital fingerprint. Hash functions are used in a variety of ways in TLS:
As part of the signature and check of digital certificates;

1) As part of TLS authentication based on digital certificates;
2) Generating pseudo-random numbers (PRNG), for example for the key derivation function (KDF), to derive secret keys from a source value;
3) As integrity protection for messages that are being exchanged during the handshake phase, and
4) As integrity protection during the application phase if use is made of HMAC.

1.2.3 Cipher suite

A combination of the algorithms that can be used during a TLS session is called a *cipher suite*. Cipher suites are used both in terms of the client's configuration and the server's configuration as well as during the handshake phase, for the purpose of alignment. Each cipher suite describes one combination of algorithms according to a standardised format. TLS 1.3 applies a different definition and format of a cipher suite compared to older versions. Given the fact that the recommendation in this publication comprises both TLS 1.3 as well as older versions, we describe both definitions and formats below.

Cipher suites prior to TLS 1.3

Up until TLS 1.2, a cipher suite, as described earlier, consisted of the algorithms for key exchange, authentication, bulk encryption and hashing. The figure below illustrates the notation of cipher suites prior to TLS 1.3

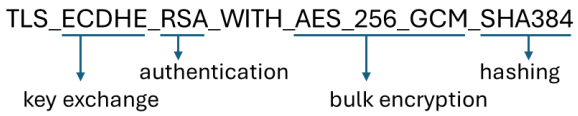

Figure 3: Example of a cipher suite in TLS versions prior to TLS 1.3

The adjustment in the handshake in TLS 1.3 also resulted in a change in the definition of cipher suite. In TLS 1.3, a cipher suite only describes the algorithms for bulk encryption and hashing. The figure below illustrates the notation of cipher suites as used in TLS 1.3.

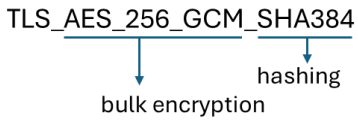

Figure 4: Example of a cipher suite in TLS 1.3

1.3 Other settings

In addition to the choice for cryptographic algorithms, there are also a number of other TLS options relevant from a security perspective. These are briefly further explained below.

1.3.1 Compression

Compression is a technique by which information is represented with less data, leading to less (communication) overhead. An application can implement compression itself, or make use of TLS compression. During the handshake, clients may indicate whether they prefer TLS compression; however, they must always support TLS without compression. Compression may have an adverse effect on the security of a TLS session [2] [3]. Little use is made of TLS compression in practice. Support for TLS compression is even completely disabled in TLS 1.3.

1.3.2 Renegotiation

Renegotiation means that the client and the server start a *new handshake during the application phase*. For example, the client might prefer to use a different bulk encryption algorithm, or wish to restart a session which is about to expire.

Older versions of renegotiation contained vulnerabilities, which are resolved by applying a *renegotiation indication extension* [4]. Renegotiation has not been supported since TLS 1.3.

1.3.3 Resuming sessions (session ID, session tickets and 0-RTT)

As previously described, the client and the server can resume an already established TLS session during the handshake phase. In TLS versions up to TLS 1.2, resuming sessions was supported by making use of session IDs or session tickets [5]. In TLS 1.3, it is only possible to make use of session tickets.

- **Session identifiers (session IDs):** During the setup of a TLS session, the client and the server save the settings chosen by them. They refer to these settings with a session ID.

  If the client wants to resume a session, it will send the session ID along with the handshake. The server and the client use this session ID to fetch the associated settings from their storage and resume the session.

- **Session ticket:** Using session IDs has the disadvantage that a server has to store the chosen settings of *all* valid TLS sessions. Session tickets can be used to avoid this. The server creates a session ticket as soon as the settings for the session have been chosen. This ticket is effectively the list of chosen settings, encrypted by the server. The server sends this ticket to the client at the end of the handshake. If the client wants to resume the session, they will send the session ticket along with a next handshake. The server decrypts the session ticket and resumes the session with the settings described in it.

***Zero Round-Trip Time* (0-RTT)** is an option that was introduced in TLS 1.3. The purpose of 0-RTT is to let the client and the server resume their session more quickly. When using 0-RTT, a section of application data is sent along by the client as early as the handshake phase, i.e. *before* the application phase has started. The use of 0-RTT comes with security risks. It does not offer any protection against replay attacks on the TLS layer and secure use of 0-RTT depends on the application using it.

1.4 Other relevant terms

The previous sections described the main settings relevant within the context of a secure TLS configuration. This section describes a final set of terms relevant to follow the recommendation from this publication.

1.4.1 TLS library

TLS is used in many different applications. Programming all features of TLS is a lot of work and requires specialist knowledge, which is why most software make use of a TLS software library containing an implementation of the TLS protocol.

There are a few dozen TLS libraries available. For some libraries, there is a source code available, while others are made available as a closed product. They can be integrated in control systems, but can also be supplied separately. Not all libraries implement all TLS settings available or are equally well maintained by their developer. OpenSSL may well be the most commonly used TLS library.

By making use of a TLS library, you avoid making mistakes in a TLS implementation. However, please beware that all software contain bugs, including TLS libraries. In turn, bugs can result in vulnerabilities[3]. The use of a TLS library does not guarantee a TLS implementation without vulnerabilities.

1.4.2 Unique properties and random numbers

Unique and cryptographically-secure or pseudo-cryptographically secure random numbers play an important role in many cryptographic applications, including TLS. All operating systems and TLS libraries contain methods to generate unique and cryptographically-secure random numbers. In addition, there are hardware modules to generate such random numbers.

1.4.3 TLS proxy

So far, we described TLS as a protocol that ensures a secure connection between the client and the server. In practice, a TLS session can also be set up between a client and a *proxy:* a system which is located in between the client and the server, as outlined in Figure 5. Optionally, a secure connection is also set up between a proxy and a server.

A proxy can be used for different purposes. For instance, organisations can inspect all incoming network traffic at a single location, the proxy, for malware (TLS inspection). A proxy can also be part of anti-DDoS services and intrusion detection and

[3] *One example is vulnerability 'Heartbleed Bug' in OpenSSL software, which had been resolved in 2014. Also see* https://heartbleed.com/

prevention systems (IDS/IPS). In addition, TLS proxies can be used to shield off older systems from the outside world and to indirectly provide them with a TLS connection.

There are several security risks associated with the use of a TLS proxy. The large volumes of (unencrypted) network traffic processed by a TLS proxy make it an attractive target for digital attacks. An attacker who gains access to the proxy can view sensitive information, such as financial data, personal data, and passwords. A TLS proxy can also be used for attacker-in-the-middle attacks. Additionally, a proxy prevents the use of certificate pinning, a security mechanism whereby a client establishes a TLS connection only with a specific server, and makes it impossible for the user to verify the certificate holder and related organization (in the case of OV, EV, or QWAC certificates)[4].

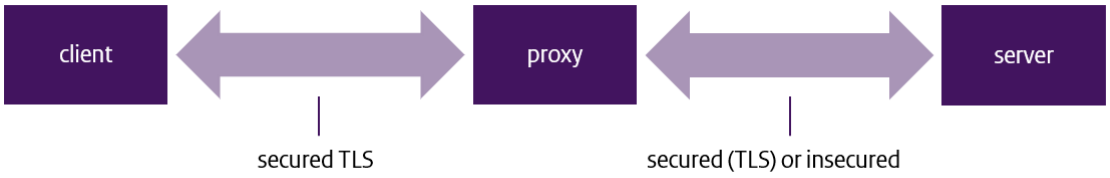

Figure 5: A TLS proxy.

1.4.4 Pre-shared key (PSK)

In the description so far, the client and the server start their communication from an initial handshake. During this handshake, they use different algorithms and secret parameters to authenticate one another and to agree on the way in which they will communicate with each other.

TLS also offers the possibility of making use of an out-of-band established *pre-shared key* (PSK). In this case, specific clients and the server are authenticated in an alternative manner and/or provided with a key for bulk encryption. For example, the secret key for bulk encryption may be exchanged up front between the client and the server by making use of a USB stick containing the key and a courier, which brings the USB stick from the server to the client.

An *out-of-band* PSK means that part of the handshake effectively takes place outside the TLS session. The use of an *out-of-band* PSK has its own challenges and risks and is subject to different preconditions for secure use. Consider, for example, the requirements that have to be imposed on the way in which key material is transported and exchanged between the server and the client. An *out-of-band* PSK can only be used for specific applications and is not scalable for general use.

Session tickets, as discussed in Section 1.3.3, are defined as part of a PSK in the TLS 1.3 handshake. In this case, it is an *in-band* PSK that has been sent during the TLS session. This does lend itself to a wider set of applications.

1.4.5 Cryptographic strength

The *cryptographic strength* is a measure used to express the security rating of a cryptographic algorithm. Cryptographic strength is expressed in bits. Roughly speaking, a cryptographic strength of 128 bits means that an attacker approximately needs 2^{128} (2 to the power of 128) attempts to successfully decipher encrypted information. The cryptographic strength is an estimate based on the (type of) algorithm, the key length used, and the effect of known vulnerabilities in algorithm. The strength of an encryption algorithm is *not* always the same as the key length that uses an algorithm. Different algorithms can achieve the same cryptographic strength with different key lengths.

A cryptographically-relevant quantum computer (CRQC) is a quantum computing system powerful enough to seriously reduce the cryptographic strength of some algorithms. For practical reasons, this publication only expresses the cryptographic strength as determined for conventional computer technology. However, we do include the effect of a CRQC on this cryptographic strength in the recommendation. More information on a CRQC and the quantum threat has been added later on in this publication.

[4] *Consult the NCSC publication "Zorgeloos certificaatbeheer" for more information about EV certificates.*

2 Step-by-step plan: How do I arrive at an appropriate TLS configuration?

This section offers a general step-by-step plan to achieve an appropriate, secure TLS configuration.

The guidelines in this publication are primarily meant to help you arrive at the safest possible TLS configuration. In practical terms, security is not the only consideration when choosing a configuration. Your application, for example, may also need to be accessible for outdated clients that do not offer support for the latest and safer versions of TLS. The choice of an appropriate TLS configuration is therefore also a business consideration. For this reason, go through the steps in this step-by-step plan together with whoever is responsible for the application for which the TLS configuration is drawn up. This is usually the product manager or product owner.

1. **Inventory.** Start your step-by-step plan by gaining insight into the possibilities of your clients and servers. This will help you ascertain which settings you can and need to support for your application.
   Ask yourself the following questions:

   a. Will your server be making use of existing applications or systems? If so, identify which TLS settings are supported by this.
      Will a new application or system be purchased for the server? If so, make sure that this supports appropriate and secure TLS settings.

   b. Do you fully understand the clients that are to support your TLS configuration?
      i. Outline a general profile of the target audience together with the product manager. Does the application, for example, have to be accessible for a wide group of end users? Or is the application intended for a more selective group of clients, such as systems on your in-house business network?

      ii. Make an assessment of the clients and the types of clients that the target audience will use. Among other things, identify:

         1. From which devices your application needs to be accessible; for example, a system on the in-house company network or a mobile phone.
         2. Which applications your target audience will use, for example a general web browser or an in-house application; and
         3. To what extent your configuration has to offer specific support for certain operating systems.

      Also take into consideration how old a client may be. Bear in mind that a broad target audience, such as citizens, will typically make use of a various set of clients, which in some cases could also be significantly outdated.

      iii. Determine the settings that are supported by the previously identified types of clients. This is easy to look up for most common devices, operating systems, browsers and TLS libraries.

2. **Selection:** Choose the settings for your TLS configuration. Make use of:

   a. The results of step one, which provide an overview of the settings you can and need to support for your application.
   b. The recommendation involving secure TLS settings and preconditions from Sections 3 and 4.

   Limit your configuration to secure settings, corresponding with security levels **Sufficient** or **Good** as indicated in Section 4. Given the results of steps 1-3, does your configuration need to support settings that are less secure? For example, does your server only use outdated TLS versions, or does your application need to be accessible to clients that only support older, and thus unsecure, settings?

   In that case, discuss the following with the product manager:
   - The extent to which an adjustment/update of a server could lead to a more secure configuration;
   - The extent to which your organisation is capable of adjusting the clients, for example, by means of software updates;
   - The extent to which it is really necessary to support clients that only support less secure settings, or the extent to which security risks outweigh accessibility; and
   - Whether there are any other (additional) security measures to mitigate the risks.

Where possible, gain specialist knowledge and experience to formulate a well-founded response to this. Even if the need were to remain to support less secure settings, never consider this to be a reason to not use TLS altogether. Even a less secure TLS configuration may be impenetrable for some attackers. Document the deviations and the results of the above trade-offs and have them endorsed by the product manager.

3. **Configuration:** Configure those TLS settings you wish to support. The TLS configuration is part of the software that uses the TLS connection. For example, if you want to offer https, then configure TLS as a component of the webserver software.

   During the handshake phase, a client sends a list of supporting settings to the server, including their preferred settings. This will allow clients to express their preference to use certain bulk encryption algorithms, for example, because the client is running on hardware that properly supports such an algorithm. The server eventually selects the settings to be used for the TLS session. Configure the server in such a way that:
   - This may prefer settings that realise a connection as secure as possible for your application (see Section 3.2);
   - From this selection, choose the preferred settings from the client's point of view.

   In addition to the TLS settings, configure the certificate which your server uses for authentication purposes. If your certificate is not directly signed by the root CA, then also configure the intermediate certificates used to authenticate the path between the root CA and your certificate[2]. Also, consider configuring TLSA records to support the DANE protocol.

4. **Check:** Check the extent to which the TLS configuration has been correctly configured. Settings may have been accidentally configured that are not acceptable to you, or certain settings may not have been configured. Make use of tools and websites that allow you to run such checks, including the way in which these relate to the security levels of this publication.[5]

5. **(Public) testing:** Test if the server works correctly with the different intended clients. If your server is not properly accessible for some clients, revisit the previous steps and consider how they can be resolved.
   Ask yourself the following questions:
   - Have all selected TLS settings been configured on the server and clients?
   - Does the configuration have an acceptable impact on the application's performance?
   - Can any problems be solved by supporting additional, secure, settings?
   - Is it possible to have the client support additional, secure settings, for example by means of a software update?

In addition to a correct setting, the security of a TLS connection is subject to a correct implementation. Regularly test the implementation and make it part of your regular management process, such as vulnerability management, to monitor and manage your TLS configurations.

[5] *Examples of such tools are testssl.sh (https://testssl.sh/) and sslyze (https://github.com/nabla-c0d3/sslyze). At https://www.internet.nl/ you can test the extent to which your web servers and e-mail servers comply with the guidelines as mentioned in this publication. Qualys SSL labs allows you to run a similar check (https://www.ssllabs.com/ssltest/).*

3 Secure TLS settings

A secure TLS session requires your server to be configured with secure settings. This section describes those TLS settings that are considered secure or less secure. We make use of four security levels: **Good, Sufficient, To be phased out** and **Insufficient**.

Below, we first describe the different security levels and how you can use them to realize a secure TLS configuration. This is followed by a description of the security level of each setting.

3.1 Security levels used in this document

In this publication, we use four security levels to express the security rating of TLS settings:

- **Good:** The label **Good** is awarded to settings considered most secure and future-proof.
- **Sufficient:** This setting offers **Sufficient** security rating by current standards.
- **To be phased out:** Settings are considered **To be phased out** if they are expected to become **Insufficient** over time, e.g. in light of further developments of attack techniques. Such settings provide only a small margin of security. Phase out the use of these settings.
- **Insufficient:** These are settings that are not secure, for example, because they contain known vulnerabilities that are easy to exploit. Do not use these settings.

The security levels have been established on the basis of expert opinion and through the consultation of relevant analyses, standards and guidelines of renowned institutions and organisations. This also included an estimate of the cryptographic strength if the setting in question can be expressed in these terms. A cryptographic strength of 128 bits is required for achieving a security level of **Sufficient** or **Good**[6]. A setting that has **To Be Phased Out** has a minimum cryptographic strength of 112 bits. Benchmarks used in the security rating of the settings are provided, where applicable.

The security levels in this publication focus on a general, widespread use of TLS and are *application-agnostic*. For example, while some TLS settings cannot be used in a secure manner for public networks, they could be used in a specific, in-house network environment. Such situations are not included in the determination of the security levels.

The words '**Insufficient'**, '**To be phased out'**, '**Sufficient'** and '**Good'** also have a meaning in everyday language. To avoid any confusion, these words are represented in a **bold font** in the guidelines whenever they refer to security levels.

The threat of cryptographically-relevant quantum computers

A cryptographically-relevant quantum computer (CRQC) is a quantum computing system powerful enough to seriously reduce the cryptographic strength of different algorithms. CRQCs are thus a threat to cryptography. In particular, common forms of asymmetric cryptography, which are used for authentication and key exchange purposes, are vulnerable and can be effectively 'cracked' with a CRQC. CRQCs also reduce the cryptographic strength of algorithms for bulk encryption and hashing.

We expect that the first CRQC will be realized between 2030 and 2040. That does not mean that your organisation is not currently at risk. Attackers could, for instance, *launch a store-now-decrypt-later* attack to collect your TLS-encrypted data now and decrypt them at a later date, e.g. when they have a CRQC. History also shows that any migration to more secure algorithms involves complex challenges that require a lot of time, planning and preparation.

In this publication, we provide guidance on how to take into account the threat posed by CRQCs. Settings known to be vulnerable for attacks with a CQRC are not future-proof and will not be awarded higher than security level **Sufficient** in this publication. For practical reasons, we only express the cryptographic strength as determined for conventional computer technology. However, we do include the effect of a CRQC on this cryptographic strength in the recommendation. Section 4.5 additionally describes a number of actions you could already perform to prepare your TLS configuration for the arrival of CRQCs.

[6] *This is in line with the Czech security authority NÚKIB (128-bit) [13] and the German BSI (120-bit) [11]. The American NIST currently still applies 112-bit, however, it too intends to take this to 128-bit in their renewed guidelines [24].*

3.2 Guideline for a secure TLS configuration

Use the following guidelines to arrive at a secure TLS configuration.

- Give preference to using **Good** settings, supplemented by **Sufficient** settings to support older software if needed.
- Only use settings **To be phased out** where necessary with a view on client-server interoperability. In doing so, formulate clear and measurable criteria for the phase out of these settings, such as 'these settings will be phased out on <date>, <Following the release of <web browser> version X in the 'stable release channel>, or <when the relative number of users is less than Y%>. Have it endorsed by the person in charge of the application for which TLS is being configured.
- Do not make use of **Insufficient** settings.
- Make use of the secure parameters described in order to reach the security level described.
- Also make sure to consider the preconditions for a secure TLS configuration, as set out in Section 4.

If you are not able to follow these guidelines, for example, because your application also needs to be accessible for outdated clients that only support **Insufficient** settings, please see the step-by-step plan in Section 2 for a course of action.

3.3 Secure settings

Use the recommendations from this section to configure secure TLS settings.

3.3.1 Versions

Use TL 1.3. Different vulnerabilities have been resolved in this TLS version, which means a higher security level. In addition, it is expected that quantum-secure cryptography, i.e. cryptography which can resist a CRQC attack, will first and only be standardised for TLS 1.3. By making use of TLS 1.3 now, you ensure a smoother future migration to quantum-safe cryptography. Not all clients offer support for TLS 1.3. Configure the use of TLS 1.2 to support older clients, though give preference to TLS 1.3 where possible [6]. Other versions are not safe to use.

Version	Security level
TLS 1.3	**Good**
TLS 1.2	**Sufficient**
TLS 1.1	**Insufficient**
TLS 1.0	
SSL 3.0	
SSL 2.0	
SSL 1.0	

Table 1: TLS versions and their security status [6].

3.3.2 Authentication

Make use of authentication based on X509 certificates[7].
For TLS authentication, use one of the algorithms from Table 2.

Algorithm	Security level
ECDSA	**Sufficient**
RSA	
EdDSA	
DSS	**Insufficient**
EXPORT-*	
PSK[8]	
Anon	
NULL	
KRB5[8]	

Table 2: Algorithms for TLS authentication and their security level [6] [7].

Consider the following when choosing an algorithm:

- All algorithms in Table 2 are vulnerable to the quantum threat. They offer **Sufficient** protection for now. Quantum-safe alternatives are not yet part of the TLS standards.
- In order to reach the same security level, ECDSA and EdDSA require a shorter (public) key than RSA. This means less data needs to be exchanged during the handshake. The signing process is also many times faster with ECDSA. RSA has a much faster verification process. Furthermore, RSA is slightly better able to withstand the threat of a CRQC by using a longer secret parameter (key length).
- The EdDSA algorithm offers **Sufficient** security; however, it is not as widely supported as ECDSA and RSA. Consider whether the use of EdDSA is really necessary for your application, or if you can make use of ECDSA or RSA.

3.3.2.1 Secure parameters

The security of ECDSA, RSA and EdDSA partly depends on the chosen parameters. Choose the parameters as indicated below in order to achieve the previously indicated security level.

---

[7] *The recommendation in this publication limits itself to those settings that are directly related to a TLS configuration. While the setup of adequate certificate management is an important prerequisite for the secure use of TLS, it falls outside the scope of this publication. See Section 4.4 for more information.*

[8] *The use of a PSK and KRB5 lends itself to niche application only and it requires specialist attention to mitigate security risks. These algorithms are thus* ***Insufficient*** *in their use for general purposes. Read Section 1.4.4 for more information.*

If you use ECDSA for TLS authentication, use a standardised elliptic curve as described in Table 3. Consider the following:

- Larger elliptic curves have greater cryptographic strength, but also have a higher performance overhead of the client and server. Only use stronger curves if necessary for your application.
- In practice, the Brainpool* curves are used significantly less than the other curves. Consider whether the use of these curves is really necessary for your application, or if you can make use of other curves.

Elliptic curve	Security level	Cryptographic strength (bit)
secp521r1	Sufficient	260
secp384r1		192
secp256r1		128
Curve448*		224
Curve25519*		128
brainpoolP512r1		256
brainpoolP384r1		224
brainpoolP256r1		128
secp224r1	To be phased out	112
Other curves	Insufficient	—

Table 3: Elliptic curves for ECDSA for TLS authentication and ECDHE for key exchange, their security level and cryptographic strength. [8] [9] [10] [11].
**Solely to be used for ECDHE.*

If you use RSA for TLS authentication, use a private/public key pair with a modulus according to Table 4. Does your TLS configuration support TLS 1.2? If so, check which padding mechanism(s) your TLS library supports (RSA-PSS or RSA-PKCS#1 v1.5). The more modern RSA-PSS is more secure (and is the only supported padding mechanism in TLS 1.3). Prefer using RSA-PSS and support RSA-PKCS#1 v1.5 only if necessary for your application.

Parameter	Value	Security level	Cryptographic strength (bit)
Key length (modulus)	At least 3072 bit	Sufficient	128+
	2048-3071 bit	To be phased out	112-128
	Less than 2048 bit	Insufficient	<112

Table 4: Parameters for RSA for TLS authentication, their security level and cryptographic strength [12] [11] [13].

If you use EdDSA for TLS authentication, use a standardised Edwards curve such as shown in Table 5.

Elliptic curve	Security level	Cryptographic strength (bit)
Ed448	Sufficient	224
Ed25519		128
Other curves	Insufficient	—

Table 5: Edwards curves for EdDSA for TLS authentication, their security level and cryptographic strength.

3.3.3 Key exchange

Make use of a key exchange algorithm that offers forward secrecy by making use of short-lived (ephemeral) keys. All algorithms with a security level **Good, Sufficient** and **To be phased out** in Table 6 support the use of ephemeral keys.

X25519MLKEM768, SecP256r1MLKEM768 and SecP384r1MLKEM1024 are hybrid, quantum-safe algorithms that use a combination of the algorithms ECDHE and ML-KEM [14]. These algorithms are relatively new and are not *yet* part of the TLS standards. They are, however, subject to an ongoing standardisation process for TLS [15]. Particularly, X25519MLKEM768 is increasingly supported by TLS software libraries and is utilized by most web browsers. Consider the option of already configuring X25519MLKEM768 now in order to be resilient against the threat of a CRQC. Also see Section 4.5.

Algorithm	Security level
X25519MLKEM768	Good
SecP256r1MLKEM768	
SecP384r1MLKEM1024	
ECDHE	Sufficient
DHE	To be phased out
RSA	Insufficient
DH	
ECDH	
KRB5[8]	
NULL	
PSK[8]	
SRP	

Table 6: Algorithms for key exchange and their security level [6].

3.3.3.1 *Secure parameters*

Do you use X25519MLKEM768, SecP256r1MLKEM768 or SecP384r1MLKEM1024 for your configuration? The parameters are inherently part of these algorithms. There is thus no need to make any additional choices.

If you use ECDHE for your configuration, please use the parameters as previously indicated in Table 3.

If your configuration supports DHE, phase this out. In the meantime, use one of the following standardised *finite field groups* (ff). Larger finite field groups have greater cryptographic strength, but also a higher performance overhead. During the phasing-out period, most applications will be adequately protected when using ffdhe3072 and ffdhe4096.

Finite field group	Security level	Cryptographic strength (bit)
ffdhe8192	To be phased out	192
ffdhe6144		175
ffdhe4096		150
ffdhe3072		125
ffdhe2048	Insufficient	103

Table 7: Finite field groups for DHE, their security level and cryptographic strength [6] [16].

3.3.4 Algorithms for bulk encryption

Use an algorithm that effectively integrates authentication and encryption into its mode of operation (AEAD algorithms). These are block ciphers in CCM or GCM mode, or ChaCha20-Poly1305, as shown in Table 8.

Algorithm	Security level	Cryptographic strength (bit)
AES-256-GCM	Good	256
ChaCha20-Poly1305		256
AES-256-CCM	Sufficient	256
AES-128-GCM		128
AES-128-CCM		128
AES-256-CBC	To be phased out	256
AES-128-CBC		128
CAMELLIA*		128/256*
ARIA*		128/256*
3DES	Insufficient	—
SEED		—
AES-256-CCM_8		64†
AES-128-CCM_8		64†
IDEA		—
DES		—
RC4		—
NULL		—

Table 8: Algorithms for bulk encryption, their security level and cryptographic strength [6] [17] [13]. AES algorithms are represented in the format 'cipher name - # - mode of operation', where # refers to the key length.
** The security levels for CAMELLIA and ARIA have been established irrespective of the selected operation mode. The cryptographic strength is identical to the selected key length, i.e. 128-bit or 256-bit.*
† AES-CCM_8 is a variation of AES-CCM with an abbreviated authentication tag. This has no effect on the cryptographic strength from a confidentiality perspective, however, it lowers the integrity protection provided to 64 bits. From this point of view, these algorithms are considered to be ***Insufficient****.*

The overall preference is to make use of AES-256-GCM or ChaCha20-Poly1305. These offer the most solid security rating. If you would also like to support other algorithms, please consider the following.

- AES ciphers in GCM mode are preferred over other modes of operation due to their widespread support in libraries and implementations. AES-GCM also generally requires less processing power than AES-CCM.
- In theory, a CRQC weakens the cryptographic strength of bulk encryption algorithms. In practice, there are currently different factors that make any attack on bulk encryption algorithms with the use of a CRQC impractical [18] [19]. Algorithms AES-128-GCM and AES-128-CCM still offer **Sufficient** protection to this end.
- Ciphers ARIA and, to a lesser extent, CAMELLIA are rarely used in the western world. The ciphers are, to a limited extent, supported in modern browsers and operating systems and, as yet, not included in the TLS 1.3 standard. There are no indications that the algorithms in itself are unsecure. However, limited adoption is, in addition to limited interoperability, a risk factor for undiscovered software vulnerabilities. Do not use these ciphers, unless there are compelling reasons from an interoperability point of view.
- Block ciphers AES, CAMELLIA and ARIA only support a 128-bit, 192-bit and 256-bit key length. In practice, only configurations with a 128-bit and 256-bit key length enjoy broad-based support, for which reason only these configurations are included in
  Table 8.
- AES-ciphers in CBC mode require the use of the 'encrypt-then-MAC' extension. The recommendation is **to phase out** its use [6]. In practice, this extension has limited support from clients and servers and the use thereof cannot be enforced, thus resulting in security risks.

3.3.5 Hash functions for TLS

Make use of secure hash functions as described in
Table 9.

Algorithm	Security level	Cryptographic strength (bit)
SHA-512	Good	256
SHA-384		192
SHA-256		128
SHA-224	To be phased out	112
SHA-1	Insufficient	< 63
MD5		< 19

Table 9: Algorithms for hash functions, their security level and cryptographic strength, based on collision resistance [6] [17] [13] [20].

3.4 Secure options

Use the recommendations from this section to make safe choices for other TLS options.

3.4.1 Compression

Do not use make use of TLS compression. This could make your application vulnerable to security attacks such as CRIME [3]. If you use TLS 1.3, there is no need to switch off TLS compression yourself, as TLS 1.3 no longer supports this.

Compressie-optie	Security level
TLS 1.3: N/A	Good
TLS 1.2: No TLS-compression	
TLS 1.2: TLS-compression	Insufficient

Table 10: Options for TLS compression and their security level.

In addition to TLS compression, you may choose to apply compression at application level. This falls outside the scope of your TLS configuration, but it is good to know that this may entail a security risk. A BREACH attack serves as an example of this, whereby compression of web traffic introduces a vulnerability [2]. Make sure to check whether your application is vulnerable to such attacks before choosing to apply compression at application level.

3.4.2 Renegotiation

If your configuration supports TLS 1.2:

- make sure that your configuration does not support *insecure* renegotiation. To do so, check whether the security fix from RFC 5746 has been correctly implemented in the TLS library you use [4].
- Preferably, switch off the option for (secure) renegotiation. Renegotiation makes your server more vulnerable for DoS attacks for which mitigation is non-trivial. Generally speaking, it is not necessary to offer clients the possibility of initiating a renegotiation.
- If your configuration does really need to support renegotiation, please limit the number of authorised attempts on the server.

If your configuration only supports TLS 1.3, you do not need to do anything. TLS 1.3 does not support renegotiation.

Client-initiated renegotiation option	Security level
TLS 1.3: N/A	**Good**
TLS 1.2: No renegotiation	
TLS 1.2: Limited secure renegotiation	**Sufficient**
TLS 1.2: Unlimited secure renegotiation	**To be phased out**
TLS 1.2: Insecure renegotiation	**Insufficient**

Table 11: Options for renegotiation and their security level.

3.4.3 Resuming sessions (session IDs, session tickets)

If you want to make it possible for your application to quickly resume sessions, make use of TLS 1.3. This TLS version offers the most secure support for this. Previous versions of TLS posed the risk that attackers could decrypt information if they could get hold of the server's *encryption key for session tickets.* This shortcoming has been corrected in TLS 1.3.

If you make use of session tickets in TLS 1.2, your server is vulnerable to attackers who know how to steal your encryption key for session tickets. Take extra measures to prevent access to this key. Make it as inaccessible to third parties as possible and replace it regularly.

If you make use of session IDs in TLS 1.2, protect the stored session parameters in a similar manner as the *encryption key for session tickets* as described above. From a usability perspective, session tickets are preferred, given the extra overhead that the use of session IDs entails.

Resume setting option	Security level
No resumption	**Good**
TLS 1.3: Session tickets	
TLS 1.2: Session tickets + additional security measures	**Sufficient**
TLS 1.2: Session-ID's + additional security measures	
Other	**Insufficient**

Table 12: Options for resuming sessions and their security level.

3.4.4 Resuming sessions (0-RTT)

If your configuration supports TLS 1.3, disable the '0-RTT' option by default. This option makes your TLS configuration vulnerable to replay attacks. The resulting risk depends on the application data that are sent along during the handshake. 0-RTT can thus only be used in a secure manner in an application-specific context [6]. Supporting 0-RTT thereby requires both expertise and following application-specific 0-RTT profiles [6]. In an application-agnostic context, the use of 0-RTT if **Insufficient**.

If your configuration only supports TLS 1.2, you do not need to do anything. TLS 1.2 does not support 0-RTT.

0-RTT option	Security level
TLS 1.3: No 0-RTT	Good
TLS 1.2: N/A	
TLS 1.3: Using 0-RTT	Insufficient

Table 13: Options for 0-RTT and their security level.

3.4.5 Certificate status check

Configure the use of OSCP stapling on your server if possible. This will offer your clients a privacy-friendly way of checking the revocation status of your certificate. Do note, however, that not all CAs offer support for OCSP for privacy considerations. In practice, this means it will not always be possible to use OCSP stapling.

If you also manage the client in addition to the server, give preference to the use of CRLs or OCSP stapling over regular OCSP for the certificate status check.

4 Preconditions

A TLS configuration is 'secure' if it supports secure settings. However, this requires more than just a secure configuration. There are different aspects that are not directly part of your TLS configuration, but which are, however, important for the secure operation of TLS. This section provides recommendations on these aspects.

4.1 TLS libraries

Choose a reliable TLS library that suits your application. Ask yourself or ask the library's supplier or the person integrating the library into your systems the following questions in order to gain insight into the reliability of this library.

Does the TLS library which you selected, for example, have:

- A public policy on the manner in which vulnerabilities can be reported?
- Developers with access to adequate means to provide support in terms of security?
- A good track record in terms of responding to TLS attacks in the past and to vulnerabilities in the implementation of the library?
- A means of informing users about security updates, in a way that is clearly distinguishable from regular updates?
- Is the library regularly monitored and assessed in an independent manner?
- Does the library make use of constant-time implementations to better withstand *side-channel attacks*?
- Has the library implemented the recommended measures from RFC 7627 (extended_master_secret [21]) and RFC 5746 (secure renegotiation [4])?
- Does the library lend itself to crypto-agility? To what extend can cryptographic algorithms simply be replaced (for example, in case any vulnerabilities are found)? And does the library already have support for quantum-safe cryptographic algorithms?

Operating systems usually have more than one TLS library. Make sure you know which library the server software uses and ensure that it is up-to-date. Always select the most up-to-date version of your chosen library that may contain fixes for potential vulnerabilities in earlier versions. Only use settings that are necessary for your application. This will prevent programming errors in non-essential features from leading to vulnerabilities.

4.2 Random numbers

Random numbers are formed by linking a source of randomness (entropy) to a PRNG, a pseudo-random number generator. Such numbers are used in a variety of ways in TLS. For example, the secret key for bulk encryption is derived from an entropy source combined with a hash function.

Most operating systems and TLS libraries contain a solid, cryptographically-safe PRNG. Identify the PRNG used in your TLS library and find out whether this generates high-quality random numbers, i.e. unpredictable random numbers. Consult the TLS library which you use for your application or its supplier. The following RNG is known to be unsecure:

- Dual EC DRBG [22]

It may be that generating random numbers causes your TLS server to overload. Adding a hardware module, i.e. a hardware-based RNG, ensures that there will always be sufficient entropy and random numbers available. Such a feature is already also embedded in modern processors.

4.3 TLS proxy

Are you considering deploying a TLS proxy? Gain insight into the security and privacy risks involved and manage them appropriately.

If a commercial party requests your secret keys in order to set up a TLS proxy – for example as part of anti-DDoS measures – do not make these readily available.

It is possible to set up a TLS proxy without giving away the secret keys[9]. If you are opting for a supplier that does not support this, identify the risks that arise from disclosing your secret keys. Take contractual measures to compensate for reduced technical control and regularly monitor the extent to which the supplier implements the agreed measures.

4.4 Certificates

Adequate certificate management is an important prerequisite for the secure use of TLS. Consult the NCSC publication "Zorgeloos Certificaatbeheer" for more information about (managing) certificates.

4.5 Quantum security

Prepare your organisation and application for the advent of CRQCs. Even though CRQCs do not yet exist, they may already pose a risk to your organisation. Attackers could, for instance, collect your encrypted data now and decrypt them at a later date when they do have a CRQC at their disposal (a *store-now-decrypt-later* attack). History also shows that any migration to more secure algorithms involves complex challenges that require a lot of time, planning and preparation. You will find more information on this topic in the Guide titled '*Make your organisation quantum secure*' [23].

Quantum-safe cryptographic algorithms for the purpose of authentication and key exchange are not *yet* part of the TLS standards. There are, however, processes underway to change this. For instance, a standardisation process is being launched for the use of key exchange algorithms X25519MLKEM768, SecP256r1MLKEM768 and SecP384r1MLKEM1024 in TLS [15]. These algorithms, and X25519MLKEM768 in particular, are increasingly used on the internet and supported by TLS libraries. It is expected that quantum-safe cryptography will gradually become part of the TLS standards. That is not to say that you have to wait for it. Take the following steps to prepare your TLS configuration and organisation for the advent of cryptographically-relevant quantum computers:

1. Support TLS 1.3. (or a recent TLS library with support for TLS 1.3) as quickly as possible. Quantum-safe crypto algorithms are expected to be only made available for TLS 1.3. By making use of TLS 1.3 now, you ensure a smoother necessary and future migration.
2. Quantum-safe cryptography requires larger ClientHello messages to function. Not all servers can deal with this due to implementation limitations. Find out the extent to which your server has such a limitation and to what extent you can eliminate it. See https://tldr.fail/ for more information.
3. Use algorithms for hashing and bulk encryption with 256-bit cryptographic strength. These offer the highest security rating, even against a cryptographically-relevant quantum computer.
4. Make sure that you know how and where your organisation makes use of TLS. That way, you ensure that you can make quick and targeted adjustments.
5. Make sure that your configuration and the cryptographic algorithms used are easy to replace (crypto-agility). This allows you to quickly perform a migration to secure cryptography.
6. Perform a risk analysis to find out the ways in which the quantum threat poses a risk for your application and organisation and the ways in which you already need to take action at this time.
    a. If your risk analysis shows that your application/organisation is not at urgent risk with regard to the quantum threat, wait to implement quantum-safe cryptographic algorithms until these form part of the TLS standards and TLS libraries.
    b. If your application/organisation is already at significant risk, gather expertise and implement one or more of the hybrid key exchange algorithms that may qualify for TLS standardisation, as described in Section 3.3.3.

Please consult the 'PQC migration handbook' for more tools and analyses of the abovementioned steps [24]. This manual has comprehensive advice on the quantum threat, the performance of a risk analysis, an overview of quantum-safe cryptographic algorithms and a description of *hybrid cryptography*, in which 'classic' and 'quantum-safe' algorithms are being combined. The manual also offers tools for different personas, from *early adopters* – those organisations that already need or want to take measures in anticipation of the standardisation of quantum-safe cryptography – to *regular adopters*, i.e. those who can wait a little longer for the standards. Also, read the guide *Make your organisation quantum secure* for a general course of action to prepare your organisation for the advent of cryptographically-relevant quantum computers.

[9] *See, for example https://blog.cloudflare.com/keyless-ssl-the-nitty-gritty-technical-details/*

Appendix A: Changes with respect to the previous version

This Appendix contains the main changes to these guidelines compared to the version from 2021, titled 'ICT-beveiligingsrichtlijnen voor Transport Layer Security v2.1 (TLS)' (*'ICT security guidelines for Transport Layer Security v2.1 (TLS)*, in Dutch).

General change

- The structure of the publication was revised so as to make a clearer distinction between background information, TLS-specific recommendations and conditional recommendations.

A fine-tuned definition of security levels and quantum security

- The definition and criteria for the security levels **Good**, **Sufficient** and **To be phased out** have been fine-tuned and further made explicit.
- The threat of cryptographically-relevant quantum computers is explicitly included in the assessment of these security levels.

TLS Versions

- In conformity with TLS best-current practices (BCP 195), TLS1.0 and TLS 1.1 have been downgraded to **Insufficient** [6].

Certificate signature and validation

- Algorithms ECDSA, RSA, EdDSA and associated parameters have been downgraded from **Good** to **Sufficient** in view of the quantum threat.
- The key length for RSA 2048-3071 has been downgraded to **To be phased out.** These guidelines pursue a cryptographic strength of 128 bits, following an RSA key length of 3,072 bits.

Key exchange

- Algorithms X25519MLKEM768, SecP256r1MLKEM768 and SecP384r1MLKEM1024 are included as **Good**.
- Algorithm ECDH and associated elliptic curves have been downgraded from **Good** to **Sufficient** with a view to the quantum threat.
- In accordance to BCP 195, DHE for key exchange has been downgraded to **To be phased out**[6]. The corresponding finite-field groups have also been downgraded. Finite-field group ffdhe2048 has been downgraded to **Insufficient**, given the limited cryptographic strength offered by this finite-field group.
- In accordance to BCP 195, RSA for key exchange has been downgraded to **Insufficient** [6].
- The recommendation for the PSS padding scheme has been moved from Table 4 to the accompanying text.

Bulk encryption

- In accordance to BCP 195, Ciphers in CBC mode are downgraded to **To be phased out** [6]. This with a view to the fact that block ciphers in CBC mode can only be securely used if the 'encrypt_then_mac' extension is used from RFC7366 [25]. Both the client as well as the server need to make use of this extension. In practice, this extension has limited support and the use thereof cannot be enforced, thus resulting in security risks.
- The security levels for CAMELLIA, ARIA and SEED are described in more explicit terms. There are no known algorithmic vulnerabilities for CAMELLIA and ARIA. However, given the limited adoption, there is a risk of undiscovered software vulnerabilities. It is for this reason that these algorithms are classified as **To be phased out**. SEED is an outdated algorithm and is included as **Insufficient**.
- AES-128-GCM has been downgraded to **Sufficient**.
- AES in CCM mode has been included as **Sufficient**.
- 3DES-CBC has been downgraded to **Insufficient**.

Hash functions

- Older versions of these guidelines made a distinction between the specific application of hash functions and the corresponding security levels. This resulted in three tables, one for *authentication*, one for *key exchange* and one for *bulk encryption*, each with a different format. In terms of content, these tables hardly differed. This version houses hash functions in one table along with associated recommendations. The use of SHA-1 has been included in this as **Insufficient**, regardless of the application.
- Hashing algorithm SHA-224 has been explicitly included as **To be phased out**. This is not a substantive change to the guideline, but a clarification.

Other options

- The use of client-initiated renegotiation has been downgraded to **To be phased out**.
- The recommendation for other options have not been changed substantially, but have been made further explicit and provided with supplementary substantiations.

Appendix B: List of cipher suites

As a tool for configuring and monitoring your TLS settings, this Appendix describes the cipher suites that are awarded security levels **Good**, **Sufficient** and **To be phased out**. We apply the rule that the security level of a cipher suite is as low as the lowest-valued algorithm. The list in this Appendix has been drawn up based on the lists of IANA, OpenSSL and GnuTLS, retrieved via https://ciphersuite.info/ on 20 March 2025.

Please note that this Appendix does not contain an exhaustive list of cipher suites that are **Good**, **Sufficient** and **To be phased out**. It is possible that your TLS library supports other cipher suites. In that case, please check the extent to which you can support these cipher suites based on the security level of the respective algorithms. Furthermore, it is not enough to only configure these cipher suites to arrive at a secure TLS configuration. For more settings and preconditions, please consult Sections 3 and 4.

TLS 1.3 - Good

TLS_AES_256_GCM_SHA384
TLS_CHACHA20_POLY1305_SHA256

TLS 1.3 - Sufficient

TLS_AES_128_CCM_SHA256
TLS_AES_128_GCM_SHA256

TLS 1.2 - Sufficient

TLS_ECDHE_ECDSA_WITH_AES_128_CCM
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_CCM
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256

TLS 1.2 - To be phased out

TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
TLS_DHE_RSA_WITH_AES_128_CCM
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
TLS_DHE_RSA_WITH_AES_256_CCM
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
TLS_DHE_RSA_WITH_ARIA_128_CBC_SHA256
TLS_DHE_RSA_WITH_ARIA_128_GCM_SHA256
TLS_DHE_RSA_WITH_ARIA_256_CBC_SHA384
TLS_DHE_RSA_WITH_ARIA_256_GCM_SHA384
TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA256
TLS_DHE_RSA_WITH_CAMELLIA_128_GCM_SHA256
TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA256
TLS_DHE_RSA_WITH_CAMELLIA_256_GCM_SHA384
TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_ECDSA_WITH_ARIA_128_CBC_SHA256
TLS_ECDHE_ECDSA_WITH_ARIA_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_ARIA_256_CBC_SHA384
TLS_ECDHE_ECDSA_WITH_ARIA_256_GCM_SHA384
TLS_ECDHE_ECDSA_WITH_CAMELLIA_128_CBC_SHA256
TLS_ECDHE_ECDSA_WITH_CAMELLIA_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_CAMELLIA_256_CBC_SHA384
TLS_ECDHE_ECDSA_WITH_CAMELLIA_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_RSA_WITH_ARIA_128_CBC_SHA256
TLS_ECDHE_RSA_WITH_ARIA_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_ARIA_256_CBC_SHA384
TLS_ECDHE_RSA_WITH_ARIA_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_CAMELLIA_128_CBC_SHA256

TLS_ECDHE_RSA_WITH_CAMELLIA_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_CAMELLIA_256_CBC_SHA384
TLS_ECDHE_RSA_WITH_CAMELLIA_256_GCM_SHA384

References

[1] Internet Engineering Task Force (IETF), “RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3,” 2018.

[2] A. Prado, N. Harris and Y. Gluck, “https://www.breachattack.com/,” 2013. [Online].

[3] D. Fischer, “CRIME Attack Uses Compression Ratio of TLS Requests as Side Channel to Hijack Secure Sessions,” ThreatPost, 2012.

[4] Internet Engineering Task Force (IETF), “RFC 5746: Transport Layer Security (TLS) Renegotiation Indication Extension,” 2010. [Online].

[5] Internet Engineering Task Force (IETF), “RFC 5077: Transport Layer Security (TLS) Session Resumption without Server-Side State,” 2008.

[6] Internet Engineering Task Force (IETF), “BCP 195 (RFC 7525 & RFC 8996): Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS),” 05 2021. [Online]. Available: https://www.rfc-editor.org/bcp/bcp195.txt.

[7] CA/Browser Forum, “Baseline Requirements for the Issuance and Management of Publicly-Trusted Certificates v1.8.4,” CA/Browser Forum, 2022.

[8] Internet Engineering Task Force (IETF), “RFC 5480: Elliptic Curve Cryptography Subject Public Key Information,” 2009.

[9] Thales, “Supported ECC curves,” Thales, [Online]. Available: https://www.thalesdocs.com/gphsm/ptk/protectserver3/docs/ps_ptk_docs/ptkc_programming/ecc_curves/index.html. [Accessed 13 02 2025].

[10] SSL Support team, “What is Elliptic Curve Cryptography (ECC)?,” SSL.com, 08 10 2024. [Online]. Available: https://www.ssl.com/article/what-is-elliptic-curve-cryptography-ecc/#ftoc-heading-6. [Accessed 13 02 2025].

[11] German Federal Office for Information Security (BSI), “TR-02102-02: "Cryptographic Mechanisms: Recommendations and Key Lengths, Part 2 – Use of Transport Layer Security (TLS)" Version: 2025-1,” 04 03 2025. [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-2.pdf.

[12] National Institute of Standards and Technology (NIST), “SP 800-131A: Transitioning the Use of Cryptographic Algorithms and Key Lengths, revision 2,” 2019.

[13] The National Cyber and Information Security Agency of the Czech Republic (NÚKIB), “Minimum requirements for cryptographic algorithms,” 2024.

[14] National Institute of Standards and Technology (NIST), “FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard,” 2024.

[15] K. Kwiatkowski, P. Kampanakis, B. Westerbaan and D. Stebila, “Post-quantum hybrid ECDHE-MLKEM Key Agreement for TLSv1.3,” IETF, 25 12 2024. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-kwiatkowski-tls-ecdhe-mlkem-03. [Accessed 17 03 2025].

[16] Internet Engineering Task Force (IETF), “RFC 7919 : Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS),” 08 2016. [Online].

[17] German Federal Office for Information Security (BSI), “BSI TR-02102-1 "Cryptographic Mechanisms: Recommendations and Key Lengths" Version: 2025-1,” 04 03 2025. [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.pdf.

[18] Sarah D., UK National Cyber Security Centre, “On the practical cost of Grover for AES key recovery,” in *5th PQC Standardization Conference*, 2024.

[19] S. Mandal, M. Rahman, S. Santanu and I. Takanori, “Implementing Grover’s on AES-based AEAD schemes,” *Nature Scientific Reports 14,* 10 11 2024.

[20] National Institute of Standards and Technology (NIST), “Hash Functions,” 09 09 2024. [Online]. Available: https://csrc.nist.gov/projects/hash-functions. [Accessed 13 02 2025].

[21] Internet Engineering Task Force (IETF), “RFC 7627: Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension,” 2015.

[22] NIST, “NIST opens draft Special Publication 800-90A, Recommendation for random number generation using deterministic random bit generators, for review and comment,” NIST, 2012.

[23] AIVD, NCSC, “Make your organization quantum secure,” 2023. [Online]. Available: https://english.ncsc.nl/publications/publications/2024/march/25/quantum-secure.

[24] AIVD, CWI, TNO, “The PQC migration handbook,” 2024. [Online]. Available: https://english.aivd.nl/publications/publications/2024/12/3/the-pqc-migration-handbook.

[25] Internet Engineering Task Force (IETF), “RFC 7366: Encrypt-then-MAC for Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS),” 2014.

[26] National Institute for Standards and Technology (NIST), “NIST SP 800-131Ar3 ipd: Transitioning the Use of Cryptographic Algorithms and Key Lengths; Initial Public Draft,” 01 10 2024. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar3.ipd.pdf.

[27] Internet Engineering Task Force (IETF), “RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile,” 05 2008. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5280.

[28] Nationaal Cyber Security Centrum, “Factsheet TLS-interceptie,” NSCS, 01 06 2019. [Online]. Available: https://www.ncsc.nl/documenten/factsheets/2019/juni/01/factsheet-tls-interceptie.