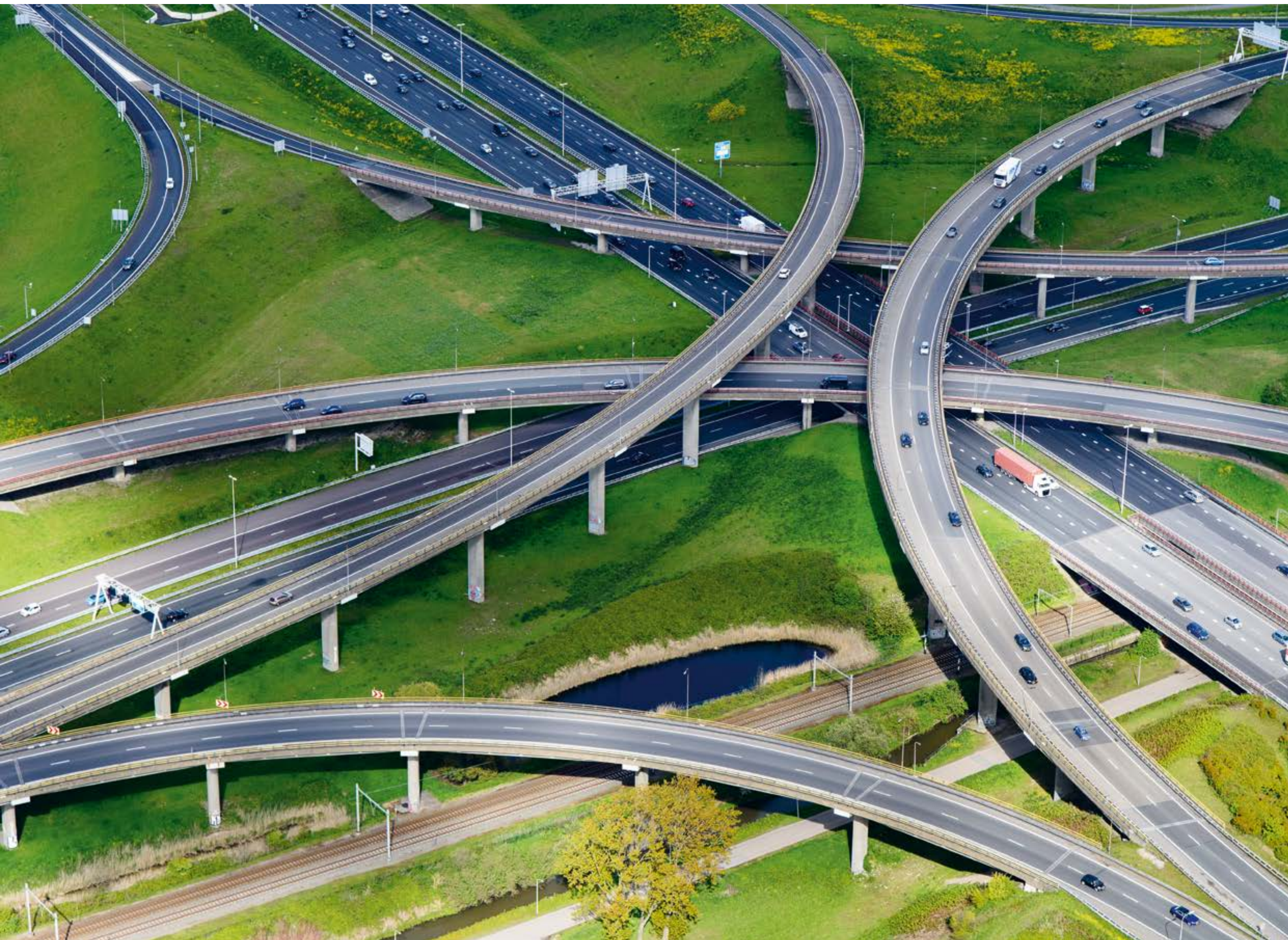
National Cyber Security Centre
Ministry of Justice and Security

IT Security Guidelines for Transport Layer Security (TLS)



National Cyber Security Centre

The National Cyber Security Centre (NCSC), in collaboration with the business community, government bodies and academics, is working to increase the ability of Dutch society to defend itself in the digital domain.

The NCSC supports the central government and organisations in the critical infrastructure sectors by providing them with expertise and advice, incident response and with actions to strengthen crisis management. In addition, the NCSC provides information and advice to citizens, the government and the business community relating to awareness and prevention. The NCSC thus constitutes the central reporting and information point for IT threats and security incidents.

These IT Security Guidelines for Transport Layer Security were first published by the NCSC in 2014. This update (v2.1) was published in 2021. See the appendix *Changes to these guidelines* for more details.

This publication was produced in collaboration with the following partners:
- the national communication security agency (NBV), part of the general intelligence and security service (AIVD)

The following organizations and individuals have provided valuable contributions:
- Autoriteit Persoonsgegevens
- Belastingdienst
- Centric
- Dienst Publiek en Communicatie
- Forum Standaardisatie
- IBD
- KPN
- NLnet Labs
- Northwave
- Platform Internetstandaarden
- RDW
- SURFnet
- de Volksbank
- Z-CERT

- Daniel Kahn Gillmor, ACLU
- Tanja Lange, Eindhoven University of Technology
- Kenny Paterson, ETH Zurich
- Rich Salz, Akamai Technologies
- Nick Sullivan, Cloudflare

IT Security Guidelines for Transport Layer Security (TLS)

Contents

Introduction

These guidelines are intended as advice during procurement, set-up or review of configurations of the Transport Layer Security protocol (TLS) on servers. TLS is the most popular protocol to secure connections on the Internet.

Purpose

These guidelines do not contain step-by-step instructions for the configuration of TLS.[1] Nevertheless, they are technical in nature. This publication helps an organisation choose between all possible configurations of TLS to arrive at a secure configuration. An administrator or supplier then applies this configuration.

Use for procurement

Organisations that procure IT systems can refer to this publication when stating their requirements. A supplier is thus asked to supply and maintain a secure TLS configuration by conforming to the guidelines in this publication.

Level of security

Deciding on the right TLS configuration is ultimately each organisation's prerogative. It is a complex job. Each option requires a choice between the available alternatives, where often many exist. Security plays a role here, but so does compatibility with software of customers or end users. The guidelines in this publication help navigate this effort.

To aid in the choice of a configuration, the settings for the options available in TLS are divided in four security levels.

- A setting that is **Insufficient** should not be chosen. TLS configurations that contain these settings are not secure.
- A **Phase out** setting is known to be fragile with respect to evolving attack techniques and merely provides a slim security margin. This places them at risk of becoming **Insufficient** in the near future. For some applications, **Phase out** settings are (still) needed to support very old clients. The use of these settings should be subject to written deprecation conditions that schedule their removal.
- If a setting is **Sufficient**, it 'does the job' for the time being. It is possible to use such a setting in a secure TLS configuration. Many **Sufficient** settings are required for compatibility with older client systems.
- The most secure and future-proof settings are **Good**. If you have the freedom to choose which settings you support, then only use **Good** settings.

New or improved attack techniques periodically appear for TLS. These attack techniques usually concern **Phase out** or **Sufficient** settings. A setting rendered insecure by an attack technique will lose its status as **Phase out**, **Sufficient** or **Good**. If that happens an addendum to these guidelines will be published. For further details, see the appendix *Changes to the guidelines*.

Good settings are likely to be more future-proof than **Sufficient** settings. Even so, there are no guarantees. Moreover, no single TLS configuration remains secure forever. Even TLS configurations consisting only of **Good** settings will need updates at some point. This is the case when **Good** settings become **Insufficient**.
The words 'insufficient', 'phase out', 'sufficient' and 'good' have a meaning in regular use. To distinguish these uses, they are displayed in a **different font** throughout this publication when referring to a security level.

1 The book 'Bulletproof SSL and TLS' by Ivan Ristic (ISBN 978-1907117046) offers step-by-step instructions on the configuration of various software for the secure use of TLS, in addition to extensive background information. Mozilla offers configuration examples for popular web server software on its wiki https://wiki.mozilla.org/Security/Server_Side_TLS. The website https://bettercrypto.org/ also offers step-by-step instructions. Note that these resources may not yet be up to date with the introduction of TLS 1.3 and that the advice in these publications differs slightly from the advice in this document.

Key message

The secure configuration of TLS is important to secure network connections. TLS has secure and less secure settings. Legacy software does not always support the most secure settings. Use **Good** settings when possible and complement these with **Sufficient** settings to support legacy software. Do you need to support a lot of legacy software? Then use a broad palette of **Sufficient** settings and complement it with **Good** settings where possible. Use **Phase out** settings only whilst you have a further need for client compatibility and set clear criteria for their deprecation. Do not use **Insufficient** settings.

Outline

The core of these guidelines consists of the chapters *Usage guidance*, *Guidelines* and *Versions, algorithms and options*. The chapter *Usage guidance* is aimed at people that need to create their own secure TLS configuration. It offers guidance to arrive at a secure configuration. The chapter *Guidelines* is meant for people who judge TLS configurations, such as auditors. This may include configurations on paper or in practice. The chapter *Versions, algorithms and options* lists relevant TLS options. It describes secure settings for every option listed. Other chapters regularly refer to the chapter *Versions, algorithms and options* for details.

These guidelines can be read in three ways:
- If you are designing a TLS configuration yourself, then read the chapter *What is Transport Layer Security?*, followed by the chapter *Usage guidance*. The chapter *Usage guidance* will refer you to the relevant parts of the chapter *Versions, algorithms and options*.
- Do you want to know how certain settings for TLS options influence its security? Then refer to the chapter *Versions, algorithms and options*.
- Are you assessing a TLS configuration? Then read the chapter *What is Transport Layer Security?*, followed by the chapter *Guidelines*. The chapter *Guidelines* will refer you to the relevant parts of the chapter *Versions, algorithms and options*.

References

This publication makes use of multiple styles of references:
- Guidelines are numbered, say B2-1, and are found in the chapter *Guidelines*.
- Technical terms are not always introduced upon first use. If a term is marked in this way, then it can be found in the *Glossary* at the back.
- To supply background information, footnotes[2] are used.
- The support for the advice provided is based on the *References* that can be found in the back. If a particular reference supports an advice, it is shown in the following manner: (**1**)

2 In this manner.

1 What is Transport Layer Security?

Transport Layer Security (TLS) is a protocol for the establishment and use of a cryptographically secured connection between two computer systems, a client and a server. After establishing a secure connection with the TLS protocol, applications can use the connection to exchange data between the client and the server. TLS is applied in a large number of contexts. Well known examples include web traffic (https), e-mail traffic (IMAP and SMTP after STARTTLS) and certain types of virtual private networks (VPN).

Why TLS?

TLS protects the communication between client and server. The protection of communication is important when sensitive information is sent over a connection. Information can be sensitive due to confidentiality (say login credentials) and due to integrity (say a financial transaction).
In some cases, the use of encrypted connections is obligatory. This obligation can flow from an organization's policy, but also from law or regulation:

- The Dutch Standards Forum's "comply or explain" regime requires use of TLS for communication between parts of the Dutch government, such as the secure exchange of e-mail.[3] Use of HTTPS will become mandatory for all government websites.[4]
- The PCI Data Security Standard (PCI DSS) is a payment industry policy that requires the use of encrypted transmission of cardholder data over open, public networks.[5]
- The Dutch Data Protection Authority (Autoriteit Persoonsgegevens) requires the use of HTTPS on websites that collect personal data.[6] This requirement flows from the General Data Protection Regulation.

In every example, use of TLS ensures that the data sent cannot be seen or modified by others in transit. Because sensitivity is user-dependent, encrypted communication (and TLS) have become the norm, rather than the exception in many contexts.

TLS only protects the contents of the communication. Information about the data transport is not protected. In this aspect, TLS differs from IPsec. TLS works on the transport layer. IPsec works on the internet layer.[7]

There are currently seven different versions of TLS. Three carry its old name: Secure Sockets Layer (SSL) 1.0, 2.0 and 3.0. These were developed by Netscape. Then came TLS 1.0, 1.1, 1.2 and 1.3. These were standardized by the Internet Engineering Task Force (IETF). The IETF maintains TLS as an open standard. The most recent version of TLS is 1.3.[8]

A client or server can support multiple versions of TLS.
The individual versions are not compatible. Every version has its own options, for example in bulk encryption, authentication and key exchange.

How TLS works

A connection between a client and server secured by TLS is called a TLS session. A TLS session consists of two phases: the handshake phase and the application phase. During the handshake, the client and server agree on the way the TLS session is established. The following are examples of matters that need to be agreed upon during the handshake:

- What version of TLS will be used?
- What key will be used to exchange further data and how will it

3 (Dutch) https://www.forumstandaardisatie.nl/standaard/tls.

4 (Dutch) https://www.rijksoverheid.nl/ministeries/ministerie-van-binnenlandse-zaken-en-koninkrijksrelaties/documenten/kamerstukken/2018/10/16/kamerbrief-over-verhogen-informatieveiligheid-bij-de-overheid

5 PCI-DSS v3.2.1, Req. 4, see https://www.pcisecuritystandards.org

6 (Dutch) https://autoriteitpersoonsgegevens.nl/nl/onderwerpen/beveiliging/beveiliging-van-persoonsgegevens#moet-ik-altijd-https-gebruiken-voor-mijn-website-6069

7 The transport layer and the internet layer are part of the internet protocol suite, a model to describe network traffic. This model is described in RFC 1122, available at https://datatracker.ietf.org/doc/rfc1122/.

8 The specification of TLS 1.3 is documented in RFC 8446, available at https://datatracker.ietf.org/doc/rfc8446/.

be chosen (key exchange)?
- Which certificate will the server use to prove its identity to the client?
- Will the client present a certificate? If so, which?
- What cipher suite will be used to encrypt data during the application phase?

The handshake is started by the client. During the handshake, the client and server negotiate four cryptographic algorithms, one algorithm for key exchange, one algorithm for digital signatures, one algorithm for bulk encryption and an algorithm for hashing. In these guidelines, this set of four algorithms is called an algorithm selection.[9] Then, the client verifies the authenticity of the certificate that the server provides. If the client offers a certificate to the server, its authenticity is verified by the server.[10]

After the handshake phase completes, the application phase starts. During the application phase, the TLS session is available as a secure tunnel for data transfer. Applications can use this tunnel to send their traffic between client and server. Applications do not have to concern themselves with the inner working of this tunnel: they can trust it as an abstract communications channel that guarantees confidentiality and integrity of information.

Software libraries

TLS is used in many different software applications. Programming all functionality in TLS from scratch is a lot of work and requires specialist knowledge. That is why most software does not contain its own code for TLS. Instead, they use a TLS software library.

There are different TLS software libraries available. Some are free software; others are available as a proprietary product. They can be included in operating systems or delivered separately. Well known TLS libraries include OpenSSL[11], SChannel[12], NSS[13] and mbed TLS[14]. These guidelines do not judge the security of specific TLS libraries. All software contains bugs, including TLS libraries. Bugs can lead to vulnerabilities. Every library has its advantages and disadvantages. Not every setting for TLS is available in each library.

Ask the supplier of your chosen TLS library (or the vendor that embeds it) about the following topics to get a rough understanding of its dependability. Does your chosen TLS software library:
- document how to report security vulnerabilities?
- have developers that have access to enough resources to provide (security) support?
- have a good track record of responding to past attacks on TLS or vulnerabilities in its implementation in the library?
- make their security-updates known to their users and distinct from feature updates?
- get audited or independently reviewed?
- use constant time implementations to harden against attacks based on timing side channels?

The NCSC advises to make conscious choices on the use of TLS libraries:
- Always use the most recent version of your chosen TLS library. This gives its creators the most time to fix vulnerabilities.
- Choose only settings that are necessary to meet business requirements. This way, bugs in non-necessary functionality will not lead to system vulnerabilities. The chapter *Usage guidance* helps with this selection.

The importance of random numbers

Random numbers play a crucial role in many applications of cryptography, including TLS. TLS uses random numbers in multiple places in the protocol.

The quality of the random numbers used is critical to the security of TLS. Choosing the right settings for TLS is important, but no single setting can take away the risks introduced by using random numbers of low quality.

Every operating system and every TLS library contains methods to generate random numbers. In addition, hardware is available for the generation of random numbers. These hardware modules produce random numbers faster and of higher quality than software-only methods do.

You can find more background and advice on methods to generate random numbers in the section *Random number generators* in the appendix *Further considerations*.

9 See the callout *Changed meaning of cipher suite in TLS 1.3* in the chapter *Guidelines* for the reason behind this naming convention.

10 Note that TLS versions prior to TLS 1.3 do not fully encrypt the information exchanged during the handshake. This affects client certificates, which are sent unencrypted.

11 https://www.openssl.org/

12 https://docs.microsoft.com/en-us/windows/desktop/secauthn/secure-channel

13 https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS

14 https://tls.mbed.org/

2 Usage guidance

The guidelines in this publication are aimed at the security of a TLS configuration. In practice, security is not the only concern when choosing a configuration. The TLS configuration of the server should also be compatible with the TLS configurations of all clients that need to connect to the server. The configurations of client and server need to match in certain aspects.

For some choices, different selections for client and server do not exclude one another. This includes supported versions, algorithm selections and groups. To allow a client and server to communicate, each of these options needs to have one choice that both client and server support. For example, if the client supports TLS 1.0, TLS 1.1 and TLS 1.2 it can communicate with a server that supports TLS 1.0 and TLS 1.1, but not with a server that only supports TLS 1.3. The same principle applies to algorithm selections and supported groups.

Other choices determine how long the cryptographic key or parameter is. This includes RSA keys and ECDSA keys. To allow a client and server to communicate, both client and server need to support the chosen length of the key or parameter. Older software does not always support keys or parameters of sufficient length and newer software sometimes prevents the use of keys or parameters of insufficient length.

Finally, there are options: configurations that can only be turned ‘on’ or ‘off’. For example, a server does or does not support TLS compression. Configuring a server with the options that are included in this publication is not known to cause compatibility problems with clients.

TLS has a plethora of options beyond those discussed. We only discuss options that influence the security of TLS and that are sometimes not secure by default.

Scenario 1: Control over both client and server

In some situations, the party that has control over the configuration of the server also has control over the configuration of the client. An example is a web server serving an internal web application. This web server is only available to the same organisation’s clients. The NCSC advises to use **Good** settings in this scenario: this is the most secure and most future-proof configuration. In the chapter *Versions, algorithms and options* you can find which settings are **Good**.

Step by step

1. Make an inventory of the available TLS options that are available on the server.
2. Inventory the different clients that will make connections to the server.
3. Inventory for each type of client the set of settings that are supported.[15]
4. Choose **Good** settings for the following options:
   a. TLS version for the server. Ensure that every client supports a version that is also supported by the server.
   b. Algorithm selections for the server. Ensure that every client supports an algorithm selection that is also supported by the server.
   c. Key lengths for the server. Ensure that every client supports the chosen length.
   d. Supported elliptic curves for the server. Ensure that every client supports an elliptic curve that is also supported by the server. Finite field groups supported by the server, in case older clients do not support elliptic curves. In that case, ensure that every client supports a **Sufficient** finite field group that is also supported by the server.
   e. Other options for the server, unless there are strong reasons to choose **Sufficient** settings. These reasons flow from the client inventory created.

15 An overview of TLS configurations for different types of clients is available from https://www.ssllabs.com/ssltest/clients.html.

5. Does the server lack support for a **Good** setting that clients support? You have three options:
   a. Replace the clients by a type that does support **Good** settings for this option.
   b. Replace the server by a type that does support **Good** settings for this option.
   c. Choose a **Sufficient** setting for this option that is supported by both clients and server.
6. Does the server lack support for a **Good** and **Sufficient** setting that clients support? You have three options:
   a. Replace the clients by a type that does support (other) **Good** or **Sufficient** settings for this option.
   b. Replace the server by a type that does support **Good** or **Sufficient** settings for this option.
   c. Choose a **Phase out** setting for this option that is supported by both clients and server. This option is least preferred, because it comes with additional risk and is the least future-proof of all available configurations.
7. Configure the server with the chosen configurations. The TLS configuration is part of the software that uses the TLS connection. For example, if you want to offer HTTPS then TLS is configured as part of the web server software.
8. Test to establish that this configuration works for all types of clients. Do you run into any compatibility issues? Then go back to step 5.
9. Document the selected settings. Spend time on at least the following:
   a. Choose and document conditions for scheduled removal of any **Phase out** settings that have been chosen. See *Scheduling removal of phase out configurations* in the chapter *Guidelines* for examples of clear conditions.
   b. Document the reasons for choosing **Sufficient** instead of **Good** settings.

Scenario 2: Only control over the server

In other situations, the entity with control over the configuration of the server does not control the configuration of (all) clients that connect to the server.[16] An example is a web server serving a public website. The NCSC advises to use **Good** settings and to complement these with **Sufficient** settings in this scenario. In some deployments, it may be necessary to include **Phase out** settings while clients transition away from these less secure configurations. In the chapter *Versions, algorithms and options* you can find which settings that are respectively **Good**, **Sufficient** and **Phase out**.

16 This heading can also be read as “Only control over the client”, though clients are not the focus of these guidelines. An example is an e-mail service that acts as a TLS client when sending e-mail.

Step by step

1. Make an inventory of the types of clients that (need to) make connections to the server.[17] This is a requirement that is usually determined in consultation with the business owner.
   a. Make the trade-off visible: the value of compatibility versus the risk and associated support cost of increasingly fragile configurations.Make an inventory of the available TLS options that are available on the server.
2. Choose **Good** and **Sufficient** TLS-versions for the server.
3. Choose **Good** and **Sufficient** algorithms for the server. Support a broad range of **Sufficient** algorithms. These are often required for compatibility with older clients. Do not support more **Sufficient** algorithms than required for compatibility.
4. Choose **Sufficient** lengths of keys. Choose **Good** lengths instead if you are sure that all clients support these.
5. Choose **Good** elliptic curves for the server. Do you have reasons to assume that not every client supports these? Then also choose **Sufficient** elliptic curves. Do not support more curves than required for compatibility.
6. Some old clients do not support elliptic curves. Do you need to support these clients? Then select **Sufficient** finite field groups. Do not support more finite field groups than necessary for compatibility.
7. Configure the server with the chosen configurations. The TLS configuration is part of the software that uses the TLS connection. For example, if you want to offer HTTPS then TLS is configured as part of the web server software.
8. Think about the software that clients will use to connect. Test whether clients using this software can connect.
9. Are you having compatibility problems?[18] Track down which options cause these problems.
   a. Usually these problems can be solved by exchanging or supplementing **Good** settings with **Sufficient** settings.
   b. If the inventory created in steps 1 and 9 includes very old client software, you may need to temporarily include **Phase out** settings in your configuration. **Phase out** settingss are known to be fragile with respect to evolving attack techniques and merely provide a slim security margin. Do not support more **Phase out** algorithms than required for compatibility.

11. Choose and document clear criteria for the scheduled removal of any **Phase out** settings that have been chosen. See *Scheduling removal of phase out configurations* in the chapter *Guidelines* for examples of clear conditions. If removal impacts the requirement defined in step 1, this will usually involve consultation with the business owner.

17 Ideally by using TLS connection statistics gathered on the server, or on a similar service.

18 Does your organization make use of TLS interception, either locally on the client or on the network? This may be the source of your compatibility problem. The factsheet “TLS Interception” treats considerations and preconditions for the deployment of TLS interception. (https://english.ncsc.nl/publications/factsheets/2019/juni/01/factsheet-tls-interception)

Points of particular interest

- The guidelines in this publication influence the selection of a supplier for certificates: not every certificate supplier can supply every type of certificate. So discuss your TLS configuration with the administrators of certificates and public key infrastructures (PKIs) within your organisation.
- Checking TLS configurations can be part of vulnerability management, penetration tests and the regular audit cycle within your organisation. Several tools and websites exist that enable you to perform similar checks yourself.[19] You can compare the results of these checks to the guidelines. This way, you are ahead of any findings during a penetration test or audit.
- Operating systems usually contain more than one TLS software libary. Make sure you know which software library is used by the server software and ensure it remains up to date.

Diverging from these usage guidelines

The NCSC advises to set up TLS configurations based on these guidelines at all times. Still, this may prove unattainable in exceptional circumstances. Keep the following in mind in such situations:

- Perform risk analysis when diverging from the guidelines. Diverging will have a negative impact on security. Why is this acceptable? How did you arrive at that conclusion? What additional measures will you take to mitigate the resulting risks? Document the divergences and results of these considerations.
- Something is better than nothing. Even a connection that is protected by an **Insufficient** TLS configuration can be impenetrable to some attackers. Inability to fully meet the guidelines can never be a reason to disable TLS in its entirety.
- Assessing a TLS configuration requires extensive knowledge. If you are diverging from this usage guidance, then discuss your TLS configuration and the resulting risks with a domain expert.

19 Examples of such tools are testssl.sh (https://testssl.sh/) and sslyze (https://github.com/nabla-c0d3/sslyze). The website Internet.nl allows online testing against the guidelines in this publication for web and e-mail servers (https://www.internet.nl/). The website Qualys SSL labs allows for a similar online check for web servers (https://www.ssllabs.com/ssltest/).

3 Guidelines

In the guidelines, repeated references are made to configurations that are **Good, Sufficient** or **Phase out**. All configurations referenced can be found in the chapter *Versions, Algorithms and Options*.

Versions

Recent versions of TLS are more secure than older versions. The older versions of TLS contain vulnerabilities that cannot be repaired. These must therefore be avoided. A TLS configuration can support more than one version.

Number	*Guideline*
B1-1	All supported versions of TLS are **Good, Sufficient** or **Phase out**
See Chapter 4 – Versions, algorithms and options, Table 1 – Versions	

Algorithm selections

For each connection, the client and server negotiate the use of four cryptographic algorithms. One algorithm for key exchange, one algorithm for digital signatures in certificate verification, one algorithm for bulk encryption and one algorithm for hashing. We will refer to a set of four selected cryptographic algorithms as an algorithm selection. The pair of cryptographic algorithms for bulk encryption and hashing are known as a cipher suite and are used for record protection.

Examples of these algorithms include:
- Certificate verification: RSA, ECDSA, etc.
- Key exchange: ECDHE, DHE, RSA, etc.
- Bulk encryption: AES-GCM, ChaCha20-Poly1305, etc.
- Hashing: SHA-1, SHA-256, etc.

Changed meaning of cipher suite in TLS 1.3

The first version of these guidelines used the term cipher suite instead of algorithm selection. We changed our use of terminology to stay in step with a change in TLS 1.3.

A cipher suite up until TLS 1.2 included the algorithms for key exchange and digital signatures. Most of the resulting hundreds of combinations are listed in the protocol registry.

To avoid this naming explosion, cipher suites in TLS 1.3 only contain the algorithms used for bulk encryption and hashing.

Figure 1 shows the changed cipher suite notation in TLS 1.2 and TLS 1.3.

To set up a connection, TLS negotiates the use of an algorithm for each of these purposes. There are hundreds of valid combinations available. A TLS configuration can support multiple algorithm selections.

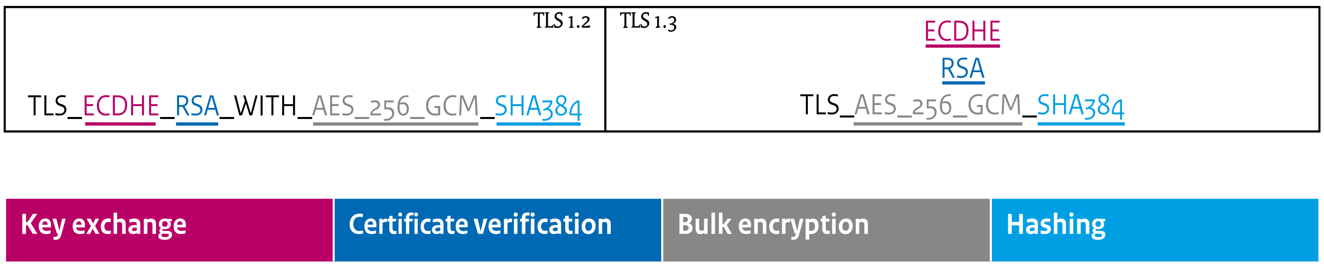


Figure 1 – Cipher suite notation in TLS 1.2 and TLS 1.3. The colours indicate the different algorithms and their purpose.
In TLS 1.3, the cryptographic algorithms for key exchange and certificate verification are no longer considered part of the cipher suite.

Number	*Guideline*
B2-1	All supported algorithm selections contain a **Good, Sufficient** or **Phase out** algorithm for certificate verification.
See Chapter 4 – Versions, algorithms and options, Table 2 – Algorithms for certificate verification.	
B2-2	All supported algorithm selections contain a **Good, Sufficient** or **Phase out** algorithm for key exchange.
See Chapter 4 – Versions, algorithms and options, Table 4 – Algorithms for key exchange.	
B2-3	All supported algorithm selections contain a **Good, Sufficient** or **Phase out** algorithm for bulk encryption.
See Chapter 4 – Versions, algorithms and options, Table 6 – Algorithms for bulk encryption.	
B2-4	All supported algorithm selections contain a **Good, Sufficient** or **Phase out** algorithm for hashing.
See Chapter 4 – Versions, algorithms and options, Table 7 – Hash functions for bulk encryption and the generation of random numbers.	
B2-5	All supported algorithm selections are Good, or the algorithm selections are chosen by the server using the prescribed ordering.
See Chapter 4 – Versions, algorithms and options, section Prefer faster and safer algorithms.	

Certificates

TLS allows the server to prove its identity using an X.509 certificate. The client can only establish through a certificate that it communicates with the server and not a third party intending to listen in or manipulate its communication. Acquiring and managing certificates is not part of these guidelines. See the section *Management of certificates* in the appendix *Further considerations* for pointers.

Number	*Guideline*
B3-1	The server offers a certificate for authentication.
B3-2	The signed fingerprint of the certificate has been created using a **Good, Sufficient** or **Phase out** algorithm for hashing (see “Hash functions for certificate verification”).
See Chapter 4 – Versions, algorithms and options, Table 3 – Hash functions for certificate verification.	
B3-3	If the server offers a certificate with an RSA key, the length of this key is **Good** or **Sufficient**.
See Chapter 4 – Versions, algorithms and options, Table 8 – RSA key size.	
B3-4	If the supplied certificate is not directly signed by a root CA, the server offers intermediate CA certificates that authenticate the path between the root CA and the supplied certificate.

Key exchange

The algorithm for key exchange specifies how a client and server determine a key for encrypted communication. Ephemeral Diffie-Hellman is a method to derive a temporary shared key between parties. There are variants of Diffie-Hellman based on finite fields (DHE) and elliptic curves (ECDHE). More information on the use of ephemeral keys can be found in the section *Forward Secrecy* in the appendix *Further considerations*.
These guidelines do not specify a minimum size for the secret parameter used in ephemeral Diffie-Hellman. This is generally hard-coded in the TLS software library or derived from a chosen group, not a setting an administrator can configure.

Number	*Guideline*
B4-1	If DHE is used for key exchange, the secret parameter is ephemeral, chosen uniformly at random[20] and of appropriate size for the chosen finite field.
B4-2	If ECDHE is used for key exchange, the secret parameter is ephemeral, chosen uniformly at random[20] and of appropriate size for the chosen group.

Elliptic curves

Computations with elliptic curves are said to happen ‘on’ an elliptic curve. The curve is the context for the computation. For secure communication, the choice of a suitable curve is necessary. Not every curve offers the same security.

Number	*Guideline*
B5-1	All elliptic curves used are **Good, Sufficient** or **Phase out**.
See Chapter 4 – Versions, algorithms and options, Table 9 – Supported elliptic curves.	

Note that B5-1 applies both to key exchange based on ECDHE and to digital signatures using ECDSA and EdDSA.

20 The NCSC advises against the use of variations on the TLS protocol that invalidate the security analysis of TLS 1.3 and its forward secrecy property by fixing a DH parameter in some way. At the time of writing the ETSI “Enterprise Transport Security (ETS)” specification (ETSI TS 103 523-3), previously known as “eTLS”, is one such example. The NCSC factsheet “TLS Interception” treats considerations and preconditions for the deployment of TLS interception using TLS proxies.

Finite fields

Computations with finite fields are said to happen 'in' a finite field. The finite field is the context for the computation. For secure computation, the choice of a suitable finite field is necessary. Not every finite field offers the same security, interoperability or efficiency.

Number	Guideline
B6-1	All finite fields used are **Good**, **Sufficient** or **Phase out**.
See Chapter 4 – Versions, algorithms and options, Table 10 – Supported finite field groups.	

Other options

TLS has a plethora of options beyond those discussed. We only discuss options that influence the security of TLS and that are sometimes not secure by default.

Compression

The use of compression can give an attacker information about the secret parts of encrypted communication. Because data is first compressed and then encrypted, the effect of compression can yield information on the data that is sent.

Number	Guideline
B7-1	The settings for compression are **Good, Sufficient** or **Phase out**.
See Chapter 4 – Versions, algorithms and options, Table 11 – Compression.	

Renegotiation

Older versions of TLS (prior to TLS 1.3) allow forcing a new handshake. This is called renegotiation.

Number	Guideline
B8-1	The settings for renegotiation are **Good, Sufficient** or **Phase out**.
See Chapter 4 – Versions, algorithms and options, Table 12 – Insecure renegotiation; and Table 13 – Client-initiated renegotiation.	

0-RTT

0-RTT is an option in TLS 1.3 that transports application data during the first handshake message. 0-RTT does not provide protection against replay attacks at the TLS layer and is therefore hard to use securely in an application agnostic environment.

Number	Guideline
B9-1	The settings for 0-RTT are **Good, Sufficient** or **Phase out**.
See Chapter 4 – Versions, algorithms and options, Table 14 – 0-RTT.	

Scheduling removal of *Phase out* configurations

Phase out settings are known to be fragile with respect to evolving attack techniques. They provide a slim security margin relative to their **Sufficient** or **Good** counterparts. They are at a higher risk of becoming **Insufficient** in the near future.
The use of **Phase out** settings should be subject to written deprecation conditions.

Number	Guideline
B10-1	Any supported configuration that is **Phase out** has a documented condition that schedules its removal.
B10-2	Any configuration that is **Phase out** is not used past the documented condition that schedules its removal.

There is value in removing **Phase out** configurations when no longer required, because removed configurations can no longer be attacked or be used to attack other parts of TLS. At the same time, compatibility requirements for some applications may require their support until client support improves.

The NCSC advises not to support **Phase out** settings indefinitely.[21] Document what requires their use and include conditions that allow for removal when met. Thus, removal is scheduled upon use.

Choosing when to cease support is application specific. The web ecosystem deprecates old TLS versions and configurations faster[22] than the e-mail ecosystem.[23] These guidelines are application agnostic and therefore only contain generic advice.

The following are examples of documented conditions. **Phase out** configurations A, B and C will be removed:
- on <date>;
- with the release of <web browser> version X in the stable release channel;
- when the relative number of users drops below Y%;
- when the absolute number of users drops below Z per month.

21 Supporting outdated (client) software is also problematic for reasons unrelated to TLS: it is more likely to contain known security vulnerabilities that have been patched in later versions.

22 For example: by summer 2020 all modern web browsers had disabled support for Phase out configurations suchs as TLS 1.0, TLS 1.1 and DHE.

23 For example: removing Phase out settings may cause your mail servers to exchange unencrypted e-mail with mail servers outside your control. Retain the Phase out settings and document appropriate conditions for removal if your statistics show this to be the case.

4 Versions, algorithms and options

This chapter treats TLS versions, algorithms, key size & choice of groups and options. The security of a configuration depends on the selections in each of these categories.

The numbers in parentheses refer to the references on the last page of this document.

Versions

Recent versions of TLS are more secure than older versions. The oldest three versions of TLS, SSL 1.0, SSL 2.0 and SSL 3.0 cannot be used securely. The most recent version of TLS, TLS 1.3 offers the best protection.

Version	*Status*
TLS 1.3	**Good (3; 4)**
TLS 1.2	**Sufficient (3; 4)**
TLS 1.1	**Phase out (3; 4)**
TLS 1.0	
SSL 3.0	**Insufficient (3; 4)**
SSL 2.0	
SSL 1.0	

Table 1 – Versions

Cryptographic algorithms

The security of a TLS connection depends on the algorithms that are configured. The guidelines to determine algorithm selections consist of guidelines in four domains:

1. Certificate verification
2. Key exchange
3. Bulk encryption
4. Hashing

The first three domains are covered in their own section. Hashing is used as building block and covered within the other domains.

Figure 2 summarizes algorithm selections and their security level and shows the correspondence between algorithm selection and cipher suite notation in TLS 1.2 and TLS 1.3. The security levels apply to algorithm selections as follows.

Good, Sufficient and Phase out

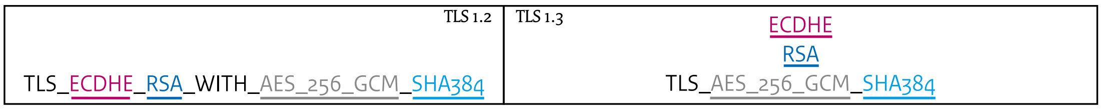
A **Good** algorithm selection is an algorithm selection that consists of **Good** choices for each of the domains. **Good** algorithms offer a security equivalent of at least 128 bits. **Good** algorithms by definition meet the guidelines B2-1 up to B2-4. See the row labelled **Good** in Figure 2 for examples of combinations that result in a **Good** algorithm selection.

A **Sufficient** algorithm selection is a selection that consists of **Sufficient** choices and possibly **Good** choices for each of the domains. **Sufficient** algorithms by definition meet the guidelines B2-1 up to B2-4. See the row labelled **Sufficient** in Figure 2 for examples of combinations that result in a **Sufficient** algorithm selection (possibly combined with choices from the row **Good**). Both **Good** and **Sufficient** algorithm selections only contain key exchange algorithms that provide forward secrecy.

A **Phase out** algorithm selection is a selection that consists of **Phase out** choices and possibly **Good** or **Sufficient** choices for each of the domains. See the row labelled **Phase out** in Figure 2 for examples of combinations that result in a **Phase out** algorithm selection (possibly combined with choices from the rows **Good** or **Sufficient**).

Prefer faster and safer algorithms

The NCSC advises to configure the server to prefer **Good** over **Sufficient** over **Phase out** algorithm selections. This prioritizes the fastest and safest algorithms. Choose your own ordering within the different security levels. This is chiefly a performance consideration.



	Key exchange	Certificate verification	Bulk encryption	Hashing
Good	ECDHE	ECDSA RSA	AES_256_GCM CHACHA20_POLY1305 AES_128_GCM	(HMAC-)SHA-384 (HMAC-)SHA-256
Sufficient	DHE		AES_256_CBC AES_128_CBC	(HMAC-)SHA-1
Phase out	RSA*		3DES-CBC⁺	

* Written as TLS_RSA_WITH_...

⁺ Written as 3DES_EDE_CBC or DES_CBC3

Figure 2 – Cipher suite notation in TLS 1.2 and TLS 1.3. The table summarizes algorithm selections and their security level.
Not included in the (old) cipher suite notation are: versions; hash functions for certificate verification; hash functions for key exchange; key sizes & choice of groups; and options. These can be found in their respective sections. For ordering, refer to the section Prefer faster and safer algorithms.

When the server only supports **Good** algorithm selections, it may honor the client's preference and does not need to enforce its own ordering.

Algorithms for certificate verification

The verification of certificates makes use of digital signatures. To guarantee the authenticity of a connection, a trustworthy algorithm for certificate verification must be used. The algorithm that is used to sign a certificate is selected by its supplier.

The certificate specifies the algorithm for digital signatures that is used by its owner during the key exchange. It is possible to configure multiple certificates to support more than one algorithm.

Algorithm	*Status*
ECDSA	Good **(2; 3)**
RSA	
DSS[24]	Insufficient
EXPORT-variants	
PSK	
Anon	
NULL	

Table 2 – Algorithms for certificate verification

The algorithm EdDSA is **Good**, but not (yet) permitted for use by suppliers of certificates (1) and therefore not added to the table.

Hash functions for certificate verification

The digital signatures on certificates use hash functions. The security of the chosen hash function is important for this purpose.

Algorithm	*Status*
SHA-512	Good **(1; 3)**
SHA-384	
SHA-256	
SHA-1	Insufficient **(1; 3)**
MD5	

Table 3 – Hash functions for certificate verification

Algorithms for key exchange

A TLS connection starts with a key exchange to establish a session key. Key exchange algorithms offering forward secrecy provide confidentiality of past communications in the event of secret key compromise. Static key exchange algorithms use the public key embedded in the certificate to transport an encrypted copy of the session key. Key exchanges based on ECDHE and DHE provide forward secrecy. Key exchanges based on static RSA, ECDH and DH keys in certificates do not.[25]

24 The algorithm DSS has been uncommon for a long time. It is **Insufficient** because rarely used code sees less testing and has a higher chance to contain undiscovered vulnerabilities.

25 Confidentiality under static RSA, ECDH and DH relies on keeping long term keys secret. An attacker can steal this long term key in the future and break confidentiality of past traffic. Under ECDHE and DHE, keys are kept for a very short term and are then destroyed, which leaves nothing to be stolen.

Algorithm	*Status*
ECDHE	Good **(3)**
DHE	Sufficient[26]
RSA	Phase out **(2; 3)**
DH[27]	Insufficient
ECDH[27]	
KRB5	
NULL	
PSK	
SRP	

Table 4 – Algorithms for key exchange

Use ECDHE over DHE

Elliptic curve cryptography is now widely deployed. Cryptography based on elliptic curves (ECDSA, ECDHE) is the fastest choice for TLS servers.

Before the broad availability of *elliptic curve cryptography*, DHE was the only mechanism to achieve forward secrecy in TLS. With the availability of elliptic curve Diffie-Hellman ephemeral (ECDHE) this is no longer the case. These guidelines advise the use of ECDHE over DHE for performance reasons.

Hash functions for key exchange[28, 29]

The certificate owner uses a digital signature during the key exchange to prove ownership of the secret key corresponding to the certificate. The owner creates this digital signature by signing the output of a hash function. The use of a secure hash function is important for this purpose.

Note: The following table differs from the others in this chapter by the reversed effect of **Phase Out**. It encourages support of modern hash functions instead of prohibiting weaker alternatives, because these are required in TLS 1.2.

SHA2 support for signatures	*Status*
Yes (SHA-256, SHA-384 or SHA-512 supported)	Good **(3; 4)**
No (SHA-256, SHA-384 or SHA-512 ***not*** supported)	Phase out **(3; 4)**

Table 5 – Hash functions for key exchange

Changes to RSA for digital signatures

The digital signature scheme to prove possession of an RSA secret key has been replaced in TLS 1.3. A modern padding scheme (RSA-PSS) replaces the previous RSA-PKCS#1 v1.5 padding scheme. This improvement applies retroactively: it is required for servers that support both TLS 1.2 and TLS 1.3. Use the latest available version of your TLS software library to make use of these improvements.

Algorithms for bulk encryption[31]

During the application phase, (data) records are encrypted in bulk by a symmetric encryption algorithm. A symmetric encryption algorithm usually consists of a cipher and a mode of operation. Algorithms that tightly integrate authentication and encryption in one mode of operation are to be preferred. These so-called AEAD algorithms include AES-GCM and ChaCha20-Poly1305. The most popular symmetric encryption algorithm is AES. TLS supports AES with two key sizes (128 and 256 bits), while ChaCha20-Poly1305 supports a single (256 bits) key size.

26 DHE is classified **Sufficient** and not **Good** because it is very slow when secure parameters are used.

27 Static (EC)DH requires special certificates and is seldom used in TLS. Their use is **Insufficient** because rarely used code sees less testing and has a higher chance to contain undiscovered vulnerabilities.

28 This section covers the use of hash functions for key confirmation and handshake integrity. The use of hash functions for key derivation is covered under *Hash functions for bulk encryption and the generation of random numbers*.

29 These are the most common algorithms for hashing. SHA-224 is also **Sufficient** but not used much.

30 Note that SHA2 support for signatures is generally a property of a TLS software library, not a configuration. An update to your library may be required if you do not support the use of SHA2 for key exchange.

31 These are the most common algorithms for bulk encryption. Other **Good** algorithms are AES-{256,128}-CCM. CAMELLIA is **Sufficient**. Other **Phase out** algorithms include SEED and ARIA. These algorithms are rarely used. If a system supports these algorithms it is worth checking if this is necessary.

Algorithm	Status
AES-256-GCM	Good (3)
ChaCha20-Poly1305	
AES-128-GCM	
AES-256-CBC	Sufficient (4)
AES-128-CBC	
3DES-CBC	Phase out[32] (2; 3)
AES-256-CCM_8[33]	Insufficient
AES-128-CCM_8[32]	
IDEA	
DES	
RC4	
NULL	

Table 6 – Algorithms for bulk encryption

Hash functions for bulk encryption and the generation of random numbers
Some algorithms for bulk encryption use hash functions to provide authenticity (MAC).[34] The security of the chosen hash function is important for this purpose.

TLS also uses the selected hash function as a component in the generation of random numbers (PRF). The security of the chosen hash function is also important for this purpose.

Algorithm	Status
HMAC-SHA-512	Good (3; 4)
HMAC-SHA-384	
HMAC-SHA-256	
HMAC-SHA-1	Sufficient (3; 4)
HMAC-MD5	Insufficient (3)

Table 7 – Hash functions for bulk encryption and the generation of random numbers

32 3DES-CBC is a legacy cipher with a 64 bit block size. Faster and safer alternatives are widely available and 3DES-CBC is seldom necessary for compatibility. The limited block size makes 3DES-CBC vulnerable under very specific circumstances (Sweet32-attack). 3DES-CBC is the only algorithm for bulk encryption available in TLS 1.0 that is not **Insufficient**. Once you stop supporting TLS 1.0 you should disable 3DES-CBC.

33 AES-CCM_8 is a variant of AES-CCM with a truncated authentication tag, which has a lower security equivalent for integrity protection.

34 AEAD algorithms tightly integrate authentication and do not use a separate hash function for record protection. When a cipher suite that includes an AEAD refers to a hash function, it is only used as a component in the generation of random numbers.

35 These are the most common algorithms for hashing. SHA-224 is also **Sufficient** but not used much.

36 Other documents may refer to these algorithms without the HMAC- prefix.

Caution: SHA-1 is only **Sufficient** for bulk encryption and as a component in the generation of random numbers. It is **Insufficient** for use in digital signatures on certificates, as stated in the section *Hash functions for certificate verification*. Its use as part of the key exchange without supporting newer alternatives is also **Insufficient**, as stated in the section *Hash functions for key exchange*.

Key sizes and choice of groups

RSA key size
The security of RSA for encryption and digital signatures is tied to the key length of the public key.[37]

Length of RSA-keys	Status
At least 3072 bit	Good (2; 3)
2048 – 3071 bit	Sufficient (2)
Less than 2048 bit	Insufficient

Table 8 – RSA key size

Supported elliptic curves[38]
Not all elliptic curves are suitable for use in TLS. The security of ECDSA and EdDSA digital signatures and the ECDHE key exchange depend on the chosen curve. The table lists the most commonly used curves.

Elliptic curve	Status
secp384r1	Good (3; 4)
secp256r1	
curve 448	
curve 25519	
secp224r1	Phase out (3; 4)
Other curves	Insufficient

Table 9 – Supported elliptic curves

Supported finite field groups
The security of the Diffie-Hellman ephemeral key exchange (DHE) depends on the lengths of the public and secret keys used within the chosen finite field group. The larger key sizes required for the use of DHE come with a performance penalty. Carefully evaluate and use ECDHE instead of DHE if you can.

37 It is tied to the public modulus, which is part of the public key.

38 These are the most common elliptic curves in TLS. Secp521r1 is **Good**. The curves brainpoolP512r1, brainpoolP384r1 and brainpoolP256r1 are **Sufficient**. These elliptic curves are rarely used in TLS and therefore not added to the table. If a system supports these curves it is worth checking if this is necessary.

Standardized finite field groups

These guidelines advise the use of standardized groups. Larger groups are chosen to mitigate against the risk of precomputation by attackers.

This conservative approach enlarges the performance penalty that comes with the use of DHE. Carefully evaluate and use ECDHE instead of DHE if you can.

The complexity associated with free choice of finite field groups has been a source of vulnerabilities in the past. The TLS 1.3 specification only includes a limited number of finite field groups for DHE to reduce this complexity. These guidelines limit **Sufficient** groups for TLS to those used in TLS 1.3 (and specified in RFC 7919).

Finite field group	*Status*
ffdhe4096 (RFC 7919)	**Sufficient**[39] **(4)**
ffdhe3072 (RFC 7919)	
ffdhe2048 (RFC 7919)	**Phase out**
Other groups	**Insufficient**

Table 10 – Supported finite field groups

Options

Compression

The use of compression can give an attacker information about the secret parts of encrypted communication. An attacker that can determine or control parts of the data sent can reconstruct the original data by performing a large number of requests. Data that contains more repetitions results in better compression than data without repetitions. An attacker can thus establish if a known part occurs more often in the data sent. This way, use of compression can negatively impact security.

TLS compression is used so rarely that disabling it is generally not a problem. This is different for application-specific compression. For example, compression in the HTTP protocol is commonly used to make more efficient use of available bandwidth.

Consider the trade-offs involved with application-level compression. If you choose to use application-level compression, verify if it is possible to mitigate related attacks at the application level. An example of such a measure is limiting the extent to which an attacker can influence the response of a server.

Compression option	*Status*
No compression	**Good**
Application-level compression	**Sufficient**[40]
TLS compression	**Insufficient**[41]

Table 11 – Compression

Renegotiation

Older versions of TLS (prior to TLS 1.3) allow forcing a new handshake. This so-called renegotiation was insecure in its original design. The standard was repaired and a safer renegotiation mechanism was added. The old version is since called *insecure renegotiation* and should be disabled.

Allowing clients to initiate renegotiation is generally not necessary and opens a server to DoS-attacks inside a TLS connection. An attacker can perform similar DoS-attacks without client-initiated renegotiation by opening many parallel TLS connections, but these are easier to detect and defend against using standard mitigations.

Insecure renegotiation	*Status*
Off[42] (or N/A for TLS 1.3)	**Good**
On	**Insufficient**

Table 12 – Insecure renegotiation

Client-initiated renegotiation	*Status*
Off (or N/A for TLS 1.3)	**Good**
On	**Sufficient**

Table 13 – Client-initiated renegotiation

39 These groups are **Sufficient** and not **Good** solely because they are slow. DHE key exchange with larger group sizes is significantly more resource intensive than a security equivalent ECDHE exchange. The groups ffdhe6144 and ffdhe8192 (RFC 7919) are also **Sufficient**, but even slower.

40 The use of application specific compression (such as HTTP compression) results in a TLS configuration that is vulnerable to the BREACH attack. Disabling application specific compression may have a negative effect on system performance.

41 The use of TLS compression results in a TLS configuration that is vulnerable to the CRIME attack.

42 Some call this “secure renegotiation”.

0-RTT

Old versions of TLS use a minimum of two round trips of communication between client and server before application data is transported. TLS 1.3 halves this overhead by requiring only a single round of communication. 0-RTT is an option in TLS 1.3 that further reduces this overhead. It transports application data during the first handshake message. The trade-off is that 0-RTT does not provide protection against replay attacks at the TLS layer and is therefore hard to use securely in an application agnostic environment.

Off (or N/A prior to TLS 1.3)	**Good**
On	**Insufficient**

Table 14 – 0-RTT

OCSP stapling

The TLS client can verify the validity of the X.509 certificate presented by the server by contacting the certificate supplier using the OCSP-protocol. OCSP provides a certificate supplier with information on clients communicating to the server: this may be a privacy risk. A server can also provide OCSP-responses itself (OCSP stapling). This solves this privacy risk, does not require connectivity between client and certificate supplier and is faster.

OCSP stapling	*Status*
On	**Good**
Off	**Sufficient**

Table 15 – OCSP stapling

Appendix A – Further considerations

Forward secrecy

Forward secrecy is a mechanism to maintain the confidentiality of past TLS communications in the event the secret key of a certificate is stolen from the server. Configurations that use ECDHE or DHE for key exchange, offer forward secrecy.

If forward secrecy is used, client and server do not directly use their own keys for bulk encryption. A second, temporary, (ephemeral) key is agreed upon instead that is only used for that session. All values used are destroyed afterwards. The ephemeral key used cannot be derived from the secret key of the certificate. Without forward secrecy, the server's keys (corresponding to its certificate) are used to exchange session keys directly.

The scenario that forward secrecy protects against consists of two steps. First, an attacker needs to eavesdrop on the communication that is protected by TLS. Then, the attacker needs to obtain the secret key corresponding to the public key in the certificate, for example by hacking or the use of a court order. With access to the secret key, an attacker can recover the session key, decrypt the encrypted traffic and thus break the confidentiality of communications.

Session tickets

In many applications, it is common for the client and server to (re)establish more connections after an initial TLS connection has been set up. TLS offers mechanisms for session resumption to allow a client and server to skip the handshake and go straight to the application phase. This reduces the cost of TLS session setup for additional connections.

With Session IDs, both client and server save session state referenced by an ID. The client presents that ID when resuming a connection. Using this Session ID both client and server retrieve the corresponding state and proceed directly to the application phase.

Session tickets are much like session IDs. A session ticket is an encrypted copy of the session state. By asking the client to keep a session ticket, the server no longer needs to store session ID and state for each client. The client retains the session ticket and presents it to the server when resuming a connection. The server decrypts the session ticket, recovers the session state and the TLS connection proceeds directly to the application phase. The server needs to keep a 'session ticket encryption key' for this purpose.

The design of session tickets in TLS 1.0, 1.1 and 1.2 is fragile.[43] An attacker that steals the session ticket encryption key can do passive decryption on all connections that exchange or use session tickets. This also breaks the forward secrecy property described previously.

These issues have been corrected in TLS 1.3. The NCSC advises organisations that want to speed up session resumption to use TLS 1.3. If session tickets are used in older version of TLS, the 'session ticket encryption key' should not be stored on disk and rotated frequently.

Random number generators

To ensure the security of cryptographic algorithms, a good source of entropy and a good generator of random numbers (pseudo random number generator, PRNG) are required. The source of entropy supplies random data and is input to the PRNG. The PRNG turns this random data into uniformly distributed random numbers. In particular, the requirement for randomness is relevant for (but not limited to):

- the generation of keys for certificates;
- the generation of temporary keys and proof of secret key possession for forward secrecy.

43 Drew Springall, Zakir Durumeric, and J. Alex Halderman. "Measuring the security harm of TLS crypto shortcuts." *Proceedings of the 2016 Internet Measurement Conference*. ACM, 2016. https://jhalderm.com/pub/papers/forward-secrecy-imc16.pdf

Generating sufficient entropy may be a bottleneck if a TLS server is under heavy load. The addition of a hardware module (hardware random number generator) to the server ensures the availability of sufficient entropy. Many modern processors integrate a hardware module to harvest randomness.

Most operating systems and TLS software libraries contain a good random number generator. You can ask the supplier of your chosen TLS library (or the vendor that embeds it) about the random number generator that is used and its source of entropy.

It is known that the following random number generator is insecure:

- Dual EC DRBG[44, 45]

It is advisable to check that your TLS software library does not use this insecure random number generator.

Certificate management

Acquiring and managing certificates is not part of these guidelines. However, good certificate management is an important condition for the safe deployment of TLS. That is why we list some points of particular interest. Further instructions can be found in the factsheet "Secure administration of digital certificates" by the NCSC.[46]

- **Secret key generation** Use a good random number generator to generate the secret key. Make sure you generate this key on a trusted system, for example a Hardware Security Module (HSM) or a computer that is physically disconnected from the internet. A key generated on a disconnected computer is then deployed on the server that will offer the certificate.
- **Certificate supplier** Choose a trustworthy certificate supplier to supply and sign the certificate. Organisations within the Dutch government may make use of certificates by PKIoverheid, and for certain applications are obligated to do so.
- **Domain names** The certificate contains a list of domain names (fully qualified domain names, FQDNs) that it applies to. Make sure the certificate covers all domains for which it is used, including subdomains.
- **Extended validation** Many certificate issuers also issue Extended Validation (EV) certificates. An EV certificate provides some assurance on the identity of the owner. Developers of client software have been removing the prominent visual distinctions between EV and normal certificates. EV certificates are usually more expensive than a normal certificate. A risk analysis can help to determine if an EV certificate is warranted.
- **Files on the server** The administrator of the server needs to include intermediate CA's between the root CA and the issued certificate on the server. The server offers these during TLS connections. The secret key needs to be adequately protected. An attacker that can obtain the secret key can read or manipulate intercepted traffic. A secret key can be stored in an HSM. An HSM is designed to offer physical protection against extraction of the secret key.
- **Administration** Keep an administration of all certificates that are in use within the organisation. If a certificate needs to be replaced, this will make it easier to determine where it is in use. Include the expiration time for all certificates, to ensure timely replacement. Expired certificates should never be used. Some certificate issuers support mechanisms for automated renewal and deployment of certificates, which may reduce the potential for human error.

Where does a TLS connection terminate?

The model of a client that connects to a server does not correspond to a setup that many organisations use. For example, decryption of TLS traffic can be centralised before it is further routed within an internal network. This setup enables the post-processing of network traffic. Keep in mind that TLS in such a setup only protects up to the point where the connection terminates. Take additional measures if confidentiality and integrity need to be guaranteed within the organisation. A possible measure is to use a new TLS session for this last leg.

Some organisations that provide DDoS mitigation services ask for the secret keys of certificates that are used for TLS. They then proceed to terminate and filter your traffic. If you want to make use of these services, do not simply provide your secret key. This may be in violation of internal policy or (sectoral) law. Consider switching to a supplier that does not require you to hand over keys.[47] Identify the risks associated with handing over secret keys. Take contractual measures to compensate for the decreased technical control and audit the supplier to assess the extent to which the supplier implements the agreed upon measures.

44 https://csrc.nist.gov/publications/nistbul/itlbul2013_09_supplemental.pdf

45 https://projectbullrun.org/dual-ec/

46 https://www.ncsc.nl/documenten/factsheets/2019/juni/01/factsheet-veilig-beheer-van-digitale-certificaten

47 See the following as an example of a method to achieve this: https://blog.cloudflare.com/keyless-ssl-the-nitty-gritty-technical-details/

Post-quantum security

Regular use of TLS is not post-quantum secure.[48] It is possible to configure TLS to provide limited[49] post-quantum security. The NCSC advises to get specialist support for use cases that have this requirement. For the Dutch government, such support is available from the national communication security agency NBV.

Authenticating clients with certificates

TLS supports mutual authentication with certificates. In mutual authentication, the client uses a certificate to authenticate itself to the server in addition to the server using a certificate to authenticate itself to the client.

Client certificates often contain sensitive information, such as personal data. An example is the name of the user. Before TLS 1.3, certificates were transmitted unencrypted as part of the handshake. If you use certificates that contain sensitive data for authentication and rely on TLS to provide confidentiality, it is recommended to use TLS 1.3.

Certificate pinning and DANE

A client that starts a TLS session with a server, checks the X.509 certificate of the server. The client checks the chain of digital signatures that connects the certificate to the root CA. This PKI system is fragile because most software trusts hundreds of root CAs. If a certificate supplier issues forged certificates the integrity of the entire system is at risk.

Are you in control over software of both client and server? Then, certificate pinning enables you to pin just the certificate(s) that the client should accept. The client no longer has to check the chain of signatures: it recognizes the certificate or it does not. A compromised certificate authority is no longer a risk for the connection. Alternatively, use of certificates supplied by one particular CA can be whitelisted by pinning the intermediate or root certificate that are used to issue them. This way, the compromise of other CAs is no longer a risk. The connection between an app on a mobile platform and a server is a scenario where certificate pinning can be effective. Account for the propagation delay of changes in certificate pinning. Clients will refuse to connect to your server after a (rapid) change of certificate(s) if a pin cannot be updated in time.

DNS-based Authentication of Named Entities (DANE) is a technique to enable clients to authenticate a certificate based on the Domain Name System (DNS). The administrator of the certificate publishes information about that certificate in a special DNS record, a TLSA record. This allows clients to verify the authenticity of the certificate using the PKI, but also using the TLSA record. Note that the traditional DNS system is not trustworthy enough to use DANE. The use of DNSSEC is necessary. An example of the use of DANE is to authenticate TLS connections between e-mail servers.[50]

48 More information about the implications of quantum computers for organizations can be found in the factsheet "Post-quantum cryptography". https://english.ncsc.nl/publications/factsheets/2019/juni/01/factsheet-post-quantum-cryptography

49 At the time of writing, such configurations lack post quantum forward secrecy.

50 More information on DANE and this use case can be found in the factsheet "Secure the connections of mail servers". https://english.ncsc.nl/publications/factsheets/2019/juni/01/factsheet-secure-the-connections-of-mail-servers

Appendix B – Changes to these guidelines

These guidelines will be updated for new versions of TLS and new insights into the (in)security of certain configurations. The security of TLS is the continuous subject of research. The coming years will bring the discovery of additional vulnerabilities. In addition, additional versions or configurations of TLS will be standardized.

Validity

The latest version of these guidelines can always be found on the website of the NCSC. The NCSC periodically evaluates the validity of its advice. These guidelines are valid unless updated and do not expire.

Critical changes

If an immediate change of these guidelines is required, it will be published as an addendum to the most recent version of the guidelines. This may occur if research shows that certain TLS configurations have become insecure.
An addendum will be published on the website of the NCSC and communicated to NCSC partners. A publication of an addendum will also be announced using the Twitter account of the NCSC (@ncsc_nl) or a then suitable medium.

New versions

Larger changes will be implemented in newer versions of these guidelines. A new version of the guidelines includes information that was part of earlier addenda. New versions will be distributed in the same manner as addenda: published on the website of the NCSC, sent to NCSC partners and using the NCSC Twitter account.

Appendix C – List of cipher suites

Using TLS the client and server agree on an algorithm selection to use in subsequent encrypted communication.
An algorithm selection is a set with four elements, one cryptographic algorithm for key exchange, one cryptographic algorithm for digital signatures, one cryptographic algorithm for bulk encryption and one cryptographic algorithm for hashing. In TLS 1.3, the set containing only these last two elements is known as a cipher suite. Prior to TLS 1.3, a cipher suite referred to what these guidelines call an algorithm selection.

This appendix provides an overview of cipher suites and their security level. This notation can be useful when configuring software. Refer to Figure 1 in the chapter *Guidelines* illustrates the cipher suite notation in TLS prior to TLS 1.3.

Not included in the cipher suite notation are: *versions; hash functions for certificate verification; hash functions for key exchange; key sizes & choice of groups*; and *options*. These can be found in their respective sections in the chapter *Versions, algorithms and options*. For ordering, refer to the section *Prefer faster and safer algorithms* in the same chapter.

Good

TLS_AES_256_GCM_SHA384[51]
TLS_CHACHA20_POLY1305_SHA256[51]
TLS_AES_128_GCM_SHA256[51]

Sufficient

TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384[52]
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256[52]
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256[52]
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384[52]
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256[52]
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256[52]
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
TLS_DHE_RSA_WITH_AES_256_CBC_SHA
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
TLS_DHE_RSA_WITH_AES_128_CBC_SHA

Phase out

TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_256_GCM_SHA384
TLS_RSA_WITH_AES_128_GCM_SHA256
TLS_RSA_WITH_AES_256_CBC_SHA256
TLS_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA256
TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA

51 When used in combination with Good algorithms for key exchange and certificate verification. See Figure 2 in chapter Versions, algorithms and options for an example that results in a lower security level.

52 These algorithm selections, combined with TLS 1.3 are **Good**. However, the (old) ciper suite notation used here will frequently result in the use of at most TLS 1.2 in software, which is **Sufficient**.

Appendix D – Glossary

Glossary	
3DES	See Bulk encryption
AEAD	Algorithms for Bulk encryption that provide Authenticated Encryption with Associated Data (AEAD) tightly integrate authentication and encryption. Popular AEAD algorithms for Bulk encryption include AES-GCM and ChaCha20-Poly1305.
AES	See Bulk encryption
Algorithm selection	An algorithm selection is a set with four elements, one cryptographic algorithm for key exchange, one cryptographic algorithm for digital signatures, one cryptographic algorithm for bulk encryption and one cryptographic algorithm for hashing. In TLS 1.3, the set containing only these last two elements is known as a cipher suite. Prior to TLS 1.3, a cipher suite referred to what these guidelines call an algorithm selection. Using TLS the client and server agree on an algorithm selection to use in subsequent encrypted communication. See key exchange, digital signature, bulk encryption, hash function and cipher suite.
Bulk encryption	Bulk encryption is the process during the application phase whereby data is enciphered using the key established during key exchange. Encipherment uses an algorithm for symmetric encryption. Well-known examples of such algorithms are AES-GCM, ChaCha20 and 3DES-CBC.
CA	See Certificate verification
CAMELLIA	See Bulk encryption
CBC	See Mode of operation
CCM	See Mode of operation
Certificate	See Certificate verification
Certificate verification	The server offers a certificate to the client during a TLS session. This certificate is digitally signed by a certificate authority (CA). The certificate authority is a trusted entity for the client. The client verifies the digital signature by the certificate authority. This establishes whether the certificate has been issued by the certificate authority. The algorithm for certificate validation is the algorithm the certificate authority uses for its digital signature. Well-known examples are RSA and ECDSA.
ChaCha20-Poly1305	See Bulk encryption
Cipher suite	A cipher suite contains an algorithm for bulk encryption and an algorithm for hashing. Cipher suites before TLS 1.3 also include the algorithms for key exchange and digital signatures. These guidelines follow the new narrow TLS 1.3 definition for cipher suite. See algorithm selection.

Glossary	
DANE	DNS-based Authentication of Named Entities (DANE) is a technique to enable clients to authenticate a certificate based on the Domain Name System (DNS).
(D)DoS attack	A Denial of Service (DoS) attack is an attack whereby a computer is made unavailable, for example by means of a flood of requests. The requests originate from a single computer system. In a Distributed Denial of Service (DDoS) attack, requests originate not from a single but from a large number of computer systems.
DES	See Bulk encryption
DHE	See Key exchange
Diffie-Hellman	See Key exchange
DNS	The Domain Name System (DNS) is a distributed system to answer queries for information on domain names. A typical query may concern the IP-address of a computer corresponding to a domain name, or the computer that handles the e-mail for a particular domain name. DNS Security Extensions (DNSSEC) can improve the trustworthiness of information in DNS. DNSSEC also enables the use of DNS for new purposes, such as DANE.
DNSSEC	See DNS
DSS	See Certificate verification
ECDHE	See Key exchange
ECDSA	See Certificate verification
ECC	See Elliptic curve
EdDSA	See Certificate verification
Elliptic curve	See Mathematical structure
Finite field	See Mathematical structure
Forward Secrecy	Forward secrecy is a mechanism to maintain the confidentiality of past TLS communications in the event the secret key of a certificate is stolen. Cipher suites that employ key exchanges based on ECDHE or DHE offer forward secrecy.
GCM	See Mode of operation
Handshake	The handshake is the phase of the TLS protocol where client and server agree on the way data is to be exchanged. The handshake phase is followed by the application phase where client and server exchange enciphered data.
Hash function	A hash function is a mathematical function that maps input data into a digital fingerprint. In general, the input is no longer retrievable from the result. Hash functions are used in TLS as a component in digital signatures, the generation of random numbers (PRF) and for bulk encryption (MAC). Examples of hash functions include MD5, SHA-1 and SHA-256.
Hashing	See Hash function
https	HTTP Secure (https) is a protocol that consists of the establishment of a TLS session that is then used to exchange HTTP traffic. This way communication with a webserver is protected against eavesdropping and manipulation in transit.
IETF	The Internet Engineering Task Force (IETF) is an organization that designs open standards for the internet. The standards are documented in so-called Requests for Comments (RFC's). The IETF does not have authority to enforce the use of standards that they design.
Key	A key is some secret data used for cryptographic computations. Encrypted data can be decrypted using the corresponding key. In symmetric algorithms for encipherment, the entire key is secret. In asymmetric algorithms for encipherment, the key consists of two parts: a public part and a secret part. The public part of the key is called the public key. This part is not kept secret. The secret part is called the secret key.

Glossary	
Key exchange	The client and server in a TLS session need a key to start bulk encryption of data. Exchanging a key happens using an algorithm for key exchange. A special algorithm is necessary because the connection is not yet encrypted during the handshake. Examples of algorithms for key exchange are DHE, ECDHE and RSA.
Mathematical structure	Elliptic Curves and Finite fields are mathematical structures useable for computation. Another example of such a mathematical structure is the set of integers. Elliptic curves can be used for cryptography, so called Elliptic Curve Cryptography (ECC). EdDSA, ECDSA and ECDHE are algorithms based on ECC. Finite fields can also be used for cryptography. DHE is an algorithm based on finite field cryptography.
MD5	See Hash function
Mode of operation	An algorithm for bulk encryption can act on blocks of data (block cipher) or on a stream of data (stream cipher). When using a block cipher, the enciphered blocks need to be safely joined together. The mode of operation is the way these blocks are joined. Examples of modes of operations are CBC and GCM.
PKI	See Public Key Infrastructure
Secret key	See Key
Public key	See Key
Public Key Infra-structure	A Public Key Infrastructure (PKI) is a hierarchical ordering of certificates where the higher certificates vouch for the authenticity of lower certificates using a digital signature. If a client trusts the highest certificates in the PKI, then it can also trust the lower certificates by checking the signatures on the intermediate certificates. The certificates that a certificate authority issues together with the root certificate together form a PKI.
RSA	RSA is an algorithm for key exchange and certificate verification. See both topics.
Secret key	See Key
Security equivalent	The security equivalent is a measure to compare the cryptographic strength of cryptographic systems. Security equivalents are expressed in bits. The strength of a cryptographic system depends on the algorithm used, the key length and the state of art of attacks on the algorithm. For example: ECDSA used with a key length of 256 bit and AES with a key length of 128 bit both have a security equivalent of 128 bit, based on the current understanding of cryptologic attacks on these algorithms.
SHA-1	See Hash function
SHA-256, SHA-384, SHA-512	See Hash function
Software library	A software library is software that provides functionality for use by programmers of other software. By using a software library, a programmer can build upon the work of others. This way, she does not have to build all functionality by herself. TLS use in software generally relies on a software library.
SSL	Secure Sockets Layer (SSL) is the old name for Transport Layer Security (TLS). Though no longer called SSL following the release of TLS 1.0 (1999), the acronym SSL remains popular.
VPN	A Virtual Private Network (VPN) is a network consisting of computers connected by non-trusted links. Encryption enables the trustworthy exchange of data between these computers.

References

1. **CA/Browser forum.** CA/Browser Forum Baseline Requirements. *CA-Browser Forum BR 1.7.3.* [Online] October 2020. https://cabforum.org/baseline-requirements-documents/
2. **Federal Office for Information Security (BSI).** Cryptographic Mechanisms: Recommendations and Key Lengths. *BSI TR-02102-1 v2020-01*, [Online] October 2020. https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.pdf?__blob=publicationFile&v=8
3. **ECRYPT-CSA.** Algorithms, Key Size and Protocols Report (2018), *H2020-ICT-2014 – Project 645421, D5.4*, [Online] February 2018. http://www.ecrypt.eu.org/csa/documents/D5.4-FinalAlgKeySizeProt.pdf
4. **Federal Office for Information Security (BSI).** Cryptographic Mechanisms: Recommendations and Key Lengths, Part 2 – Use of Transport Layer Security (TLS). *BSI TR-02102-2 v2020-01*, [Online] October 2020. https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-2.pdf?__blob=publicationFile&v=10

Colophon

Publication
National Cyber Security Centre (NCSC)
P.O. Box 117, 2501 CC The Hague
Turfmarkt 147, 2511 DP The Hague
+31 (70) 751 55 55

More information
www.ncsc.nl
info@ncsc.nl
@ncsc_nl

version 2.1 | January 2021