Security Recommendations for TLS

Table of Contents

Introduction

The TLS[1] protocol is one of the most widespread solutions for protecting network traffic. In this client–server model, the application data is encapsulated in such a way as to ensure confidentiality and integrity of the exchanges. The server is necessarily authenticated, and additional functions allow for authenticating the client when such a need has been identified.

Since the appearance of its predecessor SSL[2] in 1995, TLS has been adopted by many Internet stakeholders in order to secure traffic linked to websites and electronic messaging. This is furthermore a privileged solution for protecting internal infrastructure traffic. For these reasons, the protocol and its implementations are subject to constant research. Over the years, several vulnerabilities have been discovered, motivating the development of corrections and countermeasures in order to prevent compromising exchanges.

The TLS deployment that provides the most assurance in terms of security is therefore based on the use of up-to-date software, but also on adjusting the parameters of the protocol according to the context. The explanations provided by this guide are supplemented with several recommendations aimed at reaching a level of security that is compliant with the state of the art, in particular concerning the cipher suites to be retained.

---

[1]Transport Layer Security.
[2]Secure Sockets Layer.

Who Should Read This Guide?

The purpose of this guide is to serve as a common repository for recommendations that can be adapted according to the scope of each project attempting to comply, or making it possible to comply, with the good practices linked to the TLS protocol.

It is intended for all audiences who wish to become familiar with or interact with the TLS protocol: those responsible for the security of information systems, administrators of organisation of all sizes, or developers of solutions who want to secure the exchange of information through TLS.

Indeed, the state of a TLS connection is the result of a set of factors that are rarely under the full control of a single entity. This state depends globally:

- on the capacities of the TLS stack, in charge in particular of the logic controller and the cryptographic calculations, such as OpenSSL, GnuTLS or miTLS;
- on the capacities of the software that uses the TLS stack. For example, the Apache solution has `mod_ssl` and `mod_gnutls` modules to serve as HTTP[3] resources by protecting them with OpenSSL or GnuTLS;
- on the configuration of the previous software. For example, although the Apache software supports all of the versions SSLv3, TLS 1.0, TLS 1.1 and TLS 1.2 by default, the configuration options make it possible to never use SSLv3;
- on the capacities and the configuration of the TLS contact. For example, faced with a client that offers using TLS 1.2, an Apache server that authorises the use of TLS 1.0, TLS 1.1 and TLS 1.2 will choose to establish a TLS 1.2 session.

If there is no exact correspondence, the successive capacities of the preceding list are strictly inclusive. Software does not have more capacities than the TLS stack on which it is based. Likewise, a configuration file does not make it possible to deploy capacities that would not be supported by the software in question.

Controlling an element of this chain therefore makes it possible to exclude certain undesirable capacities. However, it is not always possible for an isolated entity to deploy or to use the optimum parameters with respect to security recommendations.

The following recommendations abstract themselves from this variety of participants by identifying the characteristics that are desirable for a TLS connection, independently of the responsibilities for implementation. This transposition work, of which the circumstances are too varied to be handled completely, are however the subject of additional publications (in French) from ANSSI [1, 2]. Common application examples are also shown at the end of the document, in Appendix B.

---

[3]Hypertext Transfer Protocol.

How to Read the Recommendations

This guide lists, through its recommendations, a set of preferred characteristics for a TLS connection. The recommendations primarily focus on the profile of the traffic exchanged and does not prejudge the means required for their implementation. The hierarchy adopted is specified in table 1.

The autonomy of the recommendations allows the various stakeholders of the TLS connection, whether direct or indirect, to apply them according to their respective positions. For example, **R4** recommends never using the version SSLv2. For an integrator, this will entail compiling the TLS stack used so that the software developed does not support SSLv2. For an administrator, if the software that he is using supports SSLv2, it will entail excluding this version using the configuration options that are accessible.

Furthermore, how a recommendation is interpreted varies according to the contexts. For example, **R3** recommends using only TLS 1.2, while **R3-** tolerates versions TLS 1.1 and TLS 1.0. Some integrators can choose to exclude from their software products, right from when the TLS pile is compiled, versions TLS 1.1 and TLS 1.0. For others, it may be necessary to include these downgraded versions in order to respond to the compatibility needs of certain clients, while allowing clients who comply more closely with the recommendations to exclude these versions using configuration options.

The reasoning, for the most part, indifferently concerns TLS clients and servers. In addition, it is independent of the nature of the application traffic protected. As such, the following recommendations apply to the HTTPS[4] traffic handled by public web servers as well as the infrastructure traffic protected by TLS within certain industrial control systems.

Rx	This recommendation makes it possible to set up the target architecture that offers a level of security that is compliant with the state of the art.
Rx-	In the case where applying the recommendation for optimum security is impossible, or insufficient with regards to needs in terms of compatibility, this measure offers a first level of derogation. The trust level is lower than with **Rx**.
Rx--	This derogation represents the lowest trust level permitted in order to remain compatible with the guide.

Table 1: Hierarchy of the recommendations

---

[4]HTTP Secure.

When TLS is deployed in an infrastructure with end-to-end control, the recommendations can be applied without restriction. For example, between **R3**, **R3–** and **R3--**, it is **R3** that should be followed: the server and its clients must be configured to use TLS 1.2, and reject any connection attempt with a prior version of the protocol.

In the distinct context of the configuration of a TLS server faced with several clients of which the capacities are not controlled, several compatibility issues arise. The ideal parameters may in fact be too restrictive. Typically, when trying to communicate with a public web server, several outdated client software programs risks not supporting a portion of the most recommended functions.

In such a situation, a profile of the clients that are able to connect to the server should be established, and consequently the permissibility of the configuration should be evaluated. Note that authorising certain deficient parameters generates a risk not only for outdated clients based on this choice, but also sometimes for the server and up-to-date clients. In particular, several vulnerabilities are based on the ability of an attacker to downgrade the security parameters of a connection in order to make use of weaker functions that may still be implemented [3, 4, 5].

Once this profile is established, the abilities of the various clients to conform to the security recommendations can be evaluated. In the case of web browsers, there are public resources that facilitate the connecting of versions of browsers with their security capacities linked to TLS [6, 7]. For example, most of the Microsoft Internet Explorer 8 clients will not be able to connect to a server that only offers TLS 1.2, and require activation of TLS 1.0 and certain associated cipher suites.

> **R1** Restrict compatibility according to the profile of the clients
>
> When the clients of a server are not controlled, a profile of desired clients should be established. The possible monitoring of downgraded recommendations (**Rx–** or **Rx--**) is based on this profile.

Chapter 1

Presentation of the TLS Protocol

1.1 Unfolding of the TLS Sessions

The development of the TLS protocol has followed several iterations [8, 9, 10] since the designing of the SSL protocol, which is now obsolete [11]. The IETF[5] is in charge of the standardisation process. The numerical values of the various parameters are referenced by IANA[6].

The messages transmitted by the intermediary of the TLS protocol are called *records*. They are generally encapsulated in TCP[7] segments, a protocol in charge of ensuring the network transport functionalities such as acknowledging the reception of data [12]. For datagram protocols, such as UDP[8], a DTLS[9] variant has been defined [13]. There are four types of records, explained hereinafter: `handshake`, `change_cipher_spec`, `application_data` and `alert`.

With a concern for interoperability, the specifications allow the two parties involved to negotiate the version of the protocol that they will jointly adopt. This parameter is established during a *TLS handshake* phase that precedes the effective protection of the exchanges. Likewise, the specifications allow for the use of different combinations of cryptographic algorithms. The cipher suite retained for the session is determined through messages of type `handshake`, in addition to messages of type `change_cipher_spec` that signals its effective activation.

Figure 1.1 shows the negotiation of these parameters in a generic case. It makes use of the following exchanges between the client and the server:

1. the client initiates a query by sending a message of type `ClientHello`, containing in particular the cipher suites that it supports;
2. the server responds with a `ServerHello` which contains the retained suite;
3. the server sends a `Certificate` message which in particular contains its public key within a digital certificate;
4. the server transmits in a `ServerKeyExchange` an ephemeral value that it signs using the private key associated with the preceding public key;
5. the server manifests that it is on hold with a `ServerHelloDone`;

---

[5]Internet Engineering Task Force.
[6]Internet Assigned Numbers Authority.
[7]Transmission Control Protocol.
[8]User Datagram Protocol.
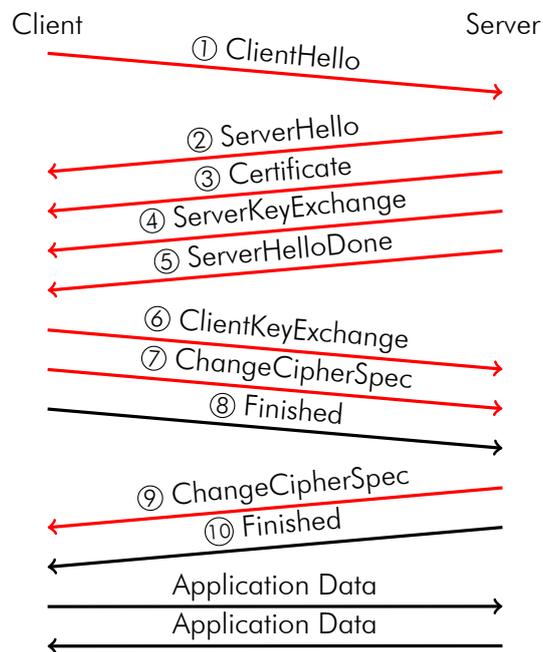[9]Datagram Transport Layer Security.

Figure 1.1: Generic initiation of a TLS session

6. after verification of the certificate and authentication of the preceding value, the client in turn chooses an ephemeral value that it encrypts using the public key of the certificate and then transmits in a `ClientKeyExchange`;

7. the server signals it activates the suite fully with a `ChangeCipherSpec`;

8. the client sends a `Finished`, the first message protected according to the cipher suite with the secrets coming from the exchange of ephemeral keys;

9. the server signals the activation of the same suite with a `ChangeCipherSpec`;

10. the server sends in turn a `Finished`, its first secure message.

The generic case described here supposes the selection of one of the cipher suites that provides PFS[10]. This property consists in the prevention of past session messages decryption, even when the private key of the server has been compromised. It is accomplished through the negotiation of an ephemeral secret using a Diffie–Hellman exchange [14].

The specifications of the protocol furthermore define several additional messages and extensions that make it possible to supervise and enrich the protection of communications [10, 15]. In contexts wherein such a need has been identified, the server is in particular able to request authentication of the client at the TLS level through a `CertificateRequest` message. With regards to the extensions, a `ClientHello` can for example contain additional information pertaining to the elliptic curves supported

---

[10]Perfect Forward Secrecy.

by the client in order to carry out the ECC[11] calculations.

After the handshake, the client and the server possess a shared *premaster secret*, whose calculation relied on elements contained in the `ServerKeyExchange` and the `ClientKeyExchange`. On both sides, the premaster secret is derived into one *master secret* that takes account of the random values contained in the `ClientHello` and the `ServerHello`. Finally, the master secret is in turn derived in order to generate the keys to ensure the confidentiality and the integrity of the application data, before encapsulating it in messages of type `application_data`.

The specifications allow to redefine the security parameters or to refresh the encryption keys without interrupting the TLS session, by using a new handshake. This session renegotiating can be triggered at the initiative of the client through a new `ClientHello`. It can also be solicited by the server with a `HelloRequest` message.

Messages of type `alert` enable to report anomalies observed during the handshake or application data exchange. Certain messages comprise simply warnings, while others call for immediate termination of the TLS session. Several alert codes exist, which for example make it possible to report that a transmitted certificate is not valid, or that the integrity check for a record has failed.

1.2 Public Key Infrastrustures

The validity of the certificates sent during TLS negotiation is crucial for the verification of the identity of the communicating parties. All of the mechanisms and entities that guarantee this validity and maintain it form a PKI[12].

For any certificate compliant with the X.509 [16] standard followed by TLS, the assurance that the public key it contains actually belongs to the server that it is announcing as a subject (generally in the form of a domain name) is based on trust inherited from an already recognised authority to the server in question. The successive trust links established by each certification authority involved are materialised by cryptographic signatures affixed to the various certificates.

As such, the `Certificate` message from which is extracted the public key whereon is based the session secrets actually contains a chain of certificates, which the client expects to form a link from a trusted root to the queried server. These roots are generally listed in registers known as *certificate stores*. Microsoft, Apple and Debian, for example, maintain such registers, which are made available to any application installed on the associated operating systems. As for Mozilla, it maintains its own store, through its NSS [17] library.

---

[11]Elliptic Curve Cryptography.
[12]Public Key Infrastructure.

Certificates contain several attributes, such as a public key and a validity period, which are usually supplemented with X.509v3 extensions. ANSSI recommends following appendix A4 [18] of the RGS[13] The extensions make it possible in particular to specify the framework of use for a certificate and to reinforce the assurances of the PKI. For example, the presence of an EV[14] extension signals that additional requirements were added to the identity verification process carried out by the CA[15] before it delivers the certificate.

In order to handle the errors or attacks that could compromise the operation of a PKI, but also to accompany the nominal procedures of security health such as key renewal, certificate revocation mechanisms have been defined. Two main solutions exist:

- CRL[16] files: these files correspond to lists of certificates revoked by a CA. Maintaining an up-to-date CRL online is part of the services that a PKI has to provide. The location of the CRL associated with a CA is entered in the CRLDP[17] extension of each certificate issued by the CA;
- the OCSP[18] protocol: this protocol, operating in client–server mode, allows a client to check the validity of a certificate by questioning OCSP responders. If a PKI makes an OCSP service available, the location of the associated responders must be available in the AIA[19] extension of each certificate issued by the CA.

Recent initiatives attempt to overcome the respective disadvantages of these two mechanisms, in particular the size of the CRLs which often exert a constraint on the network resources, and the synchronous nature of the OCSP queries that inform the responders of the profile of a portion of the connections of the client, as this can form a nuisance with regards to privacy protection.

For the Chrome browser, the use of CRLs and of OCSP was deactivated. These mechanisms were replaced with an ad hoc system. The Google team identifies the most significant revocations from the CRLs made available by the CAs, and distribute them to the users in the form of aggregations called CRLSets [19]. Mozilla launched a similar project for its Firefox browser, under the name OneCRL [20].

OCSP stapling is an alternative solution which consists for the server in directly supplying an OCSP response among the TLS extensions of `ServerHello`, in such a way that the client no longer has to query the responders itself [15].

---

[13]The *Référentiel général de sécurité* is a legal standard for the French public entities for operating trusted information systems. It includes global cryptographic requirements as well as certificate management instructions. The document is available in French only. Note however that many recommendations from the RGS relative to X.509 certificates are recalled in section 3.1 of this guide.

[14]Extended Validation.

[15]Certification Authority.

[16]Certificate Revocation List.

[17]Certificate Revocation List Distribution Point.

[18]Online Certificate Status Protocol.

[19]Authority Information Access.

In addition to the revocation mechanisms, the Certificate Transparency project, initiated by Google and standardised by IETF [21], aims to create public registers that list X.509 certificates. These registers allow a TLS client that is compatible with Certificate Transparency to check the validity of certain certificates which were sent to it. It also makes it possible to monitor the appearance of new certificates. Verification is carried out via a SCT[20] timestamped and signed by the administrators of one of the registers, which ensures that the corresponding certificate was journalised. The SCTs may be transmitted in a TLS extension, an OCSP response or within the certificate itself.

---

[20]Signed Certificate Timestamp.

Chapter 2

TLS Handshake Parameters

This chapter covers the various parameters on which the security of a TLS connection is based, without prejudging the nature of the application data to be protected or the encryption context. Website security and TLS proxy configuration are addressed by further technical documentation [1, 2].

Being widely used, the TLS protocol has been the subject of many studies which repeatedly uncovered new vulnerabilities [22, 23, 24, 4]. In light of the corrections and the improvements made over the years, both to the specifications and to the implementations, it is essential to use the latest versions of the equipment and software involved in securing communications.

> **R2** Use up-to-date software components
>
> The resources used for TLS deployment must be kept up to date.

2.1   Protocol Versions

Since the publication of SSLv2 in 1995, the identification of limitations of the protocol has justified several updates to its specifications. To date there are five variations of the protocol in use: SSLv2, SSLv3, TLS 1.0, TLS 1.1 and TLS 1.2. The specifications of TLS 1.3 are being developed by the IETF [25].

The version used during a session is negotiated during the handshake. The client reports in its `ClientHello` the highest version that it supports. Subsequently, the server responds in its `ServerHello` with the highest version that it supports and which is at most equal to that preferred by the client.

To date, under certain conditions specified in the rest of this section, TLS 1.0 is still considered safe for use. However, TLS 1.1 resolves a portion of the attacks established against TLS 1.0, and in turn TLS 1.2 was defined to be more robust than TLS 1.1.

If it appears necessary to support versions other than TLS 1.2, it is recommended to simultaneously support the `TLS_FALLBACK_SCSV` suite defined hereinbelow. In particular, although the use of TLS 1.1 or TLS 1.0 is not encouraged in the general case, these versions can be tolerated if it is necessary to communicate with components (non-compliant with the recommendations) that do not support TLS 1.2.

> **R3** Use only TLS 1.2
>
> The connection must be made with version TLS 1.2. Any attempt to connect with a prior version must be rejected.

> **R3 -** Favour TLS 1.2 and tolerate TLS 1.1 and TLS 1.0
>
> Version TLS 1.2 must be supported and favoured. Versions TLS 1.1 and TLS 1.0, with support for `TLS_FALLBACK_SCSV`, are tolerated.

The publication of SSLv3 responded to several design flaws identified for SSLv2. However, this version of the protocol is today no longer considered to be safe. In June 2015, the IETF listed several weaknesses of SSLv3 and formally declared it as obsolete [11].

In April 2015, the dummy cipher suite `TLS_FALLBACK_SCSV` was defined in order to limit the risks of attack by version downgrading, in particular linked to implementations that attempt to establish an SSLv3 connection after failed TLS connections [26]. Following a first connection failure, a client that implements `TLS_FALLBACK_SCSV` and wants to try a `ClientHello` of a downgraded version should add the preceding SCSV[21] to it. In this way, if a server observes the SCSV in a `ClientHello` which announces a version that is less than the most recent version that it supports itself, it knows that a first exchange has failed. Depending on the context, it may infer an attack attempt from this, and alert the client.

> **R3 - -** Favour TLS 1.2 and tolerate TLS 1.1, TLS 1.0 and SSLv3
>
> Version SSLv3 is not at all recommended in the general case. In addition to **R3-**, in the case where a strong need for compatibility has been identified, and only in this case, the version SSLv3 along with `TLS_FALLBACK_SCSV` is tolerated.

The lack of robustness of SSLv2 has been established for a long time. The weakly protected handshake or the use of weak cryptographic primitives expose the exchanges to several scenarios of compromise. Moreover, the DROWN [5] vulnerability revealed that the simple supporting of SSLv2 by a server was able to compromise sessions initiated with later versions of the protocol. This version of the protocol should therefore be banned.

---

[21]Signaling Cipher Suite Value.

2.2 Cipher Suites

The `ClientHello` contains a set of cipher suites that the client is ready to use during the session. It is expected that the server selects one of them, after comparison with those that it supports and accepts to use. This selection affects the way in which the cryptographic keys will be used to protect the records exchanged after the handshake, which in particular transport application data. The procedure for negotiating keys is itself modified according to the suite retained.

Each suite comprises a combination of the following encryption mechanisms:

- a key exchange method, which specifies an exchange algorithm and possibly the signature algorithm used to authenticate the exchanges. `RSA`, `ECDHE_RSA` and `PSK` are a few examples of these;
- a mechanism that ensures the confidentiality and the integrity of the data exchanged after the handshake, defined:
    - either as the composition of an encryption algorithm and of a hash function used in HMAC mode, such as `AES_256_CBC_SHA384`;
    - or as an encryption algorithm used in AEAD[22] mode, thus providing confidentiality and integrity at the same time, such as `AES_256_GCM`;
- optionally, for suites defined for version TLS 1.2, a hash function used for the derivation of the secrets, starting from the premaster secret. This option is in particular used by `AES_GCM` suites.

For example, the suite `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384` referenced by IANA under code `0xC030` represents the association of the `ECDHE_RSA` key exchange with the `AES_256_GCM` AEAD mode, along with `SHA384` for the derivation of secrets.

Among the algorithms standardised before June 2016 for use in the framework of TLS, the recommendations in this section form a whitelist of the desirable algorithms: everything that is not recommended is implicitly disapproved. In particular, the use of either the RC4 traffic encryption function or the MD5 hash function is disapproved.

To date, we issue no recommendation about the AEAD cipher `CHACHA20_POLY1305`, for which the use with TLS was standardised in June 2016 [27].

---

[22]Authenticated Encryption with Associated Data.

Key Exchange

There are several methods for exchanging keys, some of which do not require authentication from the server. However, in the absence of this protection, the key exchange is vulnerable to man-in-the-middle attacks that can compromise the security of the entire data exchange. Consequently, authenticating the server is indispensable.

> **R5** Authenticate the server during key exchange
>
> During the key exchange, the server has to be authenticated by the customer. Anonymised alternatives for these exchanges are strongly disapproved.

In the general case, the key exchange is based either on the asymmetrical encryption of a secret using the public key of the server, or on the Diffie–Hellman algorithm. They both allow for the establishment of a joint secret on either side of the unsecured channel.

The DH[23] algorithm, sometimes designated as FFDH[24], represents the historical Diffie–Hellman exchange which rely on a multiplicative group. The mathematics of the ECDH[25] algorithm are based on an elliptic curve.

Verifying perfect forward secrecy means that, should the server's private key be compromised, past communications would still be protected from third party decryption. This property is ensured by the ephemeral variants of the preceding algorithms, DHE[26] and ECDHE[27], for which the Diffie–Hellman keys are generated at each new session. In this context, the authentication provided through the key exchange takes the form of a signing of the `ServerKeyExchange` with the private key of the server.

> **R6** Always do PFS-enabling key exchanges
>
> The perfect forward secrecy property must be ensured. For this, a cipher suite based on an ephemeral Diffie–Hellman exchange (ECDHE or, failing this, DHE) must be used.

---

[23]Diffie–Hellman.
[24]Finite Field Diffie–Hellman.
[25]Elliptic Curve Diffie–Hellman.
[26]Diffie–Hellman Ephemeral.
[27]Elliptic Curve Diffie–Hellman Ephemeral.

> **R6-** Favour PFS-enabling key exchanges
>
> Ephemeral Diffie–Hellman exchanges have to be supported and favoured. Exchanges based on the encryption of a secret using the key of the server, which do not provide perfect forward secrecy, are tolerated.

Concerning ECDHE, for data protection after 2020, the RGS recommends the use of groups with a prime order of length 256-bit or more [28]. Among the curves recorded with IANA [29], the tried out, retained curves are `secp256r1`, `secp384r1`, `secp521r1` (a.k.a. P-256, P-384 and P-521), as well as `brainpoolP256r1`, `brainpoolP384r1` and `brainpoolP512r1`.

Note that the negotiation of the DHE group is not, to date, allowed by the protocol. When this solution is favoured, the group is in fact imposed by the server in a `ServerKeyExchange`. Thus, a client that wants to ensure PFS with DHE, but which is presented with a group deemed to be unsatisfactory, would have no other solution than to interrupt the session.

In contrast, the negotiation of the ECDHE parameters is made possible by the use of the `supported_groups` extension. The latter makes it possible to select the implicit parameter curves, recorded with IANA. The use of ECDHE is therefore preferred to that of DHE when one of the communicating parties is not controlled.

> **R7** Carry out an ECDHE key exchange
>
> ECDHE key exchange must be favoured, using curves `secp256r1`, `secp384r1`, `secp521r1`, `brainpoolP256r1`, `brainpoolP384r1` or `brainpoolP512r1`.

In the case of DHE, the security of the exchange is linked to the order of the multiplicative group at play. The Logjam [4] attack illustrated the insufficiency of 512-bit groups, and prompts use of 1024-bit groups for the most sensitive exchanges to be discouraged. The RGS recommends using 3072-bit groups or more, and tolerates 2048-bit groups for data protection until 2030.

> **R7-** Carry out a DHE key exchange
>
> DHE exchanges are tolerated with 2048-bit groups or more (3072-bits or more if the information has to be protected after 2030).

Other methods for exchanging keys such as PSK[28] [30] and SRP[29] [31] are based on the prior sharing of a secret between the client and the server. More complex to deploy and maintain, they form valid alternatives only in controlled environments, in particular for infrastructure applications.

Symmetric-Key Algorithm

The premaster secret negotiated through the preceding exchange is derived into a master secret. From this master secret is then derived the symmetrical encryption key used for protecting the confidentiality of communications that follow the handshake. The encryption algorithm at play is however set from the moment a cipher suite is elected.

Stream Ciphers

Currently, RC4 is the only stream cipher proposed among the standardised suites. However, following work [32] indicating a statistical bias that can lead to compromising repeated data in a large number of TLS sessions, such as a password or an HTTP cookie, IETF prohibited the use of this function [33].

Block Ciphers

In opposition, several block cipher functions can be used to secure TLS connections. Published in 1977, DES[30] is no longer considered safe today [34]. Its successor AES[31] should be favoured, with a key size of 128 bits or more, in accordance with the RGS[32]. Although they are subjected to less examination, Camellia and ARIA today offer security comparable to AES and can be considered as alternatives.

The Triple DES algorithm forms a last resort. However, like the underlying DES algorithm, it suffers from a reduced block size. Because of this, in order to prevent them from being compromised, it is necessary to renew the Triple DES encryption keys at least every gigabyte of application data exchanged.

Concerning the choice between a 128-bit or 256-bit key for AES, there is today no practical attack that questions the trust placed in AES-128. However, as AES-256 is deemed to be more robust, its use is preferred to that of AES-128 in this document.

---

[28]Pre-Shared Key.
[29]Secure Remote Password.
[30]Data Encryption Standard.
[31]Advanced Encryption Standard.
[32]Référentiel Général de Sécurité.

> **R8** Use AES encryption
>
> Suites that implement the AES-256 block cipher are to be favoured. The AES-128 algorithm is an acceptable alternative.

> **R8-** Use Camellia or ARIA encryption
>
> Suites that implement the Camellia and ARIA block ciphers are tolerated. Supporting the AES algorithm is recommended, but not obligatory.

> **R8--** Favour AES and tolerate 3DES with refreshment
>
> AES encryption must be supported and favoured. Suites that implement the Triple DES block cipher algorithm are tolerated, as long as the associated encryption keys are renewed at least at every gigabyte of application data exchanged.

Encryption modes that implement the selected block cipher algorithm are described after the introduction of the integrity mechanism based on a hash function.

Authentication Code

Once the TLS session is established, each record sent is protected in full. When the cipher suite does not use an AEAD mode, it is an integrity code calculated by the HMAC mode, which is based on a hash function. The standardised suites for TLS allow for the use of MD5, SHA-1 or an element of the SHA-2 family.

The MD5 function has been subjected to many attacks, which motivated its exclusion by the RGS. Since 2005, several studies have also altered the robustness of SHA-1 [35, 36]. Although HMAC-SHA1 is not directly questioned today, in accordance with the RGS, its use is not recommended. Use of either SHA-256 or SHA-384, both of them functions from the SHA-2 family, must be preferred. Standardised in 2015, the use of the SHA-3 [37] function for the TLS protocol has not been specified to date.

> **R9** Build the HMAC with SHA-2
>
> The HMAC used for authentication must be built using a representative of the SHA-2 family: SHA-256 or SHA-384.

Encryption Mode

For the cipher suites defined for use with TLS versions prior to version 1.2, block ciphers are used in CBC mode for encryption. The result is then combined with a HMAC which ensures the integrity.

This combination is carried out according to the following scheme: the computation of the HMAC covers a sequence number, a header, and the plain data. The plain data and the integrity pattern coming from the HMAC are then encrypted according to the CBC mode. This sequencing of operations introduces possible leaks of information during the decrypting operation, that can potentially lead to attacks that challenge the confidentiality of a piece of data transmitted through a large number of sessions [38]. Such leaks of information can be entirely avoided only at the price of a major implementation effort [39].

If the `encrypt_then_mac` extension is used, the order of the operations is reversed: the HMAC covers already encrypted data. This makes it possible to avoid information leaks during decryption, and to prevent certain vulnerabilities [40, 38].

Version 1.2 of the TLS protocol introduces the possibility of using AEAD cipher modes, offering simultaneously an encryption function and an integrity code function. Suites offering GCM and CCM modes of operation have as such been standardised.

Summary

The encryption suites that meet the preceding requirements and which are recommended for general use with the TLS protocol are listed in tables 2.1 and 2.2. Recall that the extension `encrypt_then_mac` is recommended for the suites that use the CBC encryption mode.

Value	Cipher suite
0xC02C	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
0xC02B	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
0xC0AD	TLS_ECDHE_ECDSA_WITH_AES_256_CCM
0xC0AC	TLS_ECDHE_ECDSA_WITH_AES_128_CCM
0xC024	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
0xC023	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256

Table 2.1: TLS 1.2 suites recommended for servers with an ECDSA key

Value	Cipher suite
0xC030	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
0xC02F	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
0xC028	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
0xC027	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256

Table 2.2: TLS 1.2 suites recommended for servers with an RSA key

When one of the two communicating sides is not controlled, it is not always possible to negotiate a TLS session with the preceding cipher suites. Appendix A indicates suites of lesser security that can be considered to respond to strong compatibility needs. Furthermore, the following section provides additional recommendations pertaining to the deployment context.

Deployment Contexts

In the situation where the server and client profiles are controlled, if the server has a certificate for a ECDSA key, then it is sufficient for the server and the client to use only one of the cipher suites from table 2.1, like `TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384` with elliptic curve `secp256r1`.

Should the latter become compromised, it is still preferable that each one of the communicating parties has an implementation of the other suites from table 2.1 that can be used later. This principle also applies to the six elliptic curves recommended hereinabove.

Likewise, if the server has a certificate for an RSA key, it is recommended that the server and the client have several cipher suites from table 2.2, and it is sufficient to implement only one of them, for example `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`.

> **R11** Maintain several cipher suites available
>
> For preventive purposes, the communicating sides must implement several acceptable suites. When the infrastructure is controlled from end to end, they can then use only one of them.

In the absence of control on the clients, the server has to grant more trust to its own processing than theirs. In particular, the server has to favour the preference between cipher suites established in its own configuration, rather than the cipher suites order inside the `ClientHello`.

> **R12** Prefer the suites order of the server
>
> When the clients of a server are not controlled, the order of the cipher suites appearing in its configuration must prevail over the orders of the suites advertised by the clients.

2.3   Extensions

The `ClientHello` sent by the client at the beginning of the session may contain several extensions. They generally enable the client to inform the server of its various capacities, such as its support for ECDSA signatures. They also make it possible to provide the server with additional information, such as the domain name that the client wants to reach. The server confirms its own support and its desire to use some of the capacities of the client by inserting a subset of extensions of the `ClientHello` into its `ServerHello`. Each extension is identified by a two-byte integer, registered by IANA [41].

Depending on the context, these extensions can be purely informative, or indeed indispensable for the unfolding of the TLS session. Using them is often conditioned to a particular application. Because of this, the extensions for which the use is acceptable for certain contexts do not have the value to be used for all of the sessions. Extensions that are not recommended, however, are so in all circumstances. A client must not send them in a `ClientHello`, and if a server or a client receives one of them, it should ignore it. Choosing not to support these extensions may also reduce the attack surface of the applications involved in the exchange.

Generically Recommended Extensions

- supported_groups (0x000A) [42]

This extension informs the server of the elliptic curves supported by the client (if any), in accordance with the register maintained by IANA. Its support and use are mandatory when the client or the server want to use ECC functions. The order in which the curves are presented reflects the preferences of the client.

- signature_algorithms (0x000D) [10]

This extension signals the hash and signature algorithms supported for checking the authenticity of future messages of the handshake, in particular the ServerKeyExchange. In the recommended framework of a TLS 1.2 session with authenticated key exchange, support for and use of it are required by the client and the server, and the support of at least one representative of the SHA-2 family should be announced. For prior versions of the protocol, the authentication mechanism is different and this extension must not be used.

- signed_certificate_timestamp, ou sct (0x0012) [21]

This extension signals SCT support. In the CT[33] model presented in section 1.2, an SCT can be required in order to establish the validity of the certificate presented by the server. Transmitting an SCT in the data field of this TLS extension is an alternative to the presence of an SCT among the extensions of the certificate, or among the extensions of an associated OCSP status.

- encrypt_then_mac (0x0016) [43]

This extension signals support for the cryptographic construction *encrypt-then-mac*, as an addition to the historical construction *mac-then-encrypt*. The use of this extension is highly recommended when the CBC encryption mode is used.

- extended_master_secret (0x0017) [44]

This extension signals the capacity to process an *extended master secret*. The process for calculating the extended master secret is more robust than that of the legacy master secret, because it is no longer based solely on the random values contained in ClientHello and ServerHello, but also on all of the negotiated cryptographic context (suites, method for exchanging keys, certificates), through a hash sum of all of the messages exchanged during the handshake.

- renegotiation_info (0xFF01) [45]

The renegotiation mechanism and the need for this extension are discussed in section 2.4.

---

[33]Certificate Transparency.

Acceptable, Context-Specific Extensions

- **server_name** (0x0000) [15]

This extension signals SNI[34] support. It allows the client to specify the domain name of the server that it wants to reach. Its use is widespread and makes it possible in particular to host several TLS servers behind the same IP address.

- **max_fragment_length** (0x0001) [15]

This extension signals reduced fragmentation support. It makes it possible to bring to 2^{12}, 2^{11}, 2^{10} or 2^9 the maximum standard length of 2^{14} bytes defined for fragments of the original message treated for the construction of records.

- **trusted_ca_keys** (0x0003) [15]

This extension signals certificate identifiers support. It may guide the server in the selection of the certificate chains it sends in its `Certificate` message. For this, the client supplies in the data field of the extension the names of the CAs that it trusts, or the hash values of the corresponding certificates.

- **status_request** (0x0005) [15]

This extension signals OCSP stapling support. The data field of the extension sent by the client lists the trusted OCSP responders. If the server supports this mechanism, the extension that it sends in turn does not contain any data, but its `Certificate` message is immediately followed by a `CertificateStatus` message containing an OCSP response.

- **user_mapping** (0x0006) [46]

This extension signals support for sending additional data in a new `SupplementalData` message from the client, as a helper to the server for identification. It is in particular useful in the situation where the server has an index of the clients with which communication is authorised.

- **cert_type** (0x0009) [47]

This extension makes it possible to signal OpenPGP [48] certificates support. These differ from the legacy X.509 certificates, and make it possible to establish a cryptographic link between an e-mail address, an identity and a public key. Using and supporting this extension are generally not required.

- **ec_point_formats** (0x000B) [42]

This extension signals which elliptic curve point formats are supported by the client or the server (if any). It is in fact possible to represent elliptic curve points in compressed form. In the absence of this extension, it is expected that the point coordinates are transmitted in their entirety.

---

[34]Server Name Indication.

- srp (0x000C) [31]

This extension signals SRP protocol support by the client. It furthermore contains a user-name that allows the server to decide the parameters sent in the `ServerKeyExchange`. The authentication of the server and of the client is based on the knowledge of a secret that requires the same level of protection as the private keys in the framework of traditional authentication. The use of this extension is associated with that of the SRP cipher suites. In this situation, in order to reinforce its authentication, it is strongly recommended that the server have a certificate associated with an ECDSA or RSA key that makes it possible to sign the `ServerKeyExchange`.

- use_srtp (0x000E) [49]

This extension signals DTLS-SRTP support. SRTP is a secure variant of RTP[35], a protocol optimised for transferring data in real time, such as video traffic [50]. The specifications of RTP do not cover the negotiating of session parameters and keys, which can be handled by DTLS, a variant of TLS intended for communication via datagrams. Using and supporting this extension are generally not required.

- application_layer_protocol_negotiation, ou alpn (0x0010) [51]

This extension signals support for negotiating application protocols. It makes it possible to mark a preference among various application protocols sharing the same TLS-protected port. The extension primarily aims to allow clients to announce their support for HTTP/2, the most recent version of the HTTP protocol. Using and supporting it are recommended when the use of HTTP/2 is desired.

- status_request_v2 (0x0011) [52]

This extension supplements the functionality of the `status_request` extension with regards to the OCSP stapling mechanism, presented in section 1.2. It allows the server to send the client a list of `CertificateStatus` messages, which not only contain the status of its own certificate, but also that of intermediate certificates from the certificate chain previously offered in the `Certificate` message.

- padding (0x0015) [53]

This extension allows the client to add padding to the `ClientHello` in the form of null bytes. It was exceptionally defined as a response to the detection of a bug linked to the length of `ClientHello` messages. Using and supporting this extension are generally not required.

---

[35]Real-time Transport Protocol.

- session_ticket (0x0023) [54]

This extension signals session resumption capabilities through session tickets. When it is received in a ClientHello, if the server supports this mechanism, it confirms the use of it by announcing in turn the extension in its ServerHello. Either the ClientHello extension does not contain any additional data, in which case the server delivers a session ticket which contains the parameters of the current negotiation in a crypto-graphically protected form, or the ClientHello extension contains a session ticket obtained beforehand, of which the cryptographic integrity is checked by the server, and the associated negotiating parameters are consequently restored.

Unadvised Extensions

- client_certificate_url (0x0002) [15]

This extension signals support for client authentication via remote certificates, located by one or several URLs. The latter are transmitted in a CertificateURL message, which replaces the Certificate message that is usually issued by the client. This mechanism can be hijacked to force a server to perform a significant number of queries (HTTP, FTP, and more particularly HTTPS) with various hosts. The extension is generally not necessary and its support outside of controlled environments is therefore not recommended.

- truncated_hmac (0x0004) [15]

This extension signals truncated HMACs support, which consists in suffixing each record with only the first ten bytes of the original HMAC. This construction weakens the authentication mechanism and supporting it is not recommended.

- client_authz (0x0007) [55]

This extension allows the client to signal its ability to send additional authentication data in SupplementalData messages. This enables the server to know which applications the client should be interfaced with. This authentication information does not pertain to the protocol, therefore supporting this extension is not recommended.

- server_authz (0x0008) [55]

This extension allows the server to signal its ability to send additional authentication data in SupplementalData messages. This enables the client to learn of the reputation of the server. This authentication information does not pertain to the protocol, therefore supporting this extension is not recommended.

- `heartbeat` (0x000F) [56]

This extension signals *heartbeat* support. Heartbeats aim to keep a TLS connection alive, by forcing the immediate sending of a `HeartbeatResponse` message for every `HeartbeatRequest` received. This extension is generally not necessary and a defective implementation of it in OpenSSL (versions 1.1.1 to 1.0.1f inclusive) was cause of the Heartbleed [24] vulnerability. It is strongly recommended not to support it.

- `client_certificate_type` (0x0013) [57]

When completing mutual authentication, this extension enables the client to signal that it may send a raw public key in its `Certificate` message, rather than a list of X.509 certificates. The proof of possession of the public key is established independently of the TLS connection. This extension is generally not necessary as the use of a PKI should be preferred. It is recommended not to support it.

- `server_certificate_type` (0x0014) [57]

This extension enables the server to signal that it may send a raw public key in its `Certificate` message, rather than a list of X.509 certificates. For the same reasons as the `client_certificate_type` extension, supporting this extension is not recommended.

Summary

> **R13** Use the extensions from table 2.3
>
> The extensions recommended in the general TLS framework are mentioned in table 2.3. They must be supported by the controlled equipments and used in the previously specified contexts.

Value	Name	Context of use
0x000A	supported_groups	In case of ECC
0x000D	signature_algorithms	In case of TLS 1.2
0x0012	sct	In case of EV certificate
0x0016	encrypt_then_mac	Always recommended
0x0017	extended_master_secret	Always recommended
0xFF01	renegotiation_info	In case of renegotiation

Table 2.3: TLS extensions recommended in the general framework

R14 Assess the relevance of the extensions from table 2.4

Extensions from table 2.4 are context-specific. Only those evaluated as necessary must be implemented and used.

The most commonly used extensions are `server_name`, `status_request_v2`, `status_request` (to supplement `status_request_v2` where compatibility with uncontrolled clients is necessary) and `session_ticket`.

Value	Name	Context of use
0x0000	server_name	In case of shared hosting
0x0001	max_fragment_length	In case of network constraints (rare)
0x0003	trusted_ca_keys	In case of several trust anchors (rare)
0x0005	status_request	In case of OCSP stapling
0x0006	user_mapping	In case of indexed clients (rare)
0x0009	cert_type	In case of OpenPGP certificates (rare)
0x000B	ec_point_formats	In case of compressed ECC points
0x000C	srp	In case of SRP suites
0x000E	use_srtp	In case of SRTP (rare)
0x0010	alpn	In case of HTTP/2
0x0011	status_request_v2	In case of OCSP stapling
0x0015	padding	In case of a defective server (rare)
0x0023	session_ticket	In case of session resumption

Table 2.4: Context-specific TLS extensions

R15 Do not use the extensions from table 2.5

The extensions in table 2.5 are never recommended.

Value	Name
0x0002	client_certificate_url
0x0004	truncated_hmac
0x0007	client_authz
0x0008	server_authz
0x000F	heartbeat
0x0013	client_certificate_type
0x0014	server_certificate_type

Table 2.5: Unadvised TLS extensions

2.4 Additional Considerations

Random Items

During the TLS handshake, both the client and the server generate a random item and send it, either via the `ClientHello` or the `ServerHello`. These two values intervene several times, in particular during the signing of Diffie–Hellman parameters, during the calculation of the master secret, and during the hashing of all of the handshake messages for the `Finished` messages. In contexts where the generators are controlled, the use of unpredictable values constitutes essential protection against replay attacks.

The computation of these 32-byte values must rely on a quality PRNG[36], such as defined by the RGS [28]. The specifications of TLS 1.2 separate the 32 bytes into 28 random bytes and a prefix of 4 bytes corresponding to the Unix time at the writing of the message. This construction aimed to generate random items that were not immediately predictable, should the PRNG be compromised. However, this scheme induces risks for unwanted tracking, and the use of a robust PRNG remains indispensable. Consequently, when possible, the fully randomised 32-byte structure is to be preferred.

> **R16** Use a strong PRNG
>
> The random values used in the `ClientHello` and the `ServerHello` must originate from reliable PRNGs.

> **R17** Favour random items without a predictable prefix
>
> The random values used in the `ClientHello` and the `ServerHello` must favour 32 random bytes rather than the structure with a Unix time prefix.

Compression

The handshake, in addition to the parameters for protecting records, also determines a compression algorithm used on the application date before encryption. In practice, the function usually retained is the one which actually leaves this data unchanged. It is however still possible to use the Deflate algorithm [58].

However, the use of compression combined with the absence of protection on the length of the encrypted messages can result in the compromising of a portion of the data exchanged, and in particular session cookies that certain servers use to identify their clients [59]. The use of these cookies by a third party is the same as identity theft, which is dangerous for the initial client unless the server only proposed static content that is identical for all of its clients.

---

[36]Pseudorandom Number Generator.

This vulnerability is countered by most web browsers, which have either deactivated the compression mechanisms, or have implemented a workaround such as not compressing the header containing the sensitive cookie. However, this approach covers neither the entirety of the applicative use cases, nor the entirety of the TLS-enabled clients. Consequently, except when consulting a server with static content, compression remains unadvised.

> **R18** Do not use TLS compression
>
> Except when consulting a server with static content, using the TLS compression mechanism is not recommended.

Session Resumption

There are two methods for resuming a session, making it possible to shorten the TLS handshake by restoring secrets that were previously established.

The first method is based on the session identifier contained in the `ClientHello`, associated by the client with a set of parameters and secrets that were previously cached. In this situation, if the server itself has cached the parameters of the session characterised by the received identifier, then it can choose to restore them and the phase of exchanging keys does not need to be reproduced. If the server does not recognise the session identifier, the negotiation continues in the standard way.

The second method is based on session tickets and the corresponding extension, described briefly in section 2.3. The use of session tickets is preferred to that of session identifiers, as it does not require the server to retain the parameters pertaining to a session. Moreover, protecting them via cryptographic means is required by the specifications of the extension [54].

However, regardless of the means of protection used, the retaining of parameters induces a risk with respect to the perfect forward secrecy. In addition, although other means of prevention exist, note that the Triple Handshake [60] attack is immediately thwarted by the absence of support for a mechanism to resume the session.

Although the implementations rarely allow for this, if these mechanisms are used, the period of time for retaining the cached information for the identifiers, as well as that for tickets, must be reduced according to the level of security desired. A daily purge of the caches is an acceptable compromise.

> **R19** Do not perform session resumption
>
> Session resumption is not recommended.

R19− Perform session resumption with PFS

Should a session resumption mechanism be used, then session tickets must be preferred to session identifiers. In the case of session tickets, the tickets have to be deleted at short intervals, and the encryption keys have to be deleted and regenerated on a regular basis. In the case of session identifiers, the cached data has to be deleted on both sides at regular intervals.

R19−− Perform session resumption without PFS

The use of session tickets or session identifiers without the regular deleting of the associated data is tolerated.

Renegotiation

The renegotiation mechanism initially designed for TLS exposes the client to a protocol vulnerability. It is indeed possible for an attacker to pass off the first negotiation of a client as a renegotiation. The attacker is thus in a position to inject application data that the server attributes to the legitimate client. From a server standpoint, this data transparently precedes the application data subsequently sent by the legitimate client. This process can be used as a primer for more complex attack scenarios [22, 61].

The `renegotiation_info` extension was defined in order to carry out secure renegotiations. The use of this extension requires retaining the protected content of the `Finished` messages used to authenticate the last handshake.

If an attacker performs a first negotiation with a server that supports secure renegotiations, then passes off the `ClientHello` of a legitimate client as a renegotiation, then the presence of the `renegotiation_info` extension in this `ClientHello` without it being accompanied by the content of `Finished` (which, from the legitimate client standpoint, never took place) allows the server to detect the hack and to terminate the session. Note that the use of the extension is still necessary for the client even if it never wants to perform a renegotiation.

R20 Operate secure renegotiations

A TLS client, whether or not it wants to perform renegotiations, must use the `renegotiation_info` extension. A TLS server, if it wants to perform renegotiations, must use the `renegotiation_info` extension.

Chapter 3

Setting Up a PKI

This chapter groups together the recommendations pertaining to the attributes of X.509 certificates used during the handshake. It also addresses the various revocation methods that exist. The methods for generating and storing keys, and obtaining or switching between certificates, goes beyond the scope of this document. They are covered in Appendix B2 of the RGS [62].

3.1 X.509 Certificate Attributes

An X.509 certificate is comprised of several basic fields, generally accompanied with extensions. The latter specify the contexts for use of the certificate and supply means for checking the certificate more precisely within the certification chain. This chapter describes a few of the important fields and extensions in the case of TLS authentication certificates, and issues recommendations as to their content. Recommendations on the other fields can be found in Appendix A4 of the RGS [18]. Additional information on their use and their structure is provided in RFC 5280 [16].

3.1.1 Basic Fields

The serial number of a certificate generally identifies the certificate within the certification authority that issued it. This serial number is a positive number whose size does not exceed 20 bytes. It must be unique among all of the certificates generated by the same CA. The RGS moreover recommends that CAs generate certificates with unpredictable serial numbers, in order to prevent collision attacks which could affect the signature.

The signature field indicates the algorithms used by the certification authority to sign the information of the certificate. It comprises the hash function used to calculate the data imprint, as well as the signature algorithm that is then applied. It is recommended to follow Appendix B1 of the RGS [28].

> **R21** Bear a certificate signed with SHA-2
>
> The hash function used for the signature of the certificate has to be one from the SHA-2 family.

A certificate indicates its own period of validity through two fields:

- the *notBefore* field indicates the date from which the certificate is valid;
- the *notAfter* field indicates the date after which it is no longer valid.

> **R22** Bear a certificate valid for a 3-year period at most
>
> The validity period of a TLS authentication certificate (either for a server or a client) must not exceed 3 years.

It is sometimes possible to renew a certificate with a certification authority while still retaining the same pair of asymmetric keys. However, in the same way as a certificate, a pair of keys must not be valid for more than 3 years.

The information on the public key of the subject of the certificate (the *Subject Public Key Info*) are structured into two parts: the identifier of the algorithm that will be implemented with this key, as well as the public key itself. Appendix B1 of the RGS provides several recommendations on the subject of algorithms and the key sizes to comply with.

> **R23** Use keys of sufficient size
>
> For use up to 2030, RSA keys should be at least 2048-bit long, and ECDSA keys should be at least 256-bit long. For ECDSA, the tried out, retained curves are `secp256r1`, `secp384r1`, `secp521r1`, as well as `brainpoolP256r1`, `brainpoolP384r1` and `brainpoolP512r1`.

3.1.2 Extensions

Since version 3 of the X.509 standard, extensions can be added to the certificates. The latter are primarily used to:

- identify the owner of the pair of asymmetric keys to which the certificate refers (*Subject Alternative Name*);
- specify the possible uses of the certificate (*Key Usage, Extended Key Usage, Basic Constraints, Subject Alternative Name*);
- supply information used to check the state of revocation of the certificate (*CRL Distribution Points, Freshest CRL, Authority Information Access*);
- supply information to check the certificate within the certification chain (*Authority Key Identifier, Subject Key Identifier*);
- supply a link to the certification policy that applies to the certificate (*Certificate Policies*).

There is an attribute for marking extensions considered critical. If an application does not recognise one critical extension of a certificate, it should reject the certificate.

The *Key Usage* extension defines the operations allowed to be carried out with the key associated with the certificate. This extension contains one or several values:

- the *digitalSignature* value indicates that the public key can be used to check electronic signatures which cover any object distinct from certificates and CRLs;
- the *keyEncipherment* value indicates that the public key can be used for session keys encryption. Thus the exchange would infringe PFS;
- the *keyAgreement* value signals a key useful for a key exchange protocol. The natural case corresponds to a static Diffie–Hellman key, but these certificates are rare; as explained in section 2.2, preference is to be given to Diffie–Hellman exchanges based on ephemeral keys, i.e. generated at each new session;
- the *keyCertSign* value indicates a certificate that belongs to a root or intermediate CA, of which the key was used to sign other certificates;
- the *crlSign* value also indicates a certificate that belongs to a root or intermediate CA, and of which the key was used to sign revocation lists.

> **R24** Bear an appropriate KeyUsage
>
> In a certificate used for authentication, the *Key Usage* extension must be present and marked as critical. For a server, it must contain the *digitalSignature* and/or *keyEncipherment* values. For a client, it must contain the *digitalSignature* value. No other value is allowed.

The *Extended Key Usage* extension indicates a more precise use of the certificate and adds to the *Key Usage* extension. Outside of *id-kp-OCSPSigning* used to characterise the CA certificates of which the key is used to sign OCSP responses, the other values defined for *Extended Key Usage* do not come into play in the framework of TLS.

> **R25** Bear an appropriate *ExtendedKeyUsage*
>
> In a certificate used for authentication, the *Extended Key Usage* extension must be present and marked as non-critical. For a server, it must contain only the *id-kp-serverAuth* value. For a client, it must contain only the *id-kp-clientAuth* value.

The SAN[37] extension makes it possible to indicate identities of the subject of the certificate other than the one present in the *Subject* field, such as full DNS names (*dNSName*) or email addresses (*rfc822Name*). As such, as an example, a web server certificate must

---

[37]Subject Alternative Name.

contain the domain name consulted by the user in the *Common Name* portion of the *subject* field, or among the FQDN[38] of the *dNSName* entries of the SAN extension.

> **R26** Bear an appropriate (server-side) *SubjectAlternativeName*
>
> In a certificate used by a TLS server for authentication, the *Subject Alternative Name* extension must be present and marked as non-critical. It must contain at least one *dNSName* entry that corresponds to one of the FQDNs of the application service that uses the certificate.

The SAN extension is sometimes used to associate several services with the same certificate. However, it is not recommended to use the same private key, and therefore the same certificate, for separate TLS services. Indeed, this practice will generate a sharing of the individual risks looming over each TLS termination point. In particular, the duplication, the distribution and the ubiquitous existence of the private key, in order for it to serve several TLS services, increases the risk of it being compromised. On the other hand, the use of a TLS proxy positioned in front of various application services remains acceptable, as long as the TLS termination point remains unique.

> **R27** Hold each certificate for one TLS termination point only
>
> The same certificate for authentication must not be used by more than one TLS termination point.

This reasoning extends to differentiating certificates according to the parameters retained for the negotiation. Indeed, certain attacks [5] have shown that the tolerance for a certain version of the protocol could compromise sessions carried out with other versions, if the latter were based on the same certificate.

> **Further steps**
>
> For the same termination point, it is recommended to use as many certificates as cleared versions and key exchange methods.

The AKI[39] extension makes it possible to identify the key of the CA that signed the certificate, in particular when the CA in question has several keys. Standard X.509 makes it possible to use the identifier present in the SKI[40] extension of the CA certificate

---

[38]Fully Qualified Domain Name.
[39]Authority Key Identifier.
[40]Subject Key Identifier.

with respect to the signature key, or to the association of the name of the CA and the serial number of the certificate with respect to the signature key. Only the first of the two options is RGS-compliant.

> **R28** Bear an AKI corresponding to the SKI defined by the CA
>
> In a certificate used for authentication (either by a server or a client), the AKI extension has to be present, marked as non-critical and contain the identifier present in the SKI extension of the certificate of the CA with respect to the signature key used.

The CRLDP and AIA extensions indicate access paths to the revocation information of the certificate. The CRLDP extension can contain one or several URLs (HTTP, FTP, LDAP) that point to a CRL, as does the AIA extension to an OCSP responder. A certificate must contain at least one means of verifying its revocation status.

> **R29** Offer revocation information
>
> At least one extension among CRLDP and AIA must be present and marked as non-critical.

3.2 Trust Establishment

Certificate Chains

The `Certificate` message sent by the server usually contains several certificates: the one that links the identity of the application server with the pair of asymmetrical keys used for key exchange, but also those that make it possible to establish a trusted link made of cryptographic signatures from the leaf certificate to a recognised certification authority.

In order for all clients to process this chain of trust without ambiguity, it is necessary to transmit it in an ordered manner and in full, from the leaf certificate to the intermediate certificate signed by the trusted root. It is not necessary to transmit the root certificate which makes it possible to check this last signature, because the client is already supposed to have it locally. However, in the case where the leaf certificate and the root certificate are the same self-signed certificate, it is still necessary to send it in order to disclose the identity of the server.

> **R30** Transmit an ordered and complete certificate chain
>
> The certificate chains which are sent using `Certificate` messages must be ordered and complete.

Revocation Checking

The setting up of revocation mechanisms is critical for assessing trust in certificates. The main revocation methods are presented in section 1.2. At least one of these mechanisms has to be implemented, and the URL of the CRL file or of the OCSP responder has to be present in the extensions of the certificates.

The choice from among these mechanisms must be based on an analysis of the needs in terms of security and availability, of the speed and time constraints as well as on the architecture of the information system. An OCSP responder provides fresher revocation information than the asynchronously downloaded CRL, but is has to be able to be reached constantly. Moreover, the size of some CRLs may form an obstacle in deploying them. In a massive revocation scenario, this performance constraint impacts more the CAs that are responsible for the distribution of the CRLs, which face denial of service risks [63].

As described in section 1.2, OCSP stapling provides improvements to the standard OCSP protocol:

- in terms of performance: the OCSP responder is not queried at each revocation checking because the TLS server caches the OCSP responses;
- in terms of privacy: the OCSP responder no longer knows the clients that connect to the services for which the certificate was issued.

> **R31** Prefer OCSP stapling
>
> When the OCSP protocol is implemented, it is recommended to use OCSP stapling. This solution is also preferred to the distribution of CRLs.

Most tools for certificate validation try to join the other URLs that may be present in the CRLDP (or AIA) extension, should the first one not respond. It is therefore advised to set up a redundant means for the publication of revocation information. In the case of CRLs, the CRL file can be hosted on several different web servers (or directories). In the case of OCSP, several OCSP servers can be implemented, sharing the same revocation information.

> **R32** Provide redundancy for revocation checking
>
> For reasons of availability, redundant mechasnims for the publication of revocation information must be implemented.

Revocation Information Deficiency

The services for checking revocation information, when they can be accessed, enable to validate or reject a certificate. In the event where none of them can be reached, two behaviours are defined: the hard-fail behaviour consists in refusing the connection, while the soft-fail behaviour consists in accepting the certificate all the same and continuing the exchange.

The choice between these two behaviours can depend on the context of use and the criticality of the application. An application that favours availability will be configured for soft-fail, while another wherein the need for security is predominant will activate the hard-fail. If soft-fail is preferred and the OCSP responder (or the server delivering the CRLs) becomes unavailable, then any attacker holding the key associated with a revoked certificate will still be able to impersonate the subject of the certificate.

> **R33** Operate in *hard-fail* mode
>
> The TLS software components must implement a *hard-fail* behaviour.

Certificate Transparency

In the framework of a public PKI, the CT programme presented in section 1.2 can provide additional assurance as to the validity of a certificate transmitted by a server. Currently, only the EV certificates are automatically checked by the TLS clients that are compatible with CT. However, certain CAs such as Symantec or Let's Encrypt record all of their new certificates.

> **R34** Use certificates registered to CT
>
> In the framework of a public PKI, it is recommended to use certificates that have been registered by their CA as part of the CT programme.

> **R35** Reject all certificates invalidated by CT
>
> TLS clients must reject all of the certificates which come with an invalid SCT, and the EV certificates which do not come with any SCT.

> **R35 −** Reject EV certificates invalidated by CT
>
> TLS clients must reject the EV certificates which come with an invalid SCT, or which do not come with any SCT.

Appendix A

Cipher Suites Guide

The following reference guide summarises the recommendations of section 2.2 into several tables of cipher suites.

A.1   Recommended Suites

Tables A.1 and A.2 list the cipher suites that are recommended in the general case, standardised for use with TLS 1.2. Recall that the use of CBC encryption mode is recommended if in conjunction with the `encrypt_then_mac` extension.

Value	Cipher suite
0xC02C	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
0xC02B	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
0xC0AD	TLS_ECDHE_ECDSA_WITH_AES_256_CCM
0xC0AC	TLS_ECDHE_ECDSA_WITH_AES_128_CCM
0xC024	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
0xC023	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256

Table A.1: Recommended suites for servers with an ECDSA key

Value	Cipher suite
0xC030	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
0xC02F	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
0xC028	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
0xC027	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256

Table A.2: Recommended suites for servers with an RSA key

Although the use of the AES encryption, which has been extensively tried out, is to be favoured, Camellia and ARIA are the object of no known attacks to date, and constitute acceptable alternatives. The corresponding suites are mentioned in tables A.3 and A.4. Finally, for deployments with pre-shared keys, the recommended suites are mentioned in table A.5.

Value	Cipher suite
0xC087	TLS_ECDHE_ECDSA_WITH_CAMELLIA_256_GCM_SHA384
0xC086	TLS_ECDHE_ECDSA_WITH_CAMELLIA_128_GCM_SHA256
0xC073	TLS_ECDHE_ECDSA_WITH_CAMELLIA_256_CBC_SHA384
0xC072	TLS_ECDHE_ECDSA_WITH_CAMELLIA_128_CBC_SHA256
0xC08B	TLS_ECDHE_RSA_WITH_CAMELLIA_256_GCM_SHA384
0xC08A	TLS_ECDHE_RSA_WITH_CAMELLIA_128_GCM_SHA256
0xC077	TLS_ECDHE_RSA_WITH_CAMELLIA_256_CBC_SHA384
0xC076	TLS_ECDHE_RSA_WITH_CAMELLIA_128_CBC_SHA256

Table A.3: Recommended suites for use with Camellia

Value	Cipher suite
0xC05D	TLS_ECDHE_ECDSA_WITH_ARIA_256_GCM_SHA384
0xC05C	TLS_ECDHE_ECDSA_WITH_ARIA_128_GCM_SHA256
0xC061	TLS_ECDHE_RSA_WITH_ARIA_256_GCM_SHA384
0xC060	TLS_ECDHE_RSA_WITH_ARIA_128_GCM_SHA256
0xC049	TLS_ECDHE_ECDSA_WITH_ARIA_256_CBC_SHA384
0xC048	TLS_ECDHE_ECDSA_WITH_ARIA_128_CBC_SHA256
0xC04D	TLS_ECDHE_RSA_WITH_ARIA_256_CBC_SHA384
0xC04C	TLS_ECDHE_RSA_WITH_ARIA_128_CBC_SHA256

Table A.4: Recommended suites for use with ARIA

Value	Cipher suite
0xC038	TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA384
0xC037	TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256

Table A.5: Recommended suites for use with a PSK

A.2 Relaxed Suites

When one of the two sides is not controlled, it is not always possible to negotiate a TLS session with one of the preceding cipher suites. In the case where a strong need for compatibility has been identified, other suites can be adopted. As explained in section 2.2, this widening of the possibilities for negotiation are done generally to the detriment of the security of communications. Consequently, it is suitable to evaluate the profile of the servers or of the clients in question, and to adopt only suites that are deemed as indispensable for the accomplishing of the application functions under consideration.

Relaxed Suites for TLS 1.2

In the absence of support for ECC, table A.6 lists the suites that are tolerated.

Value	Cipher suite
0x009F	TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
0x009E	TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
0xC09F	TLS_DHE_RSA_WITH_AES_256_CCM
0xC09E	TLS_DHE_RSA_WITH_AES_128_CCM
0x006B	TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
0x0067	TLS_DHE_RSA_WITH_AES_128_CBC_SHA256

Table A.6: TLS 1.2 suites tolerated in the absence of ECC support

In the absence of support for Diffie–Hellman exchanges, table A.7 lists the suites that are tolerated. They do not verify the PFS property.

Value	Cipher suite
0x009D	TLS_RSA_WITH_AES_256_GCM_SHA384
0x009C	TLS_RSA_WITH_AES_128_GCM_SHA256
0xC09D	TLS_RSA_WITH_AES_256_CCM
0xC09C	TLS_RSA_WITH_AES_128_CCM
0x003D	TLS_RSA_WITH_AES_256_CBC_SHA256
0x003C	TLS_RSA_WITH_AES_128_CBC_SHA256

Table A.7: TLS 1.2 suites tolerated in the absence of DH support

Table A.8 lists the suites tolerated for SRP use. The latter are based on a HMAC-SHA1 for the integrity check for messages, and therefore remain relevant for TLS 1.0.

Value	Cipher suite
0xC021	TLS_SRP_SHA_RSA_WITH_AES_256_CBC_SHA
0xC01E	TLS_SRP_SHA_RSA_WITH_AES_128_CBC_SHA

Table A.8: TLS suites tolerated for password authentication

Relaxed Suites for TLS 1.0

When support for TLS 1.0 is indispensable, the suites in table A.9 are to be favoured.

Value	Cipher suite
0xC00A	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
0xC009	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
0xC014	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
0xC013	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA

Table A.9: Suites tolerated for TLS 1.0

As a last resort, the suites in table A.10 are suggested. Using them is not recommended in the general case, for the reasons explained hereinabove.

Value	Cipher suite
0x0039	TLS_DHE_RSA_WITH_AES_256_CBC_SHA
0x0033	TLS_DHE_RSA_WITH_AES_128_CBC_SHA
0xC008	TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
0xC012	TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
0x0016	TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
0x0035	TLS_RSA_WITH_AES_256_CBC_SHA
0x002F	TLS_RSA_WITH_AES_128_CBC_SHA
0x000A	TLS_RSA_WITH_3DES_EDE_CBC_SHA

Table A.10: Suites barely tolerated for TLS 1.0

Downgraded suites based on a PSK or SRP key exchange, as well as those based on Camellia or ARIA for symmetrical encryption, are not represented.

The remaining suites that are not mentioned, involving in particular DSA certificates, DH or ECDH static key certificates, HMAC-MD5s, anonymous DH exchanges, or EXPORT algorithms, are strongly discouraged.

Appendix B

Examples of Applying the Recommendations

The following appendix presents applications for several software programs that use the TLS protocol, by aligning as closely as possible with the recommendations of the guide in light of their respective options. It is divided into two parts:

- the first part covers the generation of the OpenSSL library. It contains several directives for compilation for the use of this library in a given product or service;
- the second part focuses on the configuration of common application modules, `mod_ssl` for Apache and `ngx_http_ssl_module` for NGINX. It shows the directives that should be used to build a TLS context on the server side.

Application for the Compilation of OpenSSL

The chosen OpenSSL branch is 1.0.2. This is a LTS[41] version for which maintenance will be provided until 31 December 2019.

The OpenSSL build system is based on a set of shell scripts, Perl programs and Makefiles configured using `./config`. Extract B.1 shows the deactivation of the algorithms and the options that are not desirable, before running the compilation.

The compilation relates to two libraries of interest:

- `libcrypto.so`, for cryptographic algorithms;
- `libssl.so`, for the TLS protocol itself.

For use by other programs, these libraries can be selected through `LD_LIBRARY_PATH`, or explicitly during compilation at linking time.

---

[41]Long Term Support.

Listing B.1: Exemple of OpenSSL 1.0.2 compilation

```
# Remember to download the latest stable version of the branch 1.0.2
$ tar -xzf openssl-1.0.2h.tar.gz
$ cd openssl-1.0.2h
$ ./config shared -D_FORTIFY_SOURCE=2 -fstack-protector-all \
    -no-ssl2 -no-ssl3 -no-ssl2-method -no-ssl3-method \
    -no-ec2m \
    -no-weak-ssl-ciphers \
    -no-seed -no-idea \
    -no-mdc2 -no-md2 -no-md4 -no-whirlpool \
    -no-rc2 -no-rc4 -no-rc5 -no-blowfish -no-cast \
    -no-heartbeats \
    -no-srp -no-psk -no-comp
$ make depend
$ make
$ ls lib*.so
libcrypto.so libssl.so
```

Application to the Configuration of Application Modules for Apache and NGINX

Most applications have a TLS layer that makes it possible to configure them through directives that roughly control the options and the extensions pertaining to the protocol. It is not strictly necessary to use the previously generated library in order for these directives to be complied with. They are however highly dependent on the TLS library used as well as the version of it.

Here, we provide two examples of a configuration of TLS modules linked to OpenSSL: mod_ssl for Apache in branch 2.4, and ngx_http_ssl_module for NGINX in branch 1.10. Note that, in addition to the configuration directives that they provide, these modules can decide to apply additional parameters to the TLS layer without the integrator having a way to interact with it, unless the code in the module is modified directly.

The configuration extracts proposed here can be applied to separate virtual hosts. They focus on the parameters that are directly linked to TLS, and have to be supplemented using the technical note on securing websites [1].

Apache

Configuration B.2 supposes the use of an Apache server in version 2.4.88 or later. It offers recommended suites as well as certain relaxed suites. Several directives are not supported by prior versions [64].

Listing B.2: SSL Apache 2.4.8+ configuration example

```
# Activate TLS protocol
SSLEngine                on
# Force usage of TLS 1.2
SSLProtocol              all -SSLv2 -SSLv3 -TLSv1 -TLSv1.1
# Supply a certificate chain and a key
SSLCertificateKeyFile    "/path/to/private/key.pem"
SSLCertificateFile       "/path/to/certificate/chain.pem"
# Deactivate the session cache and the tickets
SSLSessionCache          none
SSLSessionTickets        off
# Deactivate compression
SSLCompression           off
# Prevent insecure renegotiations
SSLInsecureRenegotiation off
# Activate OCSP stapling
SSLUseStapling           on
SSLStaplingCache         shmcb:logs/ssl_stapling(32768)
# Use the recommended curves for ECDHE and supported_groups
SSLOpenSSLConfCmd ECDHParameters secp521r1:secp384r1:prime256v1:
    brainpoolP512r1:brainpoolP384r1:brainpoolP256r1
SSLOpenSSLConfCmd Curves secp521r1:secp384r1:prime256v1:brainpoolP512r1:
    brainpoolP384r1:brainpoolP256r1
# Use a 2048-bit DH group, generated by OpenSSL with
# openssl dhparam 2048 > /path/to/dhparams.pem
SSLOpenSSLConfCmd DHParameters "/path/to/dhparams.pem"


SSLHonorCipherOrder      on
# The branch 1.0.2 of OpenSSL :
# - does not support CCM suites;
# - does not support ARIA;
# - supports CAMELLIA (in revision `h'), but only combined with
#   SHA-1. The implementation of SHA256 with CAMELLIA128 is expected.
# Ordering relations used:
# - ECDHE > DHE > RSA encryption ;
# - GCM > CBC ;
# - AES256 > AES128 > CAMELLIA256 > CAMELLIA128 ;
# - SHA384 > SHA256 ;
# - ECDSA > RSA.
#
# With the current and future support provided, in branch 1.0.2:
SSLCipherSuite    ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:
    ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-
    AES256-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-
    AES128-SHA256:ECDHE-ECDSA-CAMELLIA256-SHA384:ECDHE-RSA-CAMELLIA256-SHA384:
    ECDHE-ECDSA-CAMELLIA128-SHA256:ECDHE-RSA-CAMELLIA128-SHA256:DHE-RSA-AES256
    -GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-SHA256:DHE-RSA-AES128
    -SHA256:AES256-GCM-SHA384:AES128-GCM-SHA256:AES256-SHA256:AES128-SHA256:
    CAMELLIA128-SHA256
```

NGINX

Configuration B.3 supposes the use of a NGINX server in version 1.6 or later. It corresponds to a situation where the client is controlled and wants to negotiate a TLS 1.2 session with a `TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384` cipher suite. Several directives are not supported by prior versions [65].

Listing B.3: SSL NGINX 1.6+ configuration example, faced with a controlled client

```
# Activate usage of the TLS protocol
listen                   443 ssl;

# Force usage of TLS 1.2
ssl_protocols            TLSv1.2;

# Supply the certificate chain and the key
ssl_certificate_key      /path/to/private/key.pem;
ssl_certificate          /path/to/certificate/chain.pem;

# Deactivate the session cache and the tickets
ssl_session_cache        none;
ssl_session_tickets      off;

# (Compression is deactivated by default since version 1.1.6)
# (Renegotiations are deactivated by default since version 0.8.24)

# Activate OCSP stapling
ssl_stapling             on;
ssl_stapling_verify      on;

# Use a recommended curve, in accordance with the controlled client
ssl_ecdh_curve           secp521r1;

# Use a recommended suite, in accordance with the controlled client
ssl_ciphers              ECDHE-ECDSA-AES256-GCM-SHA384;
```

Appendix C

List of Recommendations

Bibliography

[1] Agence nationale de la sécurité des systèmes d'information (ANSSI), "Recommandations pour la sécurisation des sites web." `<http://www.ssi.gouv.fr/uploads/IMG/pdf/NP_Securite_Web_NoteTech.pdf>`.

[2] Agence nationale de la sécurité des systèmes d'information (ANSSI), "Recommandations de sécurité concernant l'analyse des flux HTTPS." `<http://www.ssi.gouv.fr/uploads/IMG/pdf/NP_TLS_NoteTech.pdf>`.

[3] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and J. K. Zinzindohoue, "FREAK: Factoring RSA Export Keys." `<https://mitls.org/pages/attacks/SMACK#freak>`, March 2015.

[4] D. Adrian, B. Karthikeyan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelin, and P. Zimmermann, "Imperfect Forward Secrecy: How Diffie–Hellman Fails in Practice." `<https://weakdh.org/imperfect-forward-secrecy-ccs15.pdf>`, October 2015.

[5] N. Aviram, S. Schinzel, J. Somorovsky, N. Heninger, M. Dankel, J. Steube, L. Valenta, D. Adrian, J. A. Halderman, V. Dukhovni, E. Käsper, S. Cohney, S. Engels, C. Paar, and Y. Shavitt, "DROWN: Breaking TLS using SSLv2." `<https://drownattack.com/drown-attack-paper.pdf>`, March 2016.

[6] "Transport Layer Security – Applications and adoption." `<https://en.wikipedia.org/wiki/Transport_Layer_Security#Web_browsers>`.

[7] Qualys SSL Labs, "SSL/TLS Capabilities of Your Browser." `<https://www.ssllabs.com/ssltest/viewMyClient.html>`.

[8] T. Dierks and C. Allen, "The TLS Protocol Version 1.0." RFC 2246 (Proposed Standard), Jan. 1999. Obsoleted by RFC 4346, updated by RFCs 3546, 5746, 6176, 7465, 7507.

[9] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1." RFC 4346 (Proposed Standard), Apr. 2006. Obsoleted by RFC 5246, updated by RFCs 4366, 4680, 4681, 5746, 6176, 7465, 7507.

[10] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2." RFC 5246 (Proposed Standard), Aug. 2008. Updated by RFCs 5746, 5878, 6176, 7465, 7507, 7568, 7627, 7685, 7905.

[11] R. Barnes, M. Thomson, A. Pironti, and A. Langley, "Deprecating Secure Sockets Layer Version 3.0." RFC 7568 (Proposed Standard), June 2015.

[12] R. Braden, "Requirements for Internet Hosts - Communication Layers." RFC 1122 (INTERNET STANDARD), Oct. 1989. Updated by RFCs 1349, 4379, 5884, 6093, 6298, 6633, 6864.

[13] E. Rescorla and N. Modadugu, "Datagram Transport Layer Security Version 1.2." RFC 6347 (Proposed Standard), Jan. 2012. Updated by RFCs 7507, 7905.

[14] E. Rescorla, "Diffie–Hellman Key Agreement Method." RFC 2631 (Proposed Standard), June 1999.

[15] D. E. 3rd, "Transport Layer Security (TLS) Extensions: Extension Definitions." RFC 6066 (Proposed Standard), Jan. 2011.

[16] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile." RFC 5280 (Proposed Standard), May 2008. Updated by RFC 6818.

[17] Mozilla, "Network Security Services." `<https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS>`.

[18] Agence nationale de la sécurité des systèmes d'information (ANSSI), "Référentiel Général de Sécurité - Annexe A4." `<http://www.ssi.gouv.fr/uploads/2014/11/RGS_v-2-0_A4.pdf>`.

[19] Google, "CRLSets." `<https://dev.chromium.org/Home/chromium-security/crlsets>`.

[20] Mozilla, "Revoking Intermediate Certificates: Introducing OneCRL." `<https://blog.mozilla.org/security/2015/03/03/revoking-intermediate-certificates-introducing-onecrl/>`, March 2015.

[21] B. Laurie, A. Langley, and E. Kasper, "Certificate Transparency." RFC 6962 (Experimental), June 2013.

[22] Common Vulnerabilities and Exposures, "CVE-2009-3555." `<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3555>`, August 2009.

[23] J. Rizzo and T. Duong, "Browser Exploit Against SSL/TLS." `<https://packetstormsecurity.com/files/105499/Browser-Exploit-Against-SSL-TLS.html>`, October 2011.

[24] Common Vulnerabilities and Exposures, "CVE-2014-0160." `<https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2014-0160>`, April 2014.

[25] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3 – Version 12." `<https://tools.ietf.org/html/draft-ietf-tls-tls13-12>`, March 2016.

[26] B. Moeller and A. Langley, "TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks." RFC 7507 (Proposed Standard), Apr. 2015.

[27] A. Langley, W. Chang, N. Mavrogiannopoulos, J. Strombergson, and S. Josefsson, "ChaCha20-Poly1305 Cipher Suites for Transport Layer Security (TLS)." RFC 7905 (Proposed Standard), June 2016.

[28] Agence nationale de la sécurité des systèmes d'information (ANSSI), "Référentiel Général de Sécurité - Annexe B1." `<http://www.ssi.gouv.fr/uploads/2014/11/RGS_v-2-0_B1.pdf>`.

[29] Internet Assigned Numbers Authority, "Transport Layer Security (TLS) Parameters." `<https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml>`.

[30] P. Eronen and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)." RFC 4279 (Proposed Standard), Dec. 2005.

[31] D. Taylor, T. Wu, N. Mavrogiannopoulos, and T. Perrin, "Using the Secure Remote Password (SRP) Protocol for TLS Authentication." RFC 5054 (Informational), Nov. 2007.

[32] N. J. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering, and J. C. Schuldt, "On the Security of RC4 in TLS and WPA." `<https://cr.yp.to/streamciphers/rc4biases-20130708.pdf>`, July 2013.

[33] A. Popov, "Prohibiting RC4 Cipher Suites." RFC 7465 (Proposed Standard), Feb. 2015.

[34] S. Kelly, "Security Implications of Using the Data Encryption Standard (DES)." RFC 4772 (Informational), Dec. 2006.

[35] H. Y. Xiaoyun Wang, "Advances in cryptology – crypto 2005: 25th annual international cryptology conference, santa barbara, california, usa, august 14-18, 2005. proceedings," 2005.

[36] M. Stevens, P. Karpman, and T. Peyrin, "Freestart collision for full SHA-1." `<https://eprint.iacr.org/2015/967>`, 2016.

[37] M. J. Dworkin, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions." `<https://http://www.nist.gov/manuscript-publication-search.cfm?pub_id=919061>`, August 2015.

[38] N. J. AlFardan and K. G. Paterson, "Lucky thirteen: Breaking the TLS and DTLS record protocols," in *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pp. 526–540, IEEE Computer Society, 2013.

[39] Adam Langley, "Lucky Thirteen attack on TLS CBC." `<https://www.imperialviolet.org/2013/04/luckythirteen.html>`, February 2013.

[40] B. Möller, T. Duong, and K. Kotowicz, "This POODLE bites: Exploiting the SSL 3.0 Fallback," tech. rep., September 2014.

[41] Internet Assigned Numbers Authority, "Transport Layer Security (TLS) Extensions." `<https://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xhtml>`.

[42] S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)." RFC 4492 (Informational), May 2006. Updated by RFCs 5246, 7027.

[43] P. Gutmann, "Encrypt-then-MAC for Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)." RFC 7366 (Proposed Standard), Sept. 2014.

[44] K. Bhargavan, A. Delignat-Lavaud, A. Pironti, A. Langley, and M. Ray, "Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension." RFC 7627 (Proposed Standard), Sept. 2015.

[45] E. Rescorla, M. Ray, S. Dispensa, and N. Oskov, "Transport Layer Security (TLS) Renegotiation Indication Extension." RFC 5746 (Proposed Standard), Feb. 2010.

[46] S. Santesson, A. Medvinsky, and J. Ball, "TLS User Mapping Extension." RFC 4681 (Proposed Standard), Oct. 2006.

[47] N. Mavrogiannopoulos and D. Gillmor, "Using OpenPGP Keys for Transport Layer Security (TLS) Authentication." RFC 6091 (Informational), Feb. 2011.

[48] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer, "OpenPGP Message Format." RFC 4880 (Proposed Standard), Nov. 2007. Updated by RFC 5581.

[49] D. McGrew and E. Rescorla, "Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)." RFC 5764 (Proposed Standard), May 2010.

[50] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications." RFC 3550 (INTERNET STANDARD), July 2003. Updated by RFCs 5506, 5761, 6051, 6222, 7022, 7160, 7164.

[51] S. Friedl, A. Popov, A. Langley, and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension." RFC 7301 (Proposed Standard), July 2014.

[52] Y. Pettersen, "The Transport Layer Security (TLS) Multiple Certificate Status Request Extension." RFC 6961 (Proposed Standard), June 2013.

[53] A. Langley, "A Transport Layer Security (TLS) ClientHello Padding Extension." RFC 7685 (Proposed Standard), Oct. 2015.

[54] J. Salowey, H. Zhou, P. Eronen, and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State." RFC 5077 (Proposed Standard), Jan. 2008.

[55] M. Brown and R. Housley, "Transport Layer Security (TLS) Authorization Extensions." RFC 5878 (Experimental), May 2010.

[56] R. Seggelmann, M. Tuexen, and M. Williams, "Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension." RFC 6520 (Proposed Standard), Feb. 2012.

[57] P. Wouters, H. Tschofenig, J. Gilmore, S. Weiler, and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)." RFC 7250 (Proposed Standard), June 2014.

[58] S. Hollenbeck, "Transport Layer Security Protocol Compression Methods." RFC 3749 (Proposed Standard), May 2004.

[59] J. Rizzo and T. Duong, "The CRIME attack." <http://www.ekoparty.org/archive/2012/CRIME_ekoparty2012.pdf>, 2012.

[60] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Pironti, and P.-Y. Strub, "Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS." <https://www.mitls.org/downloads/tlsauth.pdf>, May 2014.

[61] M. Rex, "MITM attack on delayed TLS-client auth through renegotiation." <https://www.ietf.org/mail-archive/web/tls/current/msg03928.html>, November 2009.

[62] Agence nationale de la sécurité des systèmes d'information (ANSSI), "Référentiel Général de Sécurité - Annexe B2." <http://www.ssi.gouv.fr/uploads/2014/11/RGS_v-2-0_B2.pdf>.

[63] M. Prince, "The Hidden Costs of Heartbleed." <https://blog.cloudflare.com/the-hard-costs-of-heartbleed/>, April 2014.

[64] The Apache Software Foundation, "Apache Module mod_ssl." <https://httpd.apache.org/docs/2.4/en/ssl/ssl_howto.html>.

[65] Nginx, "Module ngx_http_ssl_module." `<http://nginx.org/en/security_advisories.html>`.

Acronyms

AEAD   Authenticated Encryption with Associated Data

AES   Advanced Encryption Standard

AIA   Authority Information Access

AKI   Authority Key Identifier

CA   Certification Authority

CRL   Certificate Revocation List

CRLDP   Certificate Revocation List Distribution Point

CT   Certificate Transparency

DES   Data Encryption Standard

DH   Diffie–Hellman

DHE   Diffie–Hellman Ephemeral

DTLS   Datagram Transport Layer Security

ECC   Elliptic Curve Cryptography

ECDH   Elliptic Curve Diffie–Hellman

ECDHE   Elliptic Curve Diffie–Hellman Ephemeral

EV   Extended Validation

FFDH   Finite Field Diffie–Hellman

FQDN   Fully Qualified Domain Name

HTTP   Hypertext Transfer Protocol

HTTPS   HTTP Secure

IANA   Internet Assigned Numbers Authority

IETF   Internet Engineering Task Force

LTS   Long Term Support

OCSP   Online Certificate Status Protocol

PFS	Perfect Forward Secrecy
PKI	Public Key Infrastructure
PRNG	Pseudorandom Number Generator
PSK	Pre-Shared Key
RGS	Référentiel Général de Sécurité
RTP	Real-time Transport Protocol
SAN	Subject Alternative Name
SCSV	Signaling Cipher Suite Value
SCT	Signed Certificate Timestamp
SKI	Subject Key Identifier
SNI	Server Name Indication
SRP	Secure Remote Password
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol

About ANSSI

The Agence nationale de la sécurité des systèmes d'information (ANSSI - French Network and Information Security Agency) was created on July 7, 2009 as an agency with national jurisdiction.

By Decree No. 2009-834 of July 7, 2009 as amended by Decree No. 2011-170 of February 11, 2011, the agency has responsibility at national level concerning the defence and security of information systems. It is attached to the Secrétaire général de la défense et de la sécurité nationale (Secretariat-General for National Defense and Security), under the authority of the Prime Minister.

For more information about ANSSI and its activites, please visit `www.ssi.gouv.fr`.

December 2016

Licence ouverte / Open Licence (Etalab v1)