

Recommandations de sécurité relatives à TLS



N°SDE-NT-35/ANSSI/SDE/NP

Document réalisé par l'ANSSI, mis en page à l'aide de L^AT_EX.

Version 1.2 : 26/03/2020

Vous pouvez envoyer vos commentaires et remarques à l'adresse suivante :

`guide.tls@ssi.gouv.fr`

Table des matières

Introduction	5
À qui s'adresse ce guide ?	7
Comment lire les recommandations ?	9
1 Présentation du protocole TLS	11
1.1 Déroulement des sessions TLS	11
1.2 Infrastructures de gestion de clés	15
2 Négociation des paramètres TLS	17
2.1 Versions de protocole	17
2.2 Suites cryptographiques	18
2.3 Extensions	28
2.4 Considérations additionnelles	36
3 Mise en place de l'IGC	43
3.1 Attributs des certificats X.509	43
3.2 Contrôle de validité	47
A Référentiel des suites cryptographiques	51
A.1 Suites recommandées	51
A.2 Suites dégradées	52
B Exemples d'application des recommandations	55
C Liste des recommandations	61
Bibliographie	63
Acronymes	69

Introduction

Le protocole TLS¹ est une des solutions les plus répandues pour la protection des flux réseau. Dans ce modèle client–serveur, les données applicatives sont encapsulées de manière à assurer la confidentialité, l’intégrité et empêcher leur rejeu. Le serveur est nécessairement authentifié, et des fonctions additionnelles permettent l’authentification du client si nécessaire.

Depuis l’apparition de son prédécesseur SSL² en 1995, TLS a été adopté par de nombreux acteurs de l’Internet pour sécuriser le trafic lié aux sites web et à la messagerie électronique. Il s’agit par ailleurs d’une solution privilégiée pour la protection de flux d’infrastructure internes. Pour ces raisons, le protocole et ses implémentations font l’objet d’un travail de recherche conséquent. Au fil des années, plusieurs vulnérabilités ont été découvertes, motivant le développement de corrections et de contre-mesures pour prévenir la compromission des échanges.

Le déploiement TLS apportant le plus d’assurance en matière de sécurité repose donc sur l’utilisation de logiciels mis à jour, mais aussi sur l’ajustement des paramètres du protocole en fonction du contexte. Les explications apportées par le présent guide sont complétées par plusieurs recommandations visant à atteindre un niveau de sécurité conforme à l’état de l’art, notamment au sujet des suites cryptographiques à retenir.

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

À qui s'adresse ce guide ?

Ce guide a pour objectif de présenter les bonnes pratiques relatives au protocole TLS.

Il s'adresse à tous les publics qui souhaitent se familiariser ou interagir avec le protocole TLS : responsables de la sécurité des systèmes d'information, administrateurs d'organismes de toutes tailles, ou encore développeurs de solutions souhaitant sécuriser des échanges d'information par l'intermédiaire de TLS.

Les caractéristiques d'une connexion TLS résultent d'un ensemble de facteurs qui sont rarement sous le contrôle intégral d'une seule et même entité. Elles dépendent globalement :

- des possibilités offertes par la pile TLS, en charge notamment de la logique d'automate et des calculs cryptographiques, telle que OpenSSL, GnuTLS ou encore mbed TLS ;
- des possibilités offertes par le logiciel qui exploite la pile TLS. Par exemple, la solution Apache dispose des modules `mod_ssl` et `mod_gnutls` pour servir des ressources HTTP³ en les protégeant, au choix, à l'aide de OpenSSL ou GnuTLS ;
- de la configuration du logiciel précédent. Par exemple, bien que le logiciel Apache prenne en charge par défaut les versions SSLv3, TLS 1.0, TLS 1.1, TLS 1.2 et TLS 1.3, les options de configuration permettent de ne jamais utiliser SSLv3 ;
- des capacités et de la configuration de l'interlocuteur TLS. Par exemple, face à un client qui propose d'utiliser TLS 1.2, un serveur Apache qui autorise l'usage de TLS 1.0, TLS 1.1 et TLS 1.2 choisira d'établir une session TLS 1.2.

Contrôler tout élément de cette ensemble de logiciels permet donc d'exclure certaines capacités indésirables. En revanche, il n'est pas toujours possible pour une entité isolée de d'utiliser les paramètres optimaux vis-à-vis des recommandations de sécurité, notamment pour des questions de compatibilité avec les clients attendus.

Les recommandations qui suivent s'abstraient de cette variété des intervenants en identifiant les caractéristiques souhaitables pour une connexion TLS, indépendamment des responsabilités de mise en œuvre. La mise en œuvre des recommandations avec les logiciels les plus courants fait l'objet de publications additionnelles de l'ANSSI [1, 2]. Des exemples d'application figurent également en fin de document, dans l'annexe B.

3. Hypertext Transfer Protocol.

Comment lire les recommandations ?

Ce guide dresse, par l'intermédiaire de ses recommandations, un ensemble de caractéristiques préférentielles pour une connexion TLS. Les recommandations portent en priorité sur le profil des flux échangés et ne préjugent pas des moyens nécessaires à leur mise en œuvre. La hiérarchie adoptée est précisée dans le tableau 1.

L'autonomie des recommandations permet aux différents acteurs de la connexion TLS, directs ou indirects, de les décliner selon leurs positions respectives. Par exemple, la **R4** recommande de ne jamais utiliser la version SSLv2. Pour un intégrateur, il s'agira de compiler la pile TLS exploitée afin que le logiciel développé ne prenne pas en charge SSLv2. Pour un administrateur, si le logiciel qu'il exploite prend en charge SSLv2, il s'agira d'exclure cette version à l'aide des options de configuration accessibles.

L'interprétation d'une recommandation varie par ailleurs selon les contextes.

Par exemple, la **R7** préconise d'utiliser l'algorithme ECDHE et définit la liste des courbes acceptables, tandis que la **R7-** tolère les échanges de clés DHE en utilisant les groupes de 2048-bits ou plus. Selon le contexte, il peut être nécessaire de prendre en compte des recommandations dérogatoire de type -. L'administrateur doit prendre en compte la nature des clients pour faire les choix adéquats.

Les raisonnements menés, pour l'essentiel, concernent de manière indifférenciée les clients et les serveurs TLS. Ils sont de plus indépendants de la nature des flux applicatifs protégés. Ainsi, les recommandations qui suivent s'appliquent aux flux HTTPS⁴ pris en charge par des serveurs web publics, aussi bien qu'aux flux d'infrastructure protégés par TLS au sein de certains systèmes industriels.

Rx	Cette recommandation permet de mettre en place l'architecture cible offrant un niveau de sécurité conforme à l'état de l'art.
Rx-	Dans le cas où l'application de la recommandation de sécurité optimale est impossible, ou insuffisante au regard des besoins en compatibilité, cette mesure propose un premier niveau dérogatoire. Le niveau de confiance est plus faible qu'avec Rx .

Table 1 – Hiérarchie des recommandations

4. HTTP Secure.

Lorsque TLS est déployé sur une infrastructure maîtrisée de bout en bout, les recommandations sont applicables sans restriction. Par exemple, entre la **R7** et la **R7-**, c'est la **R7** qui devrait être suivie : le serveur et ses clients doivent être configurés pour utiliser ECDHE exclusivement et refuser les échanges de clés DHE.

Dans le contexte distinct de la configuration d'un serveur TLS face à plusieurs clients dont les capacités ne sont pas contrôlées, plusieurs questions de compatibilité sont soulevées. Les paramètres idéaux peuvent en effet s'avérer trop restrictifs. Par exemple, des logiciels clients n'ayant pas fait l'objet de mises à jour récentes, et n'intégrant pas une partie des fonctions les plus recommandées seraient dans l'impossibilité de se connecter à un service.

Dans une telle situation, il convient d'établir un profil des clients susceptibles de se connecter au serveur, et d'évaluer en conséquence la permissivité de la configuration. Il est à noter que l'autorisation de certains paramètres déficients génère un risque non seulement pour les clients datés à la base de ce choix, mais parfois aussi pour le serveur et les clients à jour. Plusieurs vulnérabilités reposent notamment sur la capacité d'un attaquant à dégrader les paramètres de sécurité d'une connexion afin d'exploiter des fonctions faibles encore implémentées [3, 4, 5].

Une fois ce profil établi, les capacités des différents clients à se conformer aux recommandations de sécurité peuvent être évaluées. Dans le cas des navigateurs web, il existe des ressources publiques facilitant la mise en rapport des versions des navigateurs avec leurs capacités de sécurité liées à TLS [6, 7]. Par exemple, la plupart des clients Microsoft Internet Explorer 8 ne pourront pas se connecter à un serveur proposant exclusivement TLS 1.2, et nécessitent l'activation de TLS 1.0 et de certaines suites cryptographiques associées.

R1 Restreindre la compatibilité en fonction du profil des clients

Lorsque les clients d'un serveur ne sont pas maîtrisés, il convient d'établir un profil des clients souhaités. Le suivi éventuel de recommandations dégradées (**Rx-**) s'appuie sur ce profil.

Chapitre 1

Présentation du protocole TLS

1.1 Déroulement des sessions TLS

Le développement du protocole TLS a suivi plusieurs itérations [8, 9, 10, 11] depuis la conception du protocole SSL, désormais obsolète [12]. Le processus de standardisation est à la charge de l'IETF⁵. Les valeurs numériques des différents paramètres sont référencées par l'IANA⁶.

Les messages transmis par l'intermédiaire du protocole TLS sont appelés *records*. Ils sont généralement encapsulés dans des segments TCP⁷, protocole chargé d'assurer les fonctionnalités de transport réseau telles que l'acquittement à la réception de données [13]. Pour les protocoles à datagrammes, comme UDP⁸, une variante DTLS⁹ a été définie [14]. Il existe quatre types de *record*, expliqués ci-après : *handshake*, *change_cipher_spec*, *application_data* et *alert*.

Par souci d'interopérabilité, les spécifications permettent aux deux parties impliquées de négocier la version du protocole qu'ils adopteront. Ce paramètre est établi au cours d'une phase *TLS handshake*. De même, les spécifications autorisent l'utilisation de différentes combinaisons d'algorithmes cryptographiques. La suite cryptographique¹⁰ retenue pour la session est déterminée grâce aux messages de type *handshake*.

La figure 1.1 illustre la négociation de ces paramètres dans les versions TLS 1.2 et antérieures. Elle fait intervenir les échanges suivants entre le client et le serveur :

5. Internet Engineering Task Force.
6. Internet Assigned Numbers Authority.
7. Transmission Control Protocol.
8. User Datagram Protocol.
9. Datagram Transport Layer Security.
10. En anglais, *cipher suite*.

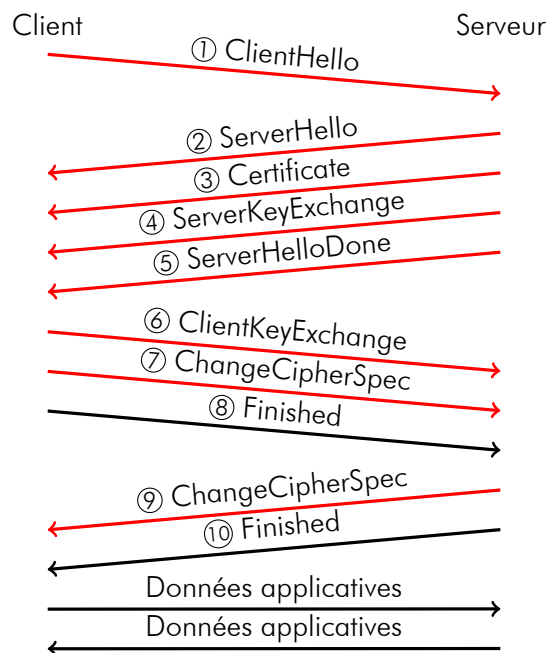


Figure 1.1 – Initiation générique d’une session TLS

1. le client initie une requête en envoyant un message de type `ClientHello`, contenant notamment les suites cryptographiques qu’il prend en charge ;
2. le serveur répond par un `ServerHello` qui contient la suite retenue ;
3. le serveur envoie un message `Certificate`, qui contient en particulier sa clé publique au sein d’un certificat numérique ;
4. le serveur transmet dans un `ServerKeyExchange` une valeur publique Diffie–Hellman qu’il signe à l’aide de la clé privée associée à la clé publique présent dans le certificat ;
5. le serveur manifeste sa mise en attente avec un `ServerHelloDone` ;
6. après validation du certificat et vérification de la signature précédente, le client poursuit l’échange de clé en choisissant à son tour une valeur publique Diffie–Hellman et en la transmettant dans un `ClientKeyExchange` ;
7. le client signale l’adoption de la suite négociée avec un `ChangeCipherSpec` ;
8. le client envoie un `Finished`, premier message protégé selon la suite cryptographique avec les secrets issus de l’échange de clé éphémère précédent ;
9. le serveur signale l’adoption de la même suite avec un `ChangeCipherSpec` ;
10. le serveur envoie à son tour un `Finished`, son premier message sécurisé.

Le cas générique décrit ici sous-entend l’adoption d’une des suites cryptographiques qui assurent la propriété de confidentialité persistante, ou PFS¹¹. Celle-ci consiste à empêcher le déchiffrement de messages de sessions passées quand bien même la clé

¹¹. Perfect Forward Secrecy.

privée liée au certificat du serveur serait compromise, en négociant un secret éphémère à l'aide d'un échange de clé Diffie–Hellman [15]. L'authentification du serveur repose alors sur la signature inscrite dans le `ServerKeyExchange`.

Les spécifications du protocole définissent par ailleurs plusieurs messages supplémentaires et extensions qui permettent d'encadrer et d'enrichir la protection des communications [10, 16]. Dans les contextes où un tel besoin aurait été identifié, le serveur est notamment en mesure de demander l'authentification du client au niveau TLS par l'intermédiaire d'un message `CertificateRequest`. En ce qui concerne les extensions, un `ClientHello` peut par exemple contenir des informations additionnelles relatives aux courbes elliptiques prises en charge par le client pour effectuer des calculs ECC¹².

Les spécifications de TLS 1.3 ont modifié la structure du *handshake*. La figure 1.2 illustre la négociation des paramètres pour une session TLS 1.3 :

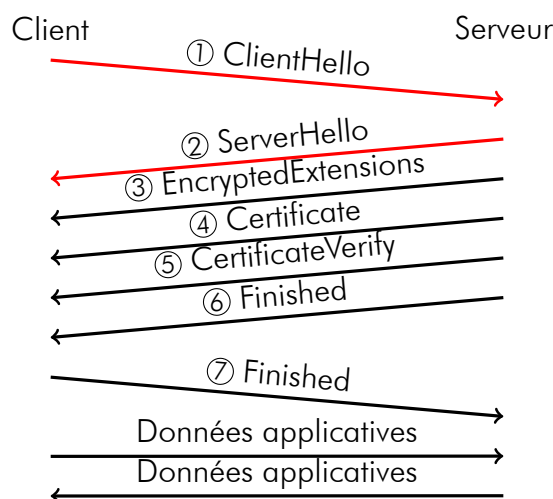


Figure 1.2 – Initiation générique d'une session TLS 1.3

1. le client initie une requête en envoyant un message de type `ClientHello`, contenant les suites cryptographiques qu'il prend en charge et des extensions ;
2. le serveur répond par un `ServerHello` qui contient la suite retenue et les extensions sélectionnées nécessaires à l'échange de clé cryptographique ;
3. le serveur envoie un message `EncryptedExtensions`, qui contient les autres extensions (non nécessaires à l'échange de clé crypto) pour cette session dont les paramètres cryptographiques ne dépendent pas ;
4. le serveur envoie un message `Certificate`, qui contient en particulier sa clé publique au sein d'un certificat numérique ;
5. le serveur s'authentifie auprès du client en transmettant `CertificateVerify` contenant des données signées par la clé privée associée à la clé publique précédente ;
6. le serveur envoie un `Finished`
7. le client envoie à son tour un `Finished`

12. Elliptic Curve Cryptography.



La version TLS 1.3 nécessite seulement un échange aller-retour¹³ pour compléter le *handshake* alors que les versions précédentes en nécessitaient deux. Cette modification permet au client de transmettre des données applicatives dès le troisième paquet transmis.

La réduction du nombre d'échanges nécessaires est due à la suppression de certains messages présents dans les versions précédentes. Les messages de signalisation **ChangeCipherSpec** et **ServerHelloDone** ont été supprimés, ainsi que les messages **ClientKeyExchange** et **ServerKeyExchange** utilisés pour échanger les valeurs publiques Diffie–Hellman.

Les valeurs publiques Diffie–Hellman s'échangent désormais avec une extension présente dans les messages **ClientHello** et **ServerHello**. L'échange de clé étant maintenant réalisé dès le début de la session, les messages suivants du *handshake* peuvent être transmis de manière chiffrée en utilisant des paramètres cryptographiques calculés à partir des valeurs Diffie–Hellman échangées dans les messages **{Client,Server}Hello**. Le premier message protégé du *handshake* est le message **EncryptedExtensions** permettant de préciser les extensions acceptées par le serveur et non nécessaires à l'établissement des paramètres cryptographiques.

Dans le cas général, la phase d'authentification du serveur est réalisée via les messages **Certificate** et **CertificateVerify** contenant la signature du condensat des messages échangés. Contrairement aux versions précédentes du protocole, cette phase d'authentification n'est donc plus réalisée en clair.

Comme pour les versions précédentes, une authentification du client peut également être exigée par le serveur avec l'envoi d'un message **CertificateRequest**. Dans ce cas, l'authentification du client reste similaire aux versions précédentes.

Pour permettre le chiffrement des messages du *handshake*, la fonction de dérivation de secret a été profondément modifiée. Elle est basée sur la primitive HKDF [17] qui utilise une fonction de hachage cryptographique.

Si le **ClientHello** propose une suite cryptographique acceptable pour le serveur mais dont il manque certaines informations (par exemple, une extension non présente dans le **ClientHello**), le serveur peut alors répondre un **HelloRetryRequest** indiquant l'ensemble des extensions attendues dans le **ClientHello**.

Pour toutes les versions du protocole TLS, le client et le serveur disposent d'un secret partagé appelé *master secret*. Le calcul du *master secret* diffère selon la version négociée. Il est ensuite dérivé afin de générer des clés qui permettent de protéger les données applicatives en confidentialité et en intégrité, avant de les encapsuler dans des messages de type **application_data**.

Les spécifications permettent de rafraîchir les clés de chiffrement sans interrompre la session TLS. Pour TLS 1.2, le rafraîchissement des clés est réalisé via une renégociation de session permettant également de redéfinir les paramètres de sécurité. Cette renégociation de session peut être déclenchée à l'initiative du client par l'intermédiaire d'un nouveau **ClientHello**. Elle peut aussi être sollicitée par le serveur à l'aide d'un message **HelloRequest**. TLS 1.3 ne supporte pas la renégociation telle que définie dans les versions précédentes. Le rafraîchissement de clé est réalisé par l'envoi d'un message **KeyUpdate** de la part du client ou du serveur. Dans TLS 1.3,

13. En anglais, *Round-Trip Time*.



il ne permet pas de renégocier la suite cryptographique ni d'établir un nouveau *master secret* contrairement à la renégociation. Lorsque ce message est transmis, cela permet de signaler que l'émetteur a modifié la clé de chiffrement en utilisant la fonction de dérivation de clé. Lors de l'envoi du message *KeyUpdate*, l'émetteur peut aussi solliciter le renouvellement des clés de la part du receveur.

Une nouvelle fonctionnalité apportée par TLS 1.3 est le mode 0-RTT. Ce mode permet de transmettre des messages protégés de type *application_data* dès le premier *record* envoyé par le client. Un secret doit être partagé afin d'utiliser ce mode, il peut être issu d'une session précédente.

Les messages de type *alert* permettent quant à eux de signaler des anomalies observées au cours du *handshake* ou bien de l'échange de données applicatives. Certains messages constituent juste des avertissements, tandis que d'autres appellent à une terminaison immédiate de la session TLS. Plusieurs codes d'alerte existent, qui permettent par exemple de signaler qu'un certificat transmis n'est pas valide, ou encore que le contrôle d'intégrité d'un *record* a échoué.

1.2 Infrastructures de gestion de clés

La validité des certificats envoyés pendant la négociation TLS est cruciale pour la vérification de l'identité des parties communicantes. L'ensemble des mécanismes et des entités qui garantissent cette validité et la maintiennent forment une infrastructure de gestion de clés.

Pour un certificat conforme à la norme X.509 [18] qui est suivie dans le cadre de TLS, l'assurance que la clé publique qu'il contient appartienne effectivement au serveur qu'il annonce en tant que sujet¹⁴ (généralement sous la forme d'un nom de domaine) repose sur la transmission de confiance depuis une autorité déjà reconnue jusqu'au serveur en question. Les liens de confiance successifs établis par chaque autorité de certification impliquée sont matérialisés par des signatures cryptographiques apposées aux différents certificats.

Ainsi, le message *Certificate* dont est extraite la clé publique sur laquelle s'appuient les secrets de session contient en réalité une chaîne de certificats, dont le client attend qu'elle forme un lien depuis une racine de confiance jusqu'au serveur interrogé. Ces racines sont généralement listées dans des registres : les magasins de certificats Microsoft, Apple et Debian, par exemple, maintiennent de tels registres, qui sont mis à disposition de toute application installée sur les systèmes d'exploitation associés. Pour sa part, Mozilla maintient un magasin propre, au travers de sa bibliothèque NSS [19].

Les certificats contiennent plusieurs attributs, comme une clé publique et une période de validité, qui sont habituellement complétés par des extensions X.509v3. En ce qui concerne ces attributs, l'ANSSI recommande le respect de l'annexe A4 du RGS¹⁵ [20]. Les extensions permettent notamment de préciser le cadre d'utilisation d'un certificat et de renforcer les assurances de l'IGC¹⁶. Par exemple, la présence d'une extension EV¹⁷ signale que des exigences addi-

14. Ce sujet peut provenir du champ *Subject* ou de l'extension *SubjectAltName*

15. Référentiel Général de Sécurité.

16. Infrastructure de Gestion de Clés.

17. Extended Validation.

tionnelles ont été portées au processus de vérification d'identité mené par l'AC¹⁸ avant qu'elle ne délivre le certificat.

Pour faire face aux erreurs ou aux attaques qui pourraient compromettre le fonctionnement d'une IGC, mais aussi pour accompagner des procédures nominales d'hygiène de sécurité telles que le renouvellement des clés, des mécanismes de révocation de certificat ont été définis. Deux solutions principales coexistent :

- les fichiers CRL¹⁹ : ces fichiers correspondent à des listes des certificats révoqués par une AC. Le maintien en ligne d'une CRL à jour fait partie des fonctions que doit assurer une IGC. L'emplacement de la CRL associée à une AC est renseigné dans l'extension CRLDP²⁰ de chaque certificat émis par cette AC ;
- le protocole OCSP²¹ : ce protocole, fonctionnant en mode client-serveur, permet à un client de vérifier en ligne la validité d'un certificat en interrogeant des répondeurs OCSP. Si une IGC met à disposition un service OCSP, l'emplacement des répondeurs associés doit être renseigné dans l'extension AIA²² de chaque certificat émis par l'AC.

Différentes initiatives cherchent à pallier les inconvénients respectifs de ces deux mécanismes, en particulier la taille des CRL qui exerce souvent une contrainte sur les ressources réseau, et le caractère synchrone des requêtes OCSP qui informe les répondeurs du profil d'une partie des connexions du client, ce qui peut être inacceptable du point de vue de la protection de la vie privée.

Pour le navigateur Chrome, l'usage des CRL et d'OCSP a été désactivé. Ces mécanismes ont été remplacés par un système *ad hoc*. Les équipes de Google identifient les révocations les plus significatives depuis les CRL mises à disposition par les AC, et les distribuent aux utilisateurs sous forme d'agrégations appelées CRLSets [21], dont le navigateur peut vérifier qu'il dispose de la dernière version. Mozilla a lancé un projet similaire pour son navigateur Firefox, sous le nom de OneCRL [22].

L'agrafage OCSP²³ est une solution alternative qui consiste pour le serveur à fournir directement une réponse OCSP parmi les extensions TLS du **ServerHello**, de sorte que le client n'ait plus à adresser directement les répondeurs [16].

En complément des mécanismes de révocation, le programme Certificate Transparency, initié par Google et standardisé par l'IETF [23], vise à créer des registres publics listant des certificats X.509. Ces registres permettent à un client TLS compatible avec Certificate Transparency de vérifier la validité de certains des certificats qui lui sont transmis. Ils permettent aussi de surveiller l'apparition de nouveaux certificats. La vérification se fait par l'intermédiaire de SCT²⁴ qui, horodatés et signés par les administrateurs des registres, constituent des assurances d'insertion du certificat. Les SCT sont transmis dans une extension TLS, une réponse OCSP, ou bien au sein du certificat lui-même.

18. Autorité de Certification.

19. Certificate Revocation List.

20. Certificate Revocation List Distribution Point.

21. Online Certificate Status Protocol.

22. Authority Information Access.

23. En anglais, *OCSP Stapling*.

24. Signed Certificate Timestamp.

Chapitre 2

Négociation des paramètres TLS

Le présent chapitre fait état des différents paramètres sur lesquels repose la sécurité d'une connexion TLS, sans préjuger de la nature des données applicatives à protéger ni du contexte de chiffrement. La sécurisation de sites web et la configuration de proxys TLS font l'objet de notes techniques complémentaires [1, 2].

De par son utilisation répandue, le protocole TLS est le sujet de nombreuses études qui aboutissent régulièrement à la découverte de nouvelles vulnérabilités [24, 25, 26, 4]. Compte tenu des corrections et des améliorations apportées au fil des années, à la fois aux spécifications et aux implémentations, il est essentiel d'utiliser les dernières versions des équipements et logiciels impliqués dans la sécurisation des communications.

R2 Utiliser des composants logiciels à jour

Les composants dont dépend le déploiement de TLS doivent être tenus à jour.

2.1 Versions de protocole

Depuis la publication de SSLv2 en 1995, l'identification de limitations du protocole a motivé plusieurs mises à jour de ses spécifications. Il existe à ce jour six déclinaisons du protocole exploitées : chronologiquement, SSLv2, SSLv3, TLS 1.0, TLS 1.1, TLS 1.2 et TLS 1.3 (cette dernière version ayant été standardisée en août 2018).

La version utilisée au cours d'une session est négociée pendant le *handshake*. Pour les versions antérieures à TLS 1.3, le client signale dans son `ClientHello` la version la plus récente du protocole qu'il prend en charge. Pour TLS 1.3, le client donne la liste des versions du protocole TLS qu'il supporte par ordre de préférence. Dans les deux cas, le serveur répond dans son `ServerHello` avec la version retenue pour la suite de la session.

Il est recommandé de prendre en charge la version TLS 1.3. Cette version propose uniquement des algorithmes cryptographiques à l'état de l'art. À ce jour, sous certaines conditions précisées dans la suite de cette section, TLS 1.2 reste considéré d'usage sûr.

R3 Privilégier TLS 1.3 et accepter TLS 1.2

La version TLS 1.3 doit être prise en charge et privilégiée. La version TLS 1.2 est également acceptée sous condition de suivre les recommandations de ce guide.

Il est recommandé de ne pas prendre en charge les autres versions du protocole TLS.

Le manque de robustesse de SSLv2 est établi de longue date. Le *handshake* faiblement protégé ou encore l'utilisation de primitives cryptographiques faibles exposent les échanges à plusieurs scénarios de compromission. Cette version est prohibée par l'IETF [27]. Par ailleurs, la vulnérabilité DROWN [5] a révélé que la simple prise en charge de SSLv2 par un serveur était susceptible de compromettre des sessions initiées avec des versions ultérieures du protocole.

De même, la version SSLv3 a également été déclarée obsolète par l'IETF [12]. Cette version du protocole n'est à ce jour plus considérée comme sûre.

Concernant les versions TLS 1.0 et TLS 1.1, un document est en cours d'élaboration par l'IETF afin de déclarer ces deux versions obsolètes [28]. Ces deux versions ne disposent pas de primitives cryptographiques à l'état de l'art. En effet, seules les fonctions de hachage MD5 et SHA-1 sont disponibles dans ces versions et celles-ci ne sont pas conformes au RGS.

R4 Ne pas utiliser SSLv2, SSLv3, TLS 1.0 et TLS 1.1

Les versions SSLv2, SSLv3, TLS 1.0 et TLS 1.1 sont à proscrire. De plus, il faut privilégier l'usage de composants logiciels qui ne prennent pas en charge ces versions du protocole, par exemple parce que le support de ces versions a été désactivé à la compilation.

2.2 Suites cryptographiques

Le `ClientHello` contient une liste de paramètres cryptographiques que le client est prêt à utiliser au cours de la session. Il est attendu du serveur qu'il sélectionne les paramètres cryptographiques de la session, après comparaison avec ceux qu'il prend en charge et accepterait d'utiliser. Cette sélection affecte la manière dont seront utilisées les clés cryptographiques pour protéger les *records* échangés après le *handshake*, qui transportent notamment les données applicatives. La procédure de négociation des clés est elle-même modifiée en fonction des paramètres retenus.

Les mécanismes cryptographiques négociés lors du *handshake* sont :

- un mécanisme d'échange de clé ;
- un mécanisme d'authentification symétrique ou asymétrique de la phase d'échange de clé ;
- des mécanismes assurant la confidentialité et l'intégrité des données échangées après le *handshake*, définis :
 - soit comme un mode de chiffrement intègre, aussi appelé mode combiné, offrant simultanément chiffrement et intégrité, tel `AES_256_GCM` ;
 - soit comme la composition d'un algorithme de chiffrement et d'une fonction de hachage utilisée en mode HMAC, telle `AES_256_CBC_SHA384` ;
- une fonction de hachage intervenant dans la dérivation pour les versions TLS 1.2 et supérieures, mais aussi dans les mécanismes d'authentification asymétrique.

Jusqu'à TLS 1.2, les algorithmes utilisés sont intégralement spécifiés dans la suite cryptographique. Par exemple, la suite `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384` référencée par l'IANA sous

le code 0xC030 représente l'association du mécanisme d'échange de clé ECDHE_ RSA avec le mode de chiffrement intègre AES_256_GCM complété de SHA384 pour la dérivation des secrets.

Dans TLS 1.3, une suite cryptographique détermine seulement l'algorithme de chiffrement et la fonction de hachage utilisée dans la dérivation des secrets. Par exemple, la suite TLS_AES_256_GCM_SHA384 référencée par l'IANA sous le code 0x1302 représente l'association de l'algorithme de chiffrement intègre AES_256_GCM complété de SHA384 pour la dérivation des secrets. L'algorithme d'échange de clé et le mécanisme d'authentification sont spécifiés à l'aide d'extensions.

Les recommandations de la présente section dressent une liste blanche des algorithmes et paramètres cryptographiques souhaitables : tout ce qui n'est pas recommandé est implicitement déconseillé. En particulier, l'usage de la fonction de chiffrement de flux RC4 et des fonctions de hachage MD5 et SHA-1 est à proscrire.

Échange de clé

Au cours du *handshake*, le client et le serveur négocient un algorithme d'échange de clés leur permettant de partager un *master secret* qui sera utilisé pour dériver les clés de chiffrement du trafic.

Les versions du protocole antérieures à TLS 1.3 proposent différents mécanismes d'échange de clé dont certains n'exigent pas l'authentification du serveur. Cependant, en l'absence de cette protection, l'échange de clé est exposé à des attaques par homme du milieu susceptibles de compromettre la sécurité de l'ensemble des échanges. Par conséquent, l'authentification du serveur est indispensable.

Il existe également la possibilité d'utiliser des certificats contenant uniquement une clé publique au format brut [29]. Ce type de certificat peut également être utilisé avec TLS 1.3. Cependant, l'utilisation de tels certificats ne permet pas de garantir l'authentification du serveur en tant que tel. Il est nécessaire pour cela d'établir un lien entre une clé publique reçue et l'entité attendue, et de vérifier la validité de ce lien de manière indépendante du protocole TLS. Pour ces raisons, l'utilisation de clé publique au format brut n'est donc pas recommandée.

R5 Authentifier le serveur à l'échange de clé

Au cours d'un échange de clé, le serveur doit être authentifié par le client. Les alternatives anonymisées de ces échanges ou reposant sur l'utilisation de certificat brut définies dans la RFC 7250 sont fortement déconseillées.

Dans les premières versions du protocole, l'échange de clé s'appuie, ou bien sur le chiffrement asymétrique d'un secret à l'aide de la clé publique du serveur (RSA), ou bien sur l'algorithme Diffie-Hellman, qui permettent tous deux l'établissement d'un secret commun de part et d'autre d'un canal non sécurisé.

L'algorithme DH²⁵, parfois désigné par FFDH²⁶, représente l'échange Diffie–Hellman historique dont l'arithmétique sous-jacente est relative à un groupe multiplicatif. L'arithmétique de l'algorithme ECDH²⁷ repose, quant à elle, sur une courbe elliptique.

Afin que l'éventuelle compromission de la clé privée du serveur ne permette pas de déchiffrer les communications passées, il est nécessaire de vérifier la propriété de confidentialité persistante (PFS²⁸). Celle-ci est assurée par les variantes éphémères des algorithmes précédents, DHE²⁹ et ECDHE³⁰, pour lesquelles les clés Diffie–Hellman sont générées à chaque nouvelle session.

R6 Échanger les clés en assurant toujours la PFS

La propriété de confidentialité persistante doit être assurée. Il faut pour cela employer une suite cryptographique reposant sur un échange Diffie–Hellman éphémère (ECDHE ou, à défaut, DHE).

L'utilisation d'échange de clé fondé sur RSA n'est donc pas recommandée puisqu'il ne permet pas de garantir la propriété de PFS. De plus, pour réaliser le chiffrement asymétrique du secret, le protocole TLS utilise le mode `RS_AES-PKCS1-v1_5`. Une attaque sur ce mode a été publiée en 1998 par Bleichenbacher [30]. Différentes contre-mesures ont été introduites dans les spécifications du protocole afin de mitiger cette attaque. Cependant, des études récentes montrent qu'il est difficile d'implémenter correctement ces contre-mesures [31, 32]. De même, les variantes statiques de l'algorithme Diffie–Hellman ne sont donc pas recommandées également puisqu'elles ne respectent pas la propriété de PFS.

Dans le cas général, TLS 1.3 permet de garantir la propriété de PFS puisqu'il met en œuvre uniquement les variantes éphémères de l'algorithme Diffie–Hellman.

Cependant, une variante d'implémentation de TLS 1.3 a été publiée par l'ETSI³¹ sous le nom d'*enterprise TLS* [33]. Cette variante de TLS 1.3 vise à protéger les communications internes au sein d'un réseau d'entreprise tout en permettant le déchiffrement passif de ces communications internes. Les communications externes au réseau d'entreprise sont protégées avec la version standard de TLS 1.3. La principale différence d'eTLS consiste en la suppression de la PFS. Pour cela, eTLS utilise des clés Diffie–Hellman statiques.

Bien que cette variante ne soit prévue que pour la protection des communications au sein d'un réseau d'entreprise, son utilisation diminue le niveau de sécurité par rapport au protocole TLS 1.3 et est donc déconseillée.

Afin d'utiliser les variantes éphémères de l'algorithme Diffie–Hellman, il est nécessaire de négocier un groupe dans lequel l'algorithme est employé. Avant TLS 1.3, la négociation de groupe

25. Diffie–Hellman.

26. Finite Field Diffie–Hellman.

27. Elliptic Curve Diffie–Hellman.

28. Perfect Forward Secrecy

29. Diffie–Hellman Ephemeral.

30. Elliptic Curve Diffie–Hellman Ephemeral.

31. European Telecommunications Standards Institute.

n'est pas permise par le protocole dans le cas de DHE. Le groupe est alors imposé par le serveur dans le message **ServerKeyExchange**. De ce fait, un client qui souhaiterait assurer la PFS avec DHE, mais se verrait présenter un groupe jugé insatisfaisant, n'aurait d'autre solution que d'interrompre la session. La négociation de groupe ECDHE est en revanche permise pour les versions précédentes de TLS 1.3 avec l'utilisation de l'extension **supported_groups**. Pour cette raison, l'usage de ECDHE est préférable à celui de DHE dès lors que l'une des parties communicantes n'est pas contrôlée.

TLS 1.3 a fait évoluer le protocole en donnant la possibilité au client de négocier les groupes DHE et les courbes elliptiques. Les groupes d'entiers et les courbes elliptiques utilisables ont également été inscrits dans les spécifications. Les groupes d'entiers retenus sont ceux définis dans [34] : **ffdhe2048**, **ffdhe3072**, **ffdhe4096**, **ffdhe6144** et **ffdhe8192**. Les courbes elliptiques retenues sont **secp256r1**, **secp384r1**, **secp521r1** (aussi appelées P-256, P-384 et P-521) ainsi que les courbes **x25519**, **x448**, **brainpoolP256r**, **brainpoolP384r** et **brainpoolP512r1**. La négociation de ces groupes est réalisée par l'extension **supported_groups**, détaillée dans la section 2.3.

Dans le cas d'ECDHE, pour une protection des données au-delà de 2020, le RGS préconise l'utilisation de groupes d'ordre multiple d'un nombre premier long d'au moins 256 bits [35]. L'ensemble des courbes retenues dans TLS 1.3 respectent le RGS.

R7 Échanger les clés avec l'algorithme ECDHE

Les échanges de clés ECDHE doivent être privilégiés, à l'aide des courbes **secp256r1**, **secp384r1**, **secp521r1**. Les courbes **x25519** et **x448** constituent des variantes acceptables. Les courbes **brainpoolP256r**, **brainpoolP384r** et **brainpoolP512r1** sont également acceptables.

Dans le cas de DHE, la sécurité de l'échange est liée à l'ordre du groupe multiplicatif en jeu. L'attaque Logjam [4] a illustré l'insuffisance des groupes de taille 512-bits, et pousse à déconseiller l'utilisation de groupes 1024-bits. Le RGS préconise l'utilisation de groupes 3072-bits ou plus, et tolère les groupes 2048-bits pour une protection des données jusqu'en 2030.

R7 – Échanger les clés avec l'algorithme DHE

Les échanges de clés DHE sont tolérés en utilisant les groupes 2048-bits ou plus (3072-bits ou plus si l'information doit être protégée au-delà de 2030) définis dans [34].

A noter

L'ensemble des groupes d'entiers et des courbes elliptiques retenus dans les spécifications de TLS 1.3 respectent le RGS.

Authentification

La recommandation **R5** demande l'authentification explicite du serveur. Le protocole TLS propose des mécanismes d'authentification symétrique basés sur une clé pré-partagée (PSK³²) [36] ou un mot de passe (SRP³³) [37] ainsi que des mécanismes d'authentification asymétrique utilisant un algorithme de signature.

Les algorithmes de signatures proposés jusqu'à TLS 1.2 sont RSASSA-PKCS1-v1_5, DSA et ECDSA. Parmi ces trois algorithmes, seul DSA n'est pas utilisable avec TLS 1.3. Les algorithmes RSASSA-PSS et EdDSA ont également été ajoutés dans les spécifications de TLS 1.3.

R8 Authentifier le serveur par certificat

Au cours d'un échange de clés, le serveur doit être authentifié par le client à l'aide d'un mécanisme asymétrique. Les méthodes basées sur ECDSA ou EdDSA sont à privilégier. Les méthodes basées sur RSA sont tolérées (RSASSA-PSS ou à défaut RSASSA-PKCS1-v1_5).

Les courbes recommandées pour ECDSA et EdDSA sont identiques à celles recommandées pour ECDHE (voir ci-dessus).

L'algorithme de signature étant lié au certificat du serveur, il se peut qu'une authentification basée sur ECDSA ne soit pas réalisable. Dans ce cas, l'authentification du serveur avec l'algorithme RSA est tolérée. Il convient alors de privilégier le mode RSASSA-PSS par rapport au mode RSASSA-PKCS1-v1_5. En effet, le mode RSASSA-PKCS1-v1_5 est vulnérable à une attaque permettant de forger une signature lorsque la clé publique RSA utilise un petit exposant (typiquement 3) et que le code de vérification de signature ne vérifie pas correctement la signature complète. Plusieurs vulnérabilités de ce type ont été trouvées dans différentes implémentations [38, 39, 40].

Cependant, le mode RSASSA-PSS n'est apparue qu'avec TLS 1.3. L'utilisation du mode RSASSA-PKCS1-v1_5 reste donc tolérée pour une session TLS 1.2 à condition de ne pas utiliser une clé publique disposant d'un petit exposant.

Les algorithmes de signature nécessitent une fonction de hachage. Les recommandations émises sur les fonctions de hachage plus loin dans ce guide sont applicables.

Les mécanismes d'authentification symétrique sont généralement plus complexes à déployer et à maintenir. Ils ne constituent des alternatives valables que dans les environnements maîtrisés, notamment pour des applications d'infrastructure. L'authentification symétrique peut néanmoins être utilisée pour effectuer une reprise de session avec TLS 1.3.

32. Pre-Shared Key.

33. Secure Remote Password.

R8 – Authentifier le serveur avec un mécanisme symétrique

Au cours d'un échange de clés, l'authentification du serveur par le client à l'aide d'un mécanisme symétrique est toléré.

Chiffrement symétrique

Le *master secret* négocié à l'aide de l'échange précédent est dérivé de part et d'autre en clés de chiffrement symétrique utilisées pour la protection en confidentialité des échanges qui suivent la phase de négociation. L'algorithme de chiffrement en jeu est cependant fixé dès la sélection d'une suite cryptographique. Le protocole TLS permet d'utiliser deux types d'algorithmes de chiffrement symétrique : les algorithmes de chiffrement par flot et des algorithmes de chiffrement par bloc.

Historiquement, le seul mode de chiffrement par flot disponible dans le protocole TLS était RC4. Cependant, suite à des travaux [41, 42] faisant état d'un biais statistique susceptible de mener à la compromission de données répétées dans un grand nombre de sessions TLS, tel un mot de passe ou un cookie HTTP, l'IETF a prohibé l'utilisation de cette fonction [43]. L'utilisation de RC4 est donc à proscrire.

Pour remplacer RC4, l'IETF a proposé l'algorithme de chiffrement ChaCha20 [44]. L'utilisation de cet algorithme a été retenue pour TLS 1.3. Ce dernier utilise des clés de chiffrement de 256 bits et aucune attaque n'est connue à ce jour sur cet algorithme. Cependant, il reste moins éprouvé dans le milieu académique que d'autres algorithmes de chiffrement par bloc.

Pour un algorithme de chiffrement par flot, le RGS préconise une taille de clé symétrique minimale de 128 bits et qu'aucune attaque nécessitant moins de 2^{128} opérations de calcul soit connue. À ce jour, l'algorithme ChaCha20 est conforme au RGS.

Plusieurs algorithmes de chiffrement par blocs ont été proposés dans les différentes versions du protocole TLS. Les algorithmes DES et Triple DES ont pendant longtemps été supportés mais ne sont plus considérés comme étant à l'état de l'art en terme de sécurité [45, 46]. L'utilisation de ces algorithmes est à proscrire.

En accord avec le RGS, l'utilisation de l'AES, le successeur du DES, est à privilégier. Dans le protocole TLS, AES peut être utilisé avec une clé de taille 128 bits ou 256 bits. Il n'existe à ce jour aucune attaque pratique qui remette en cause la confiance accordée à AES-128. Cependant, AES-256 étant jugé plus robuste, son utilisation est préférée à celle de AES-128 dans le présent document.

Avant TLS 1.3, il existe également des suites cryptographiques intégrant les algorithmes Camellia et ARIA. Bien que soumis à moins d'examen, ces deux algorithmes offrent à ce jour une sécurité comparable à AES et peuvent être envisagés comme alternatives. Le seul algorithme de chiffrement par bloc disponible dans TLS 1.3 est AES.

R9 Privilégier AES ou ChaCha20

Les suites mettant en œuvre l'algorithme de chiffrement par bloc AES sont à privilégier. L'algorithme de chiffrement par flot ChaCha20 constitue une alternative acceptable.

R9 – Tolérer Camellia et ARIA

Les suites mettant en œuvre les algorithmes de chiffrement par bloc Camellia et ARIA sont tolérées. La prise en charge de l'algorithme AES est conseillée, mais pas obligatoire.

Mode de chiffrement et d'intégrité

Pour les suites cryptographiques définies par les versions du protocole TLS antérieures strictement à la version 1.2, lorsqu'une suite cryptographique utilise un algorithme de chiffrement par bloc, le mode de chiffrement CBC est appliqué, combiné avec une instanciation de HMAC qui assure l'intégrité.

La combinaison s'effectue selon le schéma suivant : le calcul du HMAC porte sur un numéro de séquence, un entête, et les données en clair. Les données en clair et le motif d'intégrité issu de HMAC sont ensuite chiffrés suivant le mode CBC. Ce séquençement d'opérations introduit de possibles fuites d'information lors de l'opération de déchiffrement, pouvant potentiellement conduire à des attaques remettant en cause la confidentialité d'une donnée transmise au travers d'un certain nombre de sessions [47]. Pour éliminer de telles fuites d'information, l'implémentation doit réaliser les opérations de déchiffrement en temps constant. Cependant, des études récentes montrent que cela est difficile à réaliser et nécessite un gros effort d'implémentation [48].

Pour compenser cette vulnérabilité, l'extension `encrypt_then_mac` peut être utilisée. Dans ce cas, l'ordre des opérations est inversé : le HMAC porte sur les données déjà chiffrées. Cela permet d'éviter les fuites d'informations lors du déchiffrement et de prévenir certaines vulnérabilités associées [49, 47].

TLS 1.2 a introduit la possibilité d'utiliser des modes de chiffrement intègre, offrant de manière combinée une fonction de chiffrement et une fonction de calcul de motif d'intégrité.

TLS 1.3 propose exclusivement des modes de chiffrement combinés. Des suites offrant les modes d'opération GCM et CCM ont ainsi été standardisées ainsi que le mode `ChaCha20_Poly1305`

R10 Utiliser un mode de chiffrement intègre

La suite cryptographique retenue doit mettre en œuvre un mode de chiffrement intègre. Les modes GCM et CCM (hors CCM-8) sont à privilégier. Le mode combiné `ChaCha20_Poly1305` offre une alternative acceptable.

R10– Tolérer le mode CBC avec `encrypt_then_mac`

La combinaison CBC + HMAC en conjonction avec l'extension `encrypt_then_mac` est tolérée.

Les modes combinés GCM et Poly1305 nécessitent une attention particulière lors de la gestion des *nonces*. Dans ces modes, le chiffrement de chaque *record* nécessite l'utilisation d'un *nonce* unique au cours de la session. Si la propriété d'unicité du *nonce* n'est pas assurée, la confidentialité et l'intégrité des données échangées ne peuvent plus être garanties [50].

Les spécifications de TLS 1.2 ne précisent pas comment ce *nonce* doit être généré. Certaines piles TLS proposent des mécanismes de construction ne permettant pas de respecter la propriété d'unicité [51].

La construction des *nonces* a été intégrée aux spécifications de TLS 1.3, limitant ainsi le risque de mauvaise construction.

Fonction de hachage

Le protocole TLS nécessite une fonction de hachage pour réaliser de la dérivation des secrets, calculer la signature utilisée lors de l'authentification asymétrique et calculer des motifs d'intégrité lorsque le mode de chiffrement CBC est utilisé.

Pour la dérivation des secrets, TLS 1.0 et TLS 1.1 utilisent une combinaison des fonctions MD5 et SHA-1 tandis que les versions TLS 1.2 et TLS 1.3 utilisent soit SHA-256 ou soit SHA-384 selon la suite cryptographique sélectionnée.

Lorsque la suite cryptographique n'utilise pas de mode de chiffrement intègre, le motif d'intégrité est calculé par le mode HMAC, qui s'appuie sur une fonction de hachage ; les suites standardisées pour TLS permettent l'utilisation de MD5, de SHA-1 ou de SHA-256 ou SHA-384. La fonction de hachage est indiquée dans la suite cryptographique.

La fonction de hachage utilisée dans le mécanisme d'authentification asymétrique n'est pas indiquée directement par la suite cryptographique. Pour les versions TLS 1.0 et TLS 1.1, elle dépend de l'algorithme de signature utilisé pour authentifier l'échange de clé. Lorsque l'algorithme de signature est RSA, les fonctions de hachage MD5 et SHA-1 sont utilisées conjointement. Lorsque l'algorithme de signature est DSA, seule SHA-1 est utilisée. Afin de permettre de négocier la fonction de hachage utilisée, les spécifications de TLS 1.2 ont introduit l'extension `signature_algorithms` permettant au client de spécifier la fonction devant être utilisée par le serveur lors du calcul de la signature. Les spécifications de TLS 1.2 permettent l'utilisation de MD5, de SHA-1 ou d'un élément de la famille SHA-2 (SHA-256, SHA-384 ou SHA-512). Cependant, si le client n'utilise pas cette extension, SHA-1 est utilisée par défaut par le serveur.

TLS 1.3 utilise également l'extension `signature_algorithms` mais rend son utilisation obligatoire et permet d'utiliser uniquement des fonctions de hachage de la famille SHA-2.

La fonction MD5 a fait l'objet de nombreuses attaques, qui ont motivé son exclusion du RGS. Depuis 2005, plusieurs études ont mis en défaut la supposée robustesse de SHA-1 [52, 53].

La publication en 2017 d'une collision pour la fonction SHA-1 ne permet plus de considérer cette fonction de hachage comme sûre [54].

Pour toute utilisation d'une fonction de hachage dans le protocole TLS, il est recommandé d'utiliser uniquement les éléments de la famille SHA-2.

Il faut noter qu'à la date de publication de ce guide, aucune version du protocole TLS ne fait l'usage de la fonction SHA-3 [55] standardisée en 2015.

R11 Utiliser SHA-2 comme fonction de hachage

Les fonctions de hachage de la famille SHA-2 doivent être utilisées.

Synthèse

Les suites cryptographiques répondant aux exigences précédentes et recommandées dans le cadre général de l'utilisation du protocole TLS figurent dans les tableaux 2.1, 2.2 et 2.3. Pour rappel, l'extension `encrypt_then_mac` est recommandée pour l'utilisation des suites exploitant le mode de chiffrement CBC.

Code TLS	Suite cryptographique
0x1302	TLS_AES_256_GCM_SHA384
0x1301	TLS_AES_128_GCM_SHA256
0x1304	TLS_AES_128_CCM_SHA256
0x1303	TLS_CHACHA20_POLY1305_SHA256

Table 2.1 – Suites TLS 1.3 recommandées

Code TLS	Suite cryptographique
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
0xCCA9	TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256

Table 2.2 – Suites TLS 1.2 recommandées avec un serveur disposant d'un certificat avec clé publique ECDSA

Lorsqu'une des deux parties communicantes n'est pas maîtrisée, il n'est pas toujours possible de négocier une session TLS avec une des suites cryptographiques précédentes. L'annexe A fait état des suites de sécurité moindre pouvant être envisagées pour répondre à des besoins forts de compatibilité. Par ailleurs, la section suivante apporte des recommandations supplémentaires relatives au contexte de déploiement.

Code TLS	Suite cryptographique
0xC030	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
0xC02F	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
0xCCA8	TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256

Table 2.3 – Suites TLS 1.2 recommandées avec un serveur disposant d'un certificat avec clé publique RSA

Contextes de déploiement

Dans la situation où les profils de serveur et de client sont maîtrisés, si le serveur dispose d'un certificat avec une clé ECDSA ou RSA, alors il est suffisant pour le serveur et le client de n'utiliser qu'une seule des suites cryptographiques des tableaux 2.1, 2.2 ou 2.3, par exemple TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 pour une sessions TLS 1.2 ou TLS_AES_256_GCM_SHA384 pour une session TLS 1.3, avec la courbe elliptique `secp256r1`.

Dans l'éventualité de la découverte d'un problème de sécurité lié à la suite sélectionnée, il reste toutefois préférable que chacune des parties communicantes dispose d'une implémentation des autres suites des tableaux 2.1, 2.2 ou 2.3 qui puisse être activée ultérieurement par configuration. Ce principe s'applique aussi aux courbes elliptiques recommandées précédemment.

R12 Disposer de plusieurs suites cryptographiques

À des fins préventives, les parties communicantes doivent implémenter plusieurs suites acceptables. Lorsque l'infrastructure est maîtrisée de bout en bout, elles peuvent ensuite n'en utiliser qu'une seule.

Lors de la phase de négociation, le client transmet au serveur une liste de suite cryptographique ordonnée selon son ordre de préférence dans le `ClientHello`. Le serveur sélectionne une suite parmi cette liste. Cependant, les spécifications du protocole TLS ne précisent pas comment ce choix doit être réalisé. Le serveur peut sélectionner la suite en privilégiant les préférences du client ou bien les siennes. En l'absence de maîtrise des clients, le serveur doit privilégier les préférences des suites cryptographiques établies dans sa propre configuration, plutôt que celles du client.

R13 Préférer l'ordre de suites du serveur

Lorsque les clients d'un serveur ne sont pas maîtrisés, l'ordre des suites cryptographiques qui figure dans sa configuration doit prévaloir sur l'ordre des suites signalées par les clients.

2.3 Extensions

Le `ClientHello` envoyé par le client en début de session est susceptible de contenir un ensemble d'extensions. Celles-ci permettent généralement au client d'informer le serveur de ses diverses capacités, comme par exemple sa prise en charge des signatures ECDSA. Elles permettent aussi de fournir au serveur des informations complémentaires, telles que le nom de domaine qu'il souhaite joindre. Le serveur confirme sa propre prise en charge et sa volonté d'exploiter certaines des capacités du client en insérant un sous-ensemble des extensions du `ClientHello` dans son `ServerHello`. Chaque extension est identifiée par un entier codé sur deux octets, enregistré auprès de l'IANA [56].

Selon le contexte, ces extensions peuvent être purement informatives, ou bien indispensables au déroulement de la session TLS. Dans une session TLS 1.3, certains paramètres de la session comme le numéro de version et le mode d'échange de clé sont négociés à l'aide de ces extensions et sont donc indispensables. Dans d'autres cas, l'utilisation de certaines extensions est souvent conditionnée à une application particulière. De ce fait, les extensions dont l'utilisation est acceptable pour certains contextes n'ont pas vocation à être utilisées pour toutes les sessions et certaines extensions ne sont valables que pour une version donnée. Les extensions déconseillées, par contre, le sont en toutes circonstances. Un client ne devrait pas les envoyer dans un `ClientHello`, et si un serveur ou un client reçoit l'une d'entre elles, il doit l'ignorer. L'absence de prise en charge de ces extensions permet par ailleurs de réduire la surface d'attaque des applications impliquées dans l'échange.

Extensions recommandées dans le cadre général

- `supported_groups` (0x000A) [57]

Cette extension signale les groupes supportés et négociés pour réaliser l'échange de clé lors du *handshake*. Avant TLS 1.3, cette extension était appelée "*elliptic_curves*" et était utilisée uniquement pour les courbes elliptiques. Elle a été renommée pour TLS 1.3 afin de permettre également la négociation de groupe d'entier. L'ordre dans lequel les groupes sont présentés reflète les préférences du client.

- `signature_algorithms` (0x000D) [10]

Cette extension signale les algorithmes de hachage et de signature pris en charge pour vérifier l'authenticité des futurs messages du *handshake*. Dans le cadre recommandé d'une session avec échange de clé authentifié, sa prise en charge et son usage sont obligatoires par le client et le serveur, et la prise en charge d'au moins un représentant de la famille SHA-2 devrait être annoncée.

- `signed_certificate_timestamp`, ou `sct` (0x0012) [23]

Cette extension signale la prise en charge de SCT. Dans le modèle CT³⁴ présenté en section 1.2, un SCT peut être requis pour établir la validité du certificat présenté par le serveur. Transmettre un SCT dans le champ de données de cette extension TLS constitue une alternative à la présence d'un SCT parmi les extensions du certificat, ou bien parmi les extensions d'un statut OCSP associé.

34. Certificate Transparency.

Extensions recommandées pour une session TLS 1.3

- `supported_versions` (0x002B) [11]

Cette extension est utilisée dans la version TLS 1.3 pour négocier la version utilisée lors du *handshake*. L'usage de cette extension est obligatoire lorsque la version TLS 1.3 est utilisée.

- `key_share` (0x0033) [11]

Cette extension est utilisée pour échanger les paramètres cryptographiques lors d'un échange de clé (EC)DHE. Elle est recommandée car elle permet de garantir la propriété de confidentialité persistante (PFS).

- `supported_groups` (0x000A) [11]

Cette extension est utilisée dans la version TLS 1.3 pour préciser le group (EC)DHE que le client supporte.

Extensions recommandées pour une session TLS 1.2

Ces extensions ne sont utiles que pour TLS 1.2, et les problèmes de sécurité qu'elles résolvent sont traitées dans la spécification du protocole TLS 1.3.

- `encrypt_then_mac` (0x0016) [58]

Cette extension signale la prise en charge de la construction cryptographique *encrypt-then-mac*, en remplacement de la construction *mac-then-encrypt* historique. L'usage de cette extension est fortement recommandé dès lors que le mode de chiffrement CBC est utilisé.

- `extended_master_secret` (0x0017) [59]

Cette extension signale la capacité à calculer un *extended master secret*. Le procédé de calcul de l'*extended master secret* est plus robuste que celui du *master secret* historique, car il ne s'appuie plus uniquement sur les aléas contenus dans le `ClientHello` et le `ServerHello`, mais aussi sur l'ensemble du contexte cryptographique négocié (suites, méthode d'échange de clé, certificats), à travers un condensat de tous les messages échangés au cours du *handshake*.

- `renegotiation_info` (0xFF01) [60]

Le mécanisme de renégociation initialement conçu pour les versions de TLS inférieures ou égales à 1.2 expose le client à une vulnérabilité protocolaire. Il est en effet possible pour un attaquant de faire passer la première négociation d'un client pour une renégociation. L'attaquant se retrouve ainsi en position d'injecter des données applicatives que le serveur attribue au client légitime. L'extension `renegotiation_info` a été définie afin d'effectuer des renégociations sécurisées. L'utilisation de cette extension demande de conserver le contenu protégé des messages `Finished` utilisés pour authentifier le dernier *handshake*.

Extensions acceptables dans le cadre général

- `server_name` (0x0000) [16]

Cette extension signale la prise en charge du mécanisme SNI³⁵. Elle permet au client de préciser le nom de domaine du serveur qu'il souhaite joindre. Son usage est répandu et permet notamment d'héberger plusieurs serveurs TLS derrière une même adresse IP. Lorsque cette extension est utilisée avec la version TLS 1.3, elle révèle cependant l'identité du serveur contacté. Des discussions sont en cours au sein de l'IETF afin de ne pas transmettre le nom du domaine du serveur en clair.

- `status_request` (0x0005) [16]

Cette extension signale la prise en charge du mécanisme d'agrafage OCSP. Le champ de données de l'extension envoyée par le client liste les répondeurs OCSP considérés de confiance. Si le serveur prend en charge ce mécanisme, l'extension qu'il envoie à son tour ne contient aucune donnée, mais son message `Certificate` est immédiatement suivi d'un message `Certificate Status` contenant une réponse OCSP.

- `use_srtp` (0x000E) [61]

Cette extension signale la prise en charge de DTLS-SRTP. SRTP est une variante sécurisée de RTP³⁶, un protocole optimisé pour le transfert de données en temps réel, telles que des flux vidéo [62]. Les spécifications de RTP ne couvrent pas la négociation de paramètres de session et de clés, ce à quoi peut répondre DTLS, une variante de TLS destinée aux communications via datagrammes. L'utilisation et la prise en charge de cette extension ne sont généralement pas nécessaires.

- `application_layer_protocol_negotiation`, ou `alpn` (0x0010) [63]

Cette extension signale la prise en charge de la négociation de protocoles applicatifs. Elle permet de marquer une préférence parmi différents protocoles applicatifs partageant un même port protégé par TLS. L'extension vise principalement à permettre aux clients d'annoncer leur prise en charge de HTTP/2, la plus récente version du protocole HTTP. Sa prise en charge et son usage sont recommandés lorsque l'utilisation de HTTP/2 est souhaitée.

- `padding` (0x0015) [64]

Cette extension permet au client d'ajouter du bourrage³⁷ au `ClientHello` sous la forme d'octets nuls. Elle a été exceptionnellement définie en réponse à la détection d'un bug lié à la longueur des messages `ClientHello`. L'utilisation et la prise en charge de cette extension ne sont généralement pas nécessaires.

- `record_size_limit` (0x001C) [65]

Cette extension permet de négocier la taille maximum des fragments du message d'origine. Elle remplace l'extension `max_fragment_length` en ajoutant la possibilité au serveur de sélectionner une taille de fragment. Avec l'extension `max_fragment_length`, le serveur devait retourner la valeur sélectionnée par le client. Ce cas pouvait être problématique lorsque le serveur disposait de capacité limitée par rapport à un client.

35. Server Name Indication.

36. Real-time Transport Protocol.

37. En anglais, *padding*.

Extensions acceptables pour TLS 1.3

- `pre_shared_key` (0x0029) [11]

Cette extension permet de spécifier l'identité d'une clé pré-partagée (PSK) utilisée pour authentifier une session TLS 1.3. Cette PSK peut avoir été partagée de manière indépendante du protocole en amont de l'établissement de la session, ou issue d'une session TLS 1.3 précédente. Cette extension est acceptable dans le cadre d'une reprise de session mais n'est pas recommandée lorsque la PSK est établie indépendamment du protocole TLS (voir la section 2.4 pour plus de détails)

- `cookie` (0x002C) [11]

Cette extension reprend un mécanisme de protection contre les attaques par déni de service défini dans DTLS. Il permet à un serveur de répondre à un `ClientHello` par un `HelloRetryRequest` en y incluant une extension `cookie`, il est attendu alors du client qu'il renvoie un même `ClientHello` complété de l'extension `cookie`. Cette extension peut également avoir un intérêt pour décharger le serveur lorsque celui-ci est trop chargé.

- `psk_key_exchange_modes` (0x002D) [11]

Cette extension est utilisée en complément de l'extension `pre_shared_key` lorsque une authentification par PSK est réalisée. Elle permet de spécifier si un échange de clé supplémentaire est réalisé afin d'obtenir la propriété de confidentialité persistance.

- `certificate_authorities` (0x002F) [11]

Cette extension permet de spécifier la liste des AC supportées par un client ou un serveur. Pour cela, le client ou le serveur fournit dans le champ de données de l'extension les noms des AC dans lesquelles il a confiance. Elle remplace l'extension `trusted_ca_keys` utilisable dans les versions précédentes mais qui n'est pas utilisée dans TLS 1.3.

- `oid_filters` (0x0030) [11]

Cette extension permet au serveur d'exiger une liste d'extensions attendues dans le certificat du client. Elle permet notamment de fixer des valeurs attendues pour les extensions *Key Usage* et *Extended Key Usage*. Cette extension n'est utilisée que dans le cas d'une authentification du client.

- `post_handshake_auth` (0x0031) [11]

Cette extension permet au client de signaler qu'il a la possibilité de s'authentifier auprès du serveur après l'établissement de la session. L'authentification du client peut être sollicitée par le serveur via l'envoi d'un message `CertificateRequest`. Cette fonctionnalité nouvelle de TLS 1.3 peut être utilisée dans le cas où une authentification client est nécessaire selon le trafic.

- `signature_algorithms_cert` (0x0032) [11]

Cette extension signale les algorithmes de hachage et de signature pris en charge dans un certificat. L'utilisation de cette extension n'est généralement pas nécessaire puisqu'en cas d'absence de celle-ci, les algorithmes de hachage et de signature présents dans l'extension `signature_algorithms` s'applique également à la signature présente dans le certificat.

Extensions acceptables pour TLS 1.2

- `trusted_ca_keys` (0x0003) [16]

Cette extension signale la prise en charge d'identifiants de certificat, guidant éventuellement le serveur dans la sélection de la chaîne de certificats qu'il envoie dans son message `Certificate`. Pour cela, le client fournit dans le champ de données de l'extension les noms des AC dans lesquelles il a confiance, ou bien les condensats des certificats correspondants. Cette extension a un intérêt lorsqu'un client ne dispose que d'un nombre d'AC restreint. La liste des AC que possèdent un client peut être une information sensible.

- `user_mapping` (0x0006) [66]

Cette extension signale la prise en charge de l'envoi de données supplémentaires dans un nouveau message `SupplementalData` de la part du client, dont il est attendu qu'elles permettent au serveur de l'identifier plus rapidement. Elle est notamment utile dans la situation où le serveur dispose d'un index des clients avec lesquels la communication est autorisée.

- `srp` (0x000C) [37]

Cette extension signale la prise en charge du protocole SRP par le client, et contient par ailleurs un nom d'utilisateur permettant au serveur de décider des paramètres envoyés dans le `ServerKeyExchange`. L'authentification du serveur et du client s'appuie de part et d'autre sur la connaissance d'un secret qui nécessite le même niveau de protection que les clés privées dans le cadre d'authentification classique. L'usage de cette extension est associé à celui des suites cryptographiques SRP. Dans cette situation, afin de renforcer son authentification, il est fortement recommandé que le serveur dispose d'un certificat associé à une clé ECDSA ou RSA permettant de signer le `ServerKeyExchange`.

- `status_request_v2` (0x0011) [67]

Cette extension complète les fonctionnalités de l'extension `status_request` vis-à-vis du mécanisme d'agrafage OCSP, présenté en section 1.2. Elle permet au serveur d'envoyer au client une liste de messages `CertificateStatus`, qui contient non seulement le statut de son propre certificat, mais aussi celui de certificats intermédiaires de la chaîne de certificats qu'il a précédemment présentée dans son message `Certificate`.

- `session_ticket` (0x0023) [68]

Cette extension signale la prise en charge du mécanisme de tickets de session. À sa réception dans un `ClientHello`, si le serveur prend en charge ce mécanisme, il en confirme l'utilisation en annonçant à son tour l'extension dans son `ServerHello`. Ou bien l'extension du `ClientHello` ne contenait pas de donnée supplémentaire, auquel cas le serveur délivre un ticket de session qui contient les paramètres de la négociation courante sous une forme cryptographiquement protégée. Ou bien l'extension du `ClientHello` contenait un ticket de session obtenu au préalable, dont l'intégrité cryptographique est vérifiée par le serveur, et les paramètres de négociation associés sont restaurés le cas échéant.

Extensions déconseillées

- `max_fragment_length` (0x0001) [16]

Cette extension signale la prise en charge de la fragmentation réduite. Elle permet de ramener à 2^{12} , 2^{11} , 2^{10} ou 2^9 la longueur standard maximale de 2^{14} octets définie pour les fragments du message d'origine traités pour la construction des *records*. Elle est obsolète et doit être remplacée par l'extension `record_size_limit`.

- `client_certificate_url` (0x0002) [16]

Cette extension signale la prise en charge de l'authentification client par l'intermédiaire de certificats distants, localisés par une ou plusieurs URL³⁸. Celles-ci sont transmises dans un message `CertificateURL`, qui remplace le message `Certificate` habituellement émis par le client. Ce mécanisme peut être détourné pour forcer un serveur à effectuer un nombre significatif de requêtes (HTTP, FTP, et plus particulièrement HTTPS) auprès de différents hôtes. L'extension n'est généralement pas nécessaire et sa prise en charge, en-dehors d'environnements maîtrisés, est par conséquent déconseillée.

- `truncated_hmac` (0x0004) [16]

Cette extension signale la prise en charge des HMAC tronqués, qui consistent à ne suffixer chaque *record* qu'avec les dix premiers octets du HMAC originel. Cette construction affaiblit le mécanisme d'authentification et sa prise en charge est déconseillée [69].

- `client_authz` (0x0007) [70]

Cette extension permet de signaler la capacité du client à envoyer des données d'authentification supplémentaires dans des messages `SupplementalData`, qui permettent en particulier au serveur de savoir avec quelles applications interfacier le client. Ces informations d'authentification ne sont pas relatives au protocole, par conséquent la prise en charge de cette extension est déconseillée.

- `server_authz` (0x0008) [70]

Cette extension permet de signaler la capacité du serveur à envoyer des données d'authentification supplémentaires dans des messages `SupplementalData`, qui permettent par exemple au client de connaître la réputation du serveur. Ces informations d'authentification ne sont pas relatives au protocole, par conséquent la prise en charge de cette extension est déconseillée.

- `cert_type` (0x0009) [71]

Cette extension permet de signaler la prise en charge de certificats OpenPGP [72], qui diffèrent des certificats X.509 classiques. Ceux-ci permettent d'établir un lien cryptographique entre une adresse de messagerie, une identité et une clé publique. L'utilisation et la prise en charge de cette extension ne sont généralement pas nécessaires.

- `ec_point_formats` (0x000B) [57]

Cette extension signale les formats de point de courbe elliptique pris en charge par le client ou le serveur (s'il en existe). Il est en effet possible de représenter les points de courbe elliptique sous une forme compressée. En l'absence de cette extension, il est attendu que les coordonnées de points soient transmises dans leur totalité. L'usage de cette extension a été rendu obsolète par l'IETF, indiquant que seul le format non compressé devait être supporté [73].

38. Uniform Resource Locator.

- `client_certificate_type` (0x0013) [29]

Cette extension permet de signaler la capacité du client à transmettre, à l'intérieur du message `Certificate` qu'il envoie dans le cadre d'une authentification mutuelle, une clé publique brute plutôt qu'une liste de certificats X.509. La preuve d'appartenance de la clé publique est établie indépendamment de la connexion TLS. Cette extension n'est généralement pas nécessaire, l'utilisation d'une IGC restant préférable. Sa prise en charge est déconseillée.

- `server_certificate_type` (0x0014) [29]

Cette extension permet de signaler la capacité du serveur à transmettre, à l'intérieur du message `Certificate` qu'il envoie pour son authentification, une clé publique brute plutôt qu'une liste de certificats X.509. Pour les mêmes raisons que l'extension `client_certificate_type`, la prise en charge de cette extension est déconseillée.

- `early_data` (0x002A) [11]

Cette extension permet de l'envoi de données applicatives chiffrées à la suite du message `ClientHello` lorsqu'une PSK est partagée entre le client et le serveur. Elle permet de déclencher l'utilisation du mode 0-RTT qui offre un gain de performances. Cependant, cette amélioration des performances est réalisée au dépend de certaines propriétés de sécurité. Ces données applicatives ne disposent pas du même niveau de sécurité que les données applicatives envoyées à la suite du *handshake*. Pour cette raison, l'utilisation de cette extension est déconseillée.

Synthèse

Les extensions recommandées dans le cadre général de l'utilisation du protocole TLS figurent dans le tableau 2.4.

R14 Utiliser les extensions du tableau 2.4

Les extensions recommandées dans le cadre général de l'utilisation du protocole TLS figurent dans le tableau 2.4. Elles doivent être prises en charge par les équipements maîtrisés et utilisées dans les contextes précisés.

Code	Intitulé	Contexte d'utilisation
0x000A	<code>supported_groups</code>	Toujours recommandée
0x000D	<code>signature_algorithms</code>	Toujours recommandée
0x0012	<code>sct</code>	Si certificat EV
0x0016	<code>encrypt_then_mac</code>	Si version TLS 1.2
0x0017	<code>extended_master_secret</code>	Si version TLS 1.2
0x002B	<code>supported_versions</code>	Si version TLS 1.3
0x0032	<code>key_share</code>	Si version TLS 1.3
0xFF01	<code>renegotiation_info</code>	Toujours recommandée

Table 2.4 – Extensions TLS recommandées

Les tableaux 2.5, 2.6 et 2.7 précisent les valeurs recommandées pour les extensions `supported_groups`, `signature_algorithms` et `supported_versions`.

Code	Groupe
0x0017	secp256r1
0x0018	secp384r1
0x0019	secp521r1
0x001D	x25519
0x001E	x448
0x001A	brainpoolP256r1
0x001B	brainpoolP384r1
0x001C	brainpoolP512r1
0x001F	brainpoolP256r1tls13
0x0020	brainpoolP384r1tls13
0x0021	brainpoolP512r1tls13

Table 2.5 – Valeurs recommandées pour l’extension supported_groups

Code	Signature
0x0403	ecdsa_secp256r1_sha256
0x0503	ecdsa_secp384r1_sha384
0x0603	ecdsa_secp521r1_sha512
0x0807	ed25519
0x0808	ed448
0x0809	rsa_pss_pss_sha256
0x080a	rsa_pss_pss_sha384
0x080b	rsa_pss_pss_sha512
0x0804	rsa_pss_rsae_sha256
0x0805	rsa_pss_rsae_sha384
0x0806	rsa_pss_rsae_sha512
0x081A	ecdsa_brainpoolP256r1tls13_sha256
0x081B	ecdsa_brainpoolP384r1tls13_sha384
0x081C	ecdsa_brainpoolP512r1tls13_sha512

Table 2.6 – Valeurs recommandées pour l’extension signature_algorithms

Code	Version
0x0304	TLSv1.3
0x0303	TLSv1.2

Table 2.7 – Valeurs recommandées pour l’extension supported_versions

Des extensions supplémentaires d’usage acceptable dans certains contextes spécifiques figurent dans le tableau 2.8.

R15 Évaluer l'utilité des extensions du tableau 2.8

Les extensions du tableau 2.8 sont relatives à des déploiements spécifiques. Seules celles évaluées nécessaires doivent être implémentées et utilisées.

Les extensions les plus utiles sont `server_name`, `status_request_v2` (exclusivement pour TLS 1.2), `status_request` (pour compléter `status_request_v2` dans un cadre où la compatibilité avec des clients non maîtrisés est nécessaire) et `session_ticket`.

Code	Intitulé	Contexte d'utilisation
0x0000	<code>server_name</code>	Toujours recommandée
0x0003	<code>trusted_ca_keys</code>	En cas de racines alternatives (rare) (TLS 1.2)
0x0005	<code>status_request</code>	En cas d'agrafage OCSP
0x0006	<code>user_mapping</code>	En cas de clients indexés (rare) (TLS 1.2)
0x000C	<code>srp</code>	En cas de suites SRP (TLS 1.2)
0x000E	<code>use_srtp</code>	En cas de SRTP (rare)
0x0010	<code>alpn</code>	En cas de HTTP/2
0x0011	<code>status_request_v2</code>	En cas d'agrafage OCSP (TLS 1.2)
0x0015	<code>padding</code>	En cas de problème de compatibilité
0x001C	<code>record_size_limit</code>	En cas de contraintes réseau (rare)
0x0023	<code>session_ticket</code>	En cas de reprises de session (TLS 1.2)
0x0029	<code>pre_shared_key</code>	En cas de reprises de session (TLS 1.3)
0x002C	<code>cookie</code>	En cas de version TLS 1.3
0x002D	<code>psk_key_exchange_modes</code>	En cas de reprises de session (TLS 1.3)
0x002F	<code>certificate_authorities</code>	En cas de racines alternatives (rare) (TLS 1.3)
0x0030	<code>oid_filters</code>	En cas d'authentification client (TLS 1.3)
0x0031	<code>post_handshake_auth</code>	En cas d'authentification tardive client (TLS 1.3)
0x0032	<code>signature_algorithms_cert</code>	En cas d'algorithme spécifique dans le certificat

Table 2.8 – Extensions TLS relatives à un contexte spécifique

R16 Ne pas utiliser les extensions du tableau 2.9

Les extensions du tableau 2.9 sont toujours déconseillées.

2.4 Considérations additionnelles

Aléas

Lors de la négociation des paramètres d'une session TLS, le client et le serveur génèrent un aléa chacun. Ils sont partagés à l'aide des messages `ClientHello` et le `ServerHello`. En TLS 1.2,

Code	Intitulé
0x0001	max_fragment_length
0x0002	client_certificate_url
0x0004	truncated_hmac
0x0007	client_authz
0x0008	server_authz
0x0009	cert_type
0x000B	ec_point_formats
0x000F	heartbeat
0x0013	client_certificate_type
0x0014	server_certificate_type
0x002A	early_data

Table 2.9 – Extensions TLS déconseillées

ces valeurs interviennent à plusieurs reprises, en particulier lors de la signature de paramètres Diffie–Hellman, lors du calcul du *master secret*, et lors du calcul du condensat de l’ensemble des messages de la négociation échangé dans les messages **Finished**. L’utilisation de valeurs non prédictibles constitue une protection essentielle contre les attaques par rejeu.

La création de ces valeurs doit faire appel à un générateur d’aléa de qualité cryptographique, tel que défini par le RGS [35]. Celles-ci sont codées sur 32 octets. Les spécifications de TLS 1.2 séparent les 32 octets en 28 octets aléatoires et un préfixe de 4 octets correspondant à l’heure Unix à la génération du message. Cette construction visait à générer des aléas non immédiatement prédictibles quand bien même le générateur d’aléa utilisé aurait été compromis. Cependant, l’écriture de l’heure expose à un traçage indésirable, et l’utilisation d’un générateur d’aléa robuste reste indispensable. Par conséquent, lorsqu’elle est permise, la structure de 32 octets entièrement aléatoire est à préférer.

Les spécifications de TLS 1.3 précisent une construction légèrement différente de ces valeurs aléatoires permettant de fournir un mécanisme de protection contre les attaques par dégradation de version.

Ce mécanisme de protection consiste à insérer une valeur particulière dans les 8 derniers octets de l’aléa renvoyé par le serveur. Cette valeur étant signée, elle n’est donc pas modifiable par un attaquant essayant de forcer l’utilisation d’une version inférieure. Si le client implémente la vérification de l’aléa serveur, il est alors en mesure de détecter une tentative de dégradation de version et peut alors interrompre la session. Les spécifications de TLS 1.3 précisent une construction avec 32 octets aléatoires pour l’aléa utilisé dans le **ClientHello**.

R17 Utiliser un générateur d’aléa robuste

Les aléas utilisés dans le **ClientHello** et le **ServerHello** doivent provenir de générateurs d’aléa de qualité cryptographique.

L'activation de tels générateurs requiert généralement de positionner correctement ou de vérifier les options de compilation de l'implémentation de TLS employée.

R18 Privilégier l'aléa du serveur avec un suffixe prédictible

L'aléa utilisé dans le `ClientHello` doit privilégier une construction sur 32 octets aléatoires tandis que l'aléa utilisé dans le `ServerHello` doit privilégier une construction de 24 octets aléatoires avec, en suffixe les valeurs spécifiées dans [11].

R18– Privilégier les aléas sans préfixe prédictible

Les aléas utilisés dans le `ClientHello` et le `ServerHello` doivent privilégier les 32 octets aléatoires plutôt que la construction préfixée par une heure Unix.

Compression

Dans les versions du protocole antérieures à TLS 1.3, la phase de négociation détermine également un algorithme de compression en plus des paramètres pour la protection des records. Cet algorithme de compression est utilisé sur les données applicatives avant leur chiffrement. Le seul algorithme de compression de données proposé (autre que la fonction qui laisse inchangée les données) est l'algorithme Deflate [74].

Cependant, l'utilisation de la compression combinée à l'absence de protection sur la longueur des messages chiffrés peut résulter en la compromission d'une partie des données échangées, et notamment des cookies de session que certains serveurs utilisent pour identifier leurs clients [75]. L'utilisation de ces cookies par un tiers s'assimile à une usurpation d'identité, ce qui est dangereux pour le client initial à moins que le serveur ne propose que du contenu statique, identique pour tous ses clients.

Cette vulnérabilité est contrée par la plupart des navigateurs web, qui ont soit désactivé le mécanisme de compression, soit implémenté un palliatif tel que la non compression de l'entête contenant le cookie sensible. Cependant, cette approche ne couvre pas tous les cas d'usage applicatifs, ni tous les clients susceptibles d'exploiter le protocole TLS. Par conséquent, la compression (autre que la transformation identité établie par défaut) reste à éviter. La compression a été totalement supprimée dans TLS 1.3. Pour ces raisons, la compression TLS est déconseillée.

R19 Ne pas utiliser la compression TLS

L'utilisation du mécanisme de compression TLS est à proscrire.

Reprise de session

L'ensemble des versions de TLS dispose de méthodes de reprise de session permettant d'écourter la phase de négociation en restaurant des secrets précédemment établis.



Dans les versions antérieures à TLS 1.3, il existe deux méthodes de reprise de session. La première méthode repose sur l'identifiant de session contenu dans le `ClientHello`, associé par le client à un ensemble de paramètres et de secrets précédemment mis en cache. Dans cette situation, si le serveur a lui-même mis en cache les paramètres de la session caractérisée par l'identifiant transmis, alors il peut choisir de les restaurer et la phase d'échange de clé n'a pas besoin d'être reproduite. Si le serveur ne reconnaît pas l'identifiant de session, la négociation se poursuit de façon standard.

La seconde méthode s'appuie sur les tickets de session et l'extension correspondante, brièvement décrite en section 2.3. L'utilisation de tickets de session est préférable à celle des identifiants de session, car elle n'exige pas du serveur de conserver les paramètres relatifs à une session. Par ailleurs, leur protection par des moyens cryptographiques est exigée par les spécifications de l'extension [68].

Cependant, quels que soient les moyens de protection employés, la conservation des paramètres induit un risque vis-à-vis de la propriété de confidentialité persistante. De plus, même si d'autres moyens de prévention existent, il est à noter que l'attaque *Triple Handshake* [76] est immédiatement contrée par l'absence de prise en charge du mécanisme de reprise de session.

TLS 1.3 propose une méthode de reprise de session basée sur l'utilisation de tickets et de PSK établis lors d'une session initiale. Le ticket est émis par le serveur après la fin du *handshake* lors de la session initiale.

Pour reprendre une session, le client rejoue alors le ticket reçu dans le champ `identity` de l'extension `pre_shared_key` du `ClientHello`. Il précise en complément le mode utilisé via l'extension `psk_key_exchange_mode`. Afin de garantir la propriété de confidentialité persistante, il est recommandé de préciser le mode `psk_dhe_ke` permettant de réaliser un nouvel échange de clé Diffie–Hellman lors de la reprise de session.

La reprise de session ne renouvelle pas la phase d'authentification. Il n'est donc pas recommandé d'émettre un nouveau ticket dans une session reprise car cela permettrait d'étendre la période d'authentification initiale et éventuellement de dépasser la date de validité du certificat dans le cadre d'une authentification par certificat.

Lors d'une reprise de session, le client peut également transmettre des données applicatives à la suite du `ClientHello`. Cette fonctionnalité est décrite plus en détails à la fin de cette section. Pour indiquer au client qu'il ne souhaite pas utiliser cette fonctionnalité, le serveur peut spécifier, dans le ticket, une extension `early_data` dont le champ `max_early_data_size` vaut 0.

Si ces mécanismes sont utilisés, la durée de conservation des informations mises en cache pour les identifiants, ainsi que celle des tickets, doit être réduite en fonction du niveau de sécurité souhaité. Une purge quotidienne des caches constitue un compromis acceptable.

R20 Limiter la durée de vie des tickets

Le serveur doit être configuré afin d'émettre des tickets dont la durée de vie n'excède pas 24 heures.

R21 Effectuer des reprises de session avec échange de clé

L'utilisation du mécanisme de reprise de session avec un échange de clé Diffie–Hellman est recommandée.

La recommandation **R21** peut être mise en œuvre uniquement avec la version TLS 1.3. La recommandation alternative **R21-** s'appliquent pour la version TLS 1.2.

R21 – Effectuer des reprises de session avec PFS

Si le mécanisme de reprise de session est utilisé, alors l'utilisation de tickets de session doit être préférée à celle d'identifiants de session. Dans le cas des tickets de session, les tickets doivent être supprimés à intervalles réduits, et les clés de chiffrement doivent être supprimées et régénérées régulièrement. Dans le cas des identifiants de session, les données mises en cache doivent être supprimées de part et d'autre à intervalles réduits.

Renégociation

Les versions du protocole antérieures à TLS 1.3 disposent d'un mécanisme de renégociation. Ce mécanisme de renégociation expose le client à une vulnérabilité protocolaire. Il est en effet possible pour un attaquant de faire passer la première négociation d'un client pour une renégociation. L'attaquant se retrouve ainsi en position d'injecter des données applicatives que le serveur attribue au client légitime. Du point de vue du serveur, ces données précèdent de façon transparente les données applicatives ensuite envoyées par le client légitime. Ce procédé peut servir d'amorce à des scénarios d'attaque plus complexes [24, 77].

Si un attaquant effectue une première négociation auprès d'un serveur qui prend en charge les renégociations sécurisées, puis fait passer le `ClientHello` d'un client légitime pour une renégociation, alors la présence de l'extension `renegotiation_info` dans ce `ClientHello` sans qu'elle soit accompagnée du contenu du `Finished` (qui du point de vue du client légitime, n'a jamais eu lieu) permet au serveur de détecter l'attaque et de terminer la session. Il est à noter que l'utilisation de l'extension reste nécessaire pour le client même s'il ne souhaite jamais effectuer de renégociation.

R22 Toujours activer l'extension pour la renégociation sécurisée

Un client TLS, qu'il souhaite ou non effectuer des renégociations, doit utiliser l'extension `renegotiation_info`. Un serveur TLS, s'il souhaite effectuer des renégociations, doit utiliser l'extension `renegotiation_info`.

TLS 1.3 ne supporte pas de mécanisme de renégociation. Un mécanisme de renouvellement de clé est présent afin de remplacer la renégociation. Le renouvellement est réalisé via les messages `KeyUpdate` et ne nécessite pas un nouvel échange de clé asymétrique. La fonction de dérivation `HKDF-Expand` est utilisée pour calculer la nouvelle clé de chiffrement.

O-RTT

Une nouvelle fonctionnalité apparue avec TLS 1.3 consiste à transmettre des données applicatives chiffrées dès le premier échange. Ces données applicatives transmises avant la fin du *handshake* sont appelées données O-RTT³⁹.

Afin d'utiliser ce mécanisme, le client et le serveur doivent posséder une PSK. Avec TLS 1.3, cette PSK peut être établie lors d'une session précédente dans le cas d'une reprise de session ou alors établie de manière indépendante du protocole.

Les données O-RTT ne disposent cependant pas du même niveau de protection que les données applicatives transmises à la suite du *handshake*. En particulier, ces données ne disposent pas de protection anti-rejeu et sont donc vulnérables à un attaquant jouant des messages interceptés. Pour cette raison, l'envoi de données O-RTT n'est pas recommandée.

R23 Ne pas transmettre de données O-RTT

Il est déconseillé à un client de transmettre de données O-RTT. Un serveur ne doit pas accepter les données O-RTT lorsqu'il en reçoit.

39. Zero Round-Trip Time.

Chapitre 3

Mise en place de l'IGC

Ce chapitre rassemble les recommandations relatives aux attributs des certificats X.509 utilisés lors du *handshake*. Il discute également des différentes méthodes de révocation existantes. Les méthodes de génération et de conservation de clés, et d'obtention ou de bascule entre certificats, dépassent le cadre de ce document ; elles sont traitées dans l'annexe B2 du RGS [78].

3.1 Attributs des certificats X.509

Un certificat X.509 est composé de plusieurs champs de base, généralement accompagnés d'extensions. Celles-ci précisent les contextes d'utilisation du certificat et fournissent des moyens de valider plus précisément le certificat au sein de la chaîne de certification. Ce chapitre décrit quelques-uns des champs et extensions importants dans le cas de certificats d'authentification TLS, et émet des recommandations quant à leur contenu. Des recommandations sur les autres champs pourront être trouvées dans l'annexe A4 du RGS [20]. Des informations complémentaires sur leur utilisation et leur structure figurent dans la RFC 5280 [18].

3.1.1 Champs de base

Le numéro de série d'un certificat (*serial number*) identifie le certificat au sein de l'autorité de certification qui l'a émis. Ce numéro de série est un nombre positif dont la taille ne dépasse pas 20 octets. Il doit être unique parmi tous les certificats générés par une même autorité de certification. Le RGS recommande par ailleurs aux ACs de générer des certificats avec des numéros de série non prédictibles, afin de prévenir les attaques par collision qui pourraient porter sur la signature.

Le champ *signature* indique les algorithmes employés par l'autorité de certification pour signer les informations du certificat. Il comprend la fonction de hachage utilisée pour calculer l'empreinte des données, ainsi que l'algorithme de signature qui y est ensuite appliqué. Le suivi de l'annexe B1 du RGS [35] est recommandé à leur sujet.

R24 Présenter un certificat signé avec SHA-2

La fonction de hachage utilisée pour la signature du certificat doit faire partie de la famille SHA-2.

Un certificat possède deux champs indiquant la période de validité du certificat :

- le champ *notBefore* indique la date à partir de laquelle le certificat est valide ;
- le champ *notAfter* indique la date après laquelle le certificat n'est plus valide.

Pour un certificat émis après le 1^{er} mars 2018, il est recommandé que la période de validité d'un certificat ne dépasse pas 825 jours [79]. Dans le cas général, la période de validité d'un certificat ne doit pas dépasser 3 ans.

R25 Présenter un certificat valide pendant 3 ans ou moins

La période de validité d'un certificat d'authentification TLS (serveur ou client) ne doit pas excéder 3 ans. Pour un certificat émis après le 1^{er} mars 2018, sa période de validité ne doit pas dépasser 825 jours.

Il est parfois possible de renouveler un certificat auprès d'une autorité de certification tout en conservant la même paire de clés asymétriques. Cependant, de la même façon qu'un certificat, une paire de clés ne doit pas être valide plus de 3 ans.

Les informations sur la clé publique du sujet du certificat (*Subject Public Key Info*) sont structurées en deux parties, l'identifiant de l'algorithme qui sera mis en œuvre avec cette clé, ainsi que la clé publique elle-même. L'annexe B1 du RGS fournit plusieurs recommandations au sujet des algorithmes et des tailles de clé à respecter.

R26 Utiliser des clés de taille suffisante

Pour une protection des communications jusqu'en 2030, les clés RSA doivent avoir une taille minimale de 2048 bits, et les clés ECDSA doivent avoir une taille minimale de 256 bits. Pour ECDSA, les courbes éprouvées retenues sont `secp256r1`, `secp384r1`, `secp521r1`, ainsi que `brainpoolP256r1`, `brainpoolP384r1` et `brainpoolP512r1`. Pour RSA, l'exposant de la clé publique doit être supérieur ou égal à $2^{16} + 1$.

3.1.2 Extensions

Depuis la version 3 du standard X.509, des extensions peuvent être rajoutées aux certificats. Celles-ci servent principalement à :

- identifier le possesseur de la paire de clés asymétriques à laquelle se réfère le certificat (*Subject Alternative Name*) ;
- préciser les usages possibles du certificat (*Key Usage*, *Extended Key Usage*, *Basic Constraints*, *Subject Alternative Name*) ;
- fournir des informations servant à vérifier l'état de révocation du certificat (*CRL Distribution Points*, *Freshest CRL*, *Authority Information Access*) ;
- fournir des informations pour valider le certificat au sein de la chaîne de certification (*Authority Key Identifier*, *Subject Key Identifier*) ;
- fournir un lien vers la politique de certification qui s'applique au certificat (*Certificate Policies*).

Il existe un attribut permettant de marquer les extensions considérées critiques. Si une application ne reconnaît pas une extension d'un certificat marquée comme critique, alors elle doit rejeter ce certificat.

L'extension *Key Usage* définit les opérations qu'il sera possible de réaliser avec la clé associée au certificat. Cette extension contient une ou plusieurs valeurs :

- la valeur *digitalSignature* indique que la clé publique pourra servir à vérifier des signatures électroniques, autres que celles des certificats et des CRL ;
- la valeur *keyEncipherment* indique que la clé publique pourra servir à chiffrer des clés de session. Il s'agit alors d'un échange sans PFS ;
- la valeur *keyAgreement* signale une clé utile à un protocole d'échange de clés. Le cas naturel correspond à une clé Diffie–Hellman statique, mais ces certificats sont rares ; comme expliqué en section 2.2, il faut leur préférer des échanges Diffie–Hellman basés sur des clés éphémères, c'est-à-dire générées à chaque nouvelle session ;
- la valeur *keyCertSign* indique un certificat appartenant à une AC racine ou intermédiaire, dont la clé a été utilisée pour signer d'autres certificats ;
- la valeur *crlSign* indique aussi un certificat appartenant à une AC racine ou intermédiaire, et dont la clé a été utilisée afin de signer des listes de révocation.

R27 Présenter un *KeyUsage* approprié

Dans un certificat d'authentification, l'extension *Key Usage* doit être présente et marquée comme critique. Pour un serveur, elle doit contenir les valeurs *digitalSignature* et/ou *keyEncipherment*. Pour un client, elle doit contenir la valeur *digitalSignature*. Aucune autre valeur n'est admise.

L'extension *Extended Key Usage* indique un usage plus précis du certificat et complète l'extension *Key Usage*. En-dehors de *id-kp-OCSPSigning* utilisée pour caractériser les certificats d'AC dont la clé est utilisée pour signer des réponses OCSP, les autres valeurs définies pour *Extended Key Usage* qui interviennent dans le cadre de TLS sont *id-kp-serverAuth* et *id-kp-clientAuth*.

R28 Présenter un *ExtendedKeyUsage* approprié

Dans un certificat d'authentification, l'extension *Extended Key Usage* doit être présente et marquée comme non-critique. Pour un serveur, elle doit uniquement contenir la valeur *id-kp-serverAuth*. Pour un client, elle doit uniquement contenir la valeur *id-kp-clientAuth*.

L'extension SAN⁴⁰ permet d'indiquer d'autres identités du sujet du certificat que celle présente dans le champ *Subject*, telles que des noms DNS complets (*dNSName*) ou des adresses de messagerie électronique (*rfc822Name*). Ainsi, à titre d'exemple, un certificat de serveur web devrait contenir le nom de domaine consulté par l'utilisateur dans la partie *Common Name* du champ *subject*, ou bien parmi les FQDN⁴¹ des entrées *dNSName* de l'extension SAN.

40. Subject Alternative Name.

41. Fully Qualified Domain Name.

R29 Présenter un *SubjectAlternativeName* approprié (côté serveur)

Pour un certificat d'authentification utilisé par un serveur TLS, l'extension *Subject Alternative Name* doit être présente et marquée comme non-critique. Elle doit contenir au moins une entrée *dnsName* correspondant à l'un des FQDN du service applicatif qui utilise le certificat.

L'extension SAN est parfois utilisée pour associer plusieurs services à un même certificat. Il est cependant déconseillé d'utiliser une même clé privée, et donc un même certificat, pour des services TLS distincts. En effet, cette pratique engendre la mutualisation des risques portant, de façon individuelle, sur chacune des terminaisons TLS. En particulier, la duplication, la distribution et l'existence en plusieurs endroits de la clé privée, afin qu'elle serve à plusieurs services TLS, augmente le risque qu'elle soit compromise. En revanche, l'utilisation d'un proxy TLS placé en frontal devant différents services applicatifs reste acceptable, dans la mesure où la terminaison TLS est unique.

R30 Réserver chaque certificat à une seule terminaison TLS

Un même certificat d'authentification ne doit pas être utilisé par plus d'une seule terminaison TLS.

Le raisonnement précédent se prolonge à la différenciation des certificats selon les paramètres retenus pour la négociation. En effet, certaines attaques [5] ont montré que la tolérance envers une version du protocole pouvait compromettre les sessions menées avec d'autres versions, si celles-ci s'appuyaient sur un même certificat.

R31

Pour une même terminaison, il est recommandé d'utiliser autant de certificats que de versions et de méthodes d'échange de clés acceptées.

L'extension AKI⁴² permet d'identifier la clé de l'AC qui a signé le certificat, notamment lorsque l'AC en question possède plusieurs clés. La norme X.509 permet d'utiliser l'identifiant présent dans l'extension SKI⁴³ du certificat de l'AC relatif à la clé de signature, ou bien l'association du nom de l'AC et du numéro de série du certificat relatif à la clé de signature. Seule la première des deux options est conforme au RGS.

42. Authority Key Identifier.

43. Subject Key Identifier.

R32 Présenter un AKI correspondant au SKI défini par l'AC

Pour un certificat d'authentification TLS (serveur ou client), l'extension AKI doit être présente, marquée comme non-critique et contenir l'identifiant présent dans l'extension SKI du certificat de l'AC relatif à la clé de signature utilisée.

Les extensions CRLDP et AIA indiquent des chemins d'accès aux informations de révocation du certificat. L'extension CRLDP peut contenir une ou plusieurs URL (HTTP, FTP, LDAP) pointant vers un fichier de CRL. L'extension AIA peut contenir une ou plusieurs URL correspondant à un répondeur OCSP. Un certificat doit contenir au moins un moyen de vérifier son état de révocation.

R33 Présenter un certificat avec des sources de révocation

Au moins une extension parmi CRLDP et AIA doit être présente et marquée comme non-critique.

3.2 Contrôle de validité

Chaînes de certificats

En règle générale, le message `Certificate` envoyé par le serveur contient plusieurs certificats : celui qui lie l'identité du serveur applicatif avec la paire de clés asymétriques utilisée pour l'échange de clés du *handshake*, mais aussi ceux qui permettent, par le jeu des signatures cryptographiques, d'établir un lien de confiance depuis le certificat terminal jusqu'à une autorité de certification reconnue.

Afin de permettre aux clients d'interpréter cette chaîne de confiance sans ambiguïté, il est nécessaire de la transmettre de façon ordonnée par relation de signature et dans son intégralité, depuis le certificat terminal jusqu'au certificat intermédiaire signé par la racine de confiance : chaque certificat est suivi d'un certificat qui le signe. Il est inutile de transmettre le certificat racine qui permet de vérifier cette ultime signature, dans la mesure où le client est déjà censé en disposer localement.

Même si TLS 1.3 a levé la contrainte de l'ordre de la chaîne de certificats, et afin de s'assurer une compatibilité avec TLS 1.2, il est important de conserver l'ordre de ces derniers pour éviter tout risque de dysfonctionnement.

R34 Transmettre une chaîne de certificats ordonnée et complète

Les chaînes de certificats transmises à l'aide des messages `Certificate` doivent, afin de conserver une compatibilité avec TLS 1.2, être ordonnées et complètes.

Mécanismes de révocation

La mise en place de mécanismes de révocation est critique pour conserver la confiance dans les certificats. Les méthodes existantes principales sont présentées en section 1.2. Au moins un de ces mécanismes doit être mis en œuvre, et l'URL du fichier CRL ou du répondeur OCSP doit être présente dans les extensions des certificats.

Le choix parmi ces mécanismes doit reposer sur l'analyse des besoins en sécurité et en disponibilité, des contraintes de débit et de temps ainsi que sur l'architecture du système d'information. Un répondeur OCSP fournit des informations de révocation plus récentes que les CRL téléchargées de façon asynchrone, mais il doit être joignable en permanence. Par ailleurs, la taille de certaines CRL peut constituer un obstacle à leur déploiement. Dans un scénario de révocation massive, cette contrainte en performance affecte davantage les AC responsables de la distribution des CRL, qui font face à des risques de déni de service [80].

Comme décrit en section 1.2, l'agrafage OCSP apporte des améliorations au protocole OCSP standard :

- en performance : le répondeur OCSP n'est pas interrogé à chaque validation du certificat car le serveur TLS maintient en cache des réponses OCSP ;
- en protection de la vie privée : le répondeur OCSP ne connaît plus les clients qui se connectent aux services pour lesquels le certificat a été émis.

R35 Préférer l'agrafage OCSP

Lorsque le protocole OCSP est mis en œuvre, il est recommandé d'utiliser l'agrafage OCSP. Cette solution est aussi préférable à la distribution de CRL.

La plupart des outils validant les certificats tentent de contacter les autres URL éventuellement présentes dans l'extension CRLDP (ou AIA) si la première ne répond pas. Il est donc judicieux de mettre en place des mécanismes de redondance de publication des informations de révocation. Dans le cas des CRL, le fichier CRL pourra être hébergé sur plusieurs serveurs web (ou annuaires) différents. Dans le cas d'OCSP, plusieurs serveurs OCSP seront mis en œuvre, partageant les mêmes informations de révocation.

R36 Prévoir une redondance des moyens de révocation

Pour des raisons de disponibilité, des mécanismes de redondance de publication des informations de révocation doivent être mis en œuvre.

Comportement en l'absence d'informations de révocation

Les services de vérification des informations de révocation, lorsqu'ils sont accessibles, permettent catégoriquement de valider ou de rejeter un certificat. Si aucun d'entre eux n'est joignable, deux comportements sont définis : le comportement *hard-fail* consiste à refuser la connexion, tandis que le comportement *soft-fail* consiste à accepter tout de même le certificat et continuer l'échange.

Le choix entre ces deux comportements peut dépendre du contexte d'utilisation et de la criticité de l'application. Une application qui privilégie la disponibilité sera configurée pour le *soft-fail*, tandis qu'une autre où le besoin de sécurité est prédominant activera le *hard-fail*. Si le *soft-fail* est activé et que le répondeur OCSP (ou le serveur délivrant les CRL) est rendu indisponible, alors un attaquant possédant la clé associée à un certificat révoqué sera tout de même en mesure d'usurper l'identité du sujet du certificat.

R37 Réagir en *hard-fail*

Les composants logiciels TLS privilégiant la sécurité doivent réagir en *hard-fail*.

Certificate Transparency

Dans le cadre d'une IGC publique, le programme CT présenté en section 1.2 peut apporter une assurance supplémentaire quant à la validité d'un certificat transmis par un serveur. À la rédaction de ce document, seuls les certificats *Extended Validation* (EV) sont automatiquement vérifiés par les clients TLS compatibles avec CT. Néanmoins, certaines AC comme Symantec ou Let's Encrypt enregistrent tous leurs nouveaux certificats dans un *Certificate Log*.

R38 Utiliser des certificats enregistrés par CT

Dans le cadre d'une IGC publique, il est recommandé d'utiliser des certificats d'authentification enregistrés par leur AC parmi les registres du programme CT.

R39 Rejeter tous les certificats invalidés par CT

Dans le cadre d'une IGC publique, les clients TLS doivent rejeter tous les certificats qui sont accompagnés d'un *Signed Certificate Timestamp* (SCT) invalide, et les certificats EV qui ne sont pas accompagnés d'un SCT.

Annexe A

Référentiel des suites cryptographiques

Le référentiel suivant synthétise les recommandations de la section 2.2 selon plusieurs tableaux de suites cryptographiques.

A.1 Suites recommandées

Les suites cryptographiques offrant un niveau de sécurité conforme à l'état de l'art sont données dans les tableaux A.1, A.2 et A.3.

Le tableau A.1 liste les suites cryptographiques recommandées dans le cas général pour un usage avec TLS 1.3. Pour rappel, l'utilisation de l'algorithme ECDHE est recommandée pour établir la clé de session.

Code TLS	Suite cryptographique
0x1302	TLS_AES_256_GCM_SHA384
0x1301	TLS_AES_128_GCM_SHA256
0x1304	TLS_AES_128_CCM_SHA256
0x1303	TLS_CHACHA20_POLY1305_SHA256

Table A.1 – Suites recommandées définies pour TLS 1.3

Le tableau A.2 liste les suites cryptographiques recommandées pour un usage avec TLS 1.2 lorsque le serveur d'une clé ECDSA.

Code TLS	Suite cryptographique
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
0xCCA9	TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256

Table A.2 – Suites TLS 1.2 recommandées avec un serveur disposant d'un certificat avec clé publique ECDSA

Le A.3 liste également les suites cryptographiques recommandées avec TLS 1.2 lorsque le serveur dispose d'une clé RSA.

Code TLS	Suite cryptographique
0xC030	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
0xC02F	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
0xCCA8	TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256

Table A.3 – Suites TLS 1.2 recommandées avec un serveur disposant d'un certificat avec clé publique RSA

A.2 Suites dégradées

Lorsqu'une des deux parties communicantes n'est pas maîtrisée, il n'est pas toujours possible de négocier une session TLS avec une des suites cryptographiques précédentes. Dans le cas où un besoin fort de compatibilité a été identifié, d'autres suites peuvent être adoptées. Comme explicité dans la section 2.2, cet élargissement des possibilités de négociation se fait généralement au détriment de la sécurité des communications. En conséquence, il convient d'évaluer le profil des serveurs ou bien des clients visés, et de n'adopter que les suites jugées indispensables à l'accomplissement des fonctions applicatives considérées.

Le tableau A.4 liste les suites cryptographiques recommandées pour une usage général avec TLS 1.2 en cas d'absence de prise en charge de ECC ou de mode de chiffrement authentifié.

Code TLS	Suite cryptographique
0x009F	TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
0x009E	TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
0xCCAA	TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256
0xC09E	TLS_DHE_RSA_WITH_AES_128_CCM
0x0067	TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
0x006B	TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
0xC024	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
0xC023	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
0xC028	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
0xC027	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256

Table A.4 – Suites dégradées définies pour TLS 1.2

Pour rappel, l'utilisation des suites exploitant le mode de chiffrement CBC est recommandée en conjonction avec l'extension `encrypt_then_mac`.

Bien que l'usage du chiffrement AES⁴⁴, plus éprouvé, soit à privilégier, Camellia et ARIA ne font à ce jour l'objet d'aucune attaque connue, et constituent des alternatives acceptables. Les suites correspondantes figurent dans les tableaux A.5 et A.6.

44. Advanced Encryption Standard.

Code TLS	Suite cryptographique
0xC08B	TLS_ECDHE_RSA_WITH_CAMELLIA_256_GCM_SHA384
0xC08A	TLS_ECDHE_RSA_WITH_CAMELLIA_128_GCM_SHA256
0xC087	TLS_ECDHE_ECDSA_WITH_CAMELLIA_256_GCM_SHA384
0xC086	TLS_ECDHE_ECDSA_WITH_CAMELLIA_128_GCM_SHA256
0xC07D	TLS_DHE_RSA_WITH_CAMELLIA_256_GCM_SHA384
0xC07C	TLS_DHE_RSA_WITH_CAMELLIA_128_GCM_SHA256
0xC073	TLS_ECDHE_ECDSA_WITH_CAMELLIA_256_CBC_SHA384
0xC072	TLS_ECDHE_ECDSA_WITH_CAMELLIA_128_CBC_SHA256
0xC077	TLS_ECDHE_RSA_WITH_CAMELLIA_256_CBC_SHA384
0xC076	TLS_ECDHE_RSA_WITH_CAMELLIA_128_CBC_SHA256
0x00C4	TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA256
0x00BE	TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA256

Table A.5 – Suites dégradées pour l’usage de Camellia

Code TLS	Suite cryptographique
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
0xC053	TLS_DHE_RSA_WITH_ARIA_256_GCM_SHA384
0xC052	TLS_DHE_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
0xC045	TLS_DHE_RSA_WITH_ARIA_256_CBC_SHA384
0xC044	TLS_DHE_RSA_WITH_ARIA_128_CBC_SHA256

Table A.6 – Suites dégradées pour l’usage d’ARIA

Enfin, dans le cadre des déploiements avec *pre-shared keys*, les suites recommandées figurent dans le tableau A.7.

Code TLS	Suite cryptographique
0xD002	TLS_ECDHE_PSK_WITH_AES_256_GCM_SHA384
0xD001	TLS_ECDHE_PSK_WITH_AES_128_GCM_SHA256
0xD005	TLS_ECDHE_PSK_WITH_AES_128_CCM_SHA256
0x00AB	TLS_DHE_PSK_WITH_AES_256_GCM_SHA384
0x00AA	TLS_DHE_PSK_WITH_AES_128_GCM_SHA256
0xCCAD	TLS_DHE_PSK_WITH_CHACHA20_POLY1305_SHA256
0xC0A7	TLS_DHE_PSK_WITH_AES_256_CCM
0xC0A6	TLS_DHE_PSK_WITH_AES_128_CCM
0xC038	TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA384
0xC037	TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256
0x00B3	TLS_DHE_PSK_WITH_AES_256_CBC_SHA384
0x00B2	TLS_DHE_PSK_WITH_AES_128_CBC_SHA256
0xC091	TLS_DHE_PSK_WITH_CAMELLIA_256_GCM_SHA384
0xC090	TLS_DHE_PSK_WITH_CAMELLIA_128_GCM_SHA256
0xC09B	TLS_ECDHE_PSK_WITH_CAMELLIA_256_CBC_SHA384
0xC09A	TLS_ECDHE_PSK_WITH_CAMELLIA_128_CBC_SHA256
0xC097	TLS_DHE_PSK_WITH_CAMELLIA_256_CBC_SHA384
0xC096	TLS_DHE_PSK_WITH_CAMELLIA_128_CBC_SHA256
0xC06D	TLS_DHE_PSK_WITH_ARIA_256_GCM_SHA384
0xC06C	TLS_DHE_PSK_WITH_ARIA_128_GCM_SHA256
0xC071	TLS_ECDHE_PSK_WITH_ARIA_256_CBC_SHA384
0xC070	TLS_ECDHE_PSK_WITH_ARIA_128_CBC_SHA256
0xC067	TLS_DHE_PSK_WITH_ARIA_256_CBC_SHA384
0xC066	TLS_DHE_PSK_WITH_ARIA_128_CBC_SHA256

Table A.7 – Suites dégradées pour l’usage de PSK

Annexe B

Exemples d'application des recommandations

L'annexe suivante présente des applications à plusieurs logiciels qui exploitent le protocole TLS, en s'approchant autant que possible des recommandations du guide compte tenu de leurs options respectives. Elle se décompose en deux parties :

- la première partie traite de la génération de la bibliothèque OpenSSL. Il y figure un ensemble de directives de compilation pour l'usage de cette bibliothèque dans un produit ou service donné ;
- la seconde partie se focalise sur la configuration de modules applicatifs courants, `mod_ssl` pour Apache et `ngx_http_ssl_module` pour NGINX. Elle illustre les directives qu'il convient d'utiliser pour monter un contexte TLS (côté serveur).

Application à la compilation de OpenSSL

La branche OpenSSL choisie est la 1.1.1. Il s'agit d'une version LTS⁴⁵ implémentant TLS 1.3 et dont la maintenance sera assurée jusqu'au 11 septembre 2023.

Le système de build OpenSSL repose sur un ensemble de scripts shell, de programmes Perl et de Makefiles configurés à partir de `./config`. L'extrait B.1 illustre la désactivation des algorithmes et des options non souhaitables, avant de lancer la compilation.

La compilation aboutit ici à deux bibliothèques d'intérêt :

- `libcrypto.so`, pour les algorithmes cryptographiques ;
- `libssl.so`, pour le protocole TLS en lui-même.

45. Long Term Support.

Extrait B.1 – Exemple de compilation OpenSSL 1.1.1

```
# Pensez à télécharger la dernière version stable de la branche 1.1.1
# et à vérifier son empreinte récupérée de manière intègre (via une page HTTPS)
$ tar xvf openssl-1.1.1a.tar.gz
$ cd openssl-1.1.1a
$ ./config shared -D_FORTIFY_SOURCE=2 -fstack-protector-all \
    -no-ssl3 -no-ssl3-method -no-tls1 -no-tls1-method -no-tls1_1 -no-tls1_1-
    method
    -no-ec2m \
    -no-weak-ssl-ciphers \
    -no-seed -no-idea -no-des \
    -no-mdc2 -no-md2 -no-md4 -no-whirlpool \
    -no-rc2 -no-rc4 -no-rc5 -no-cast \
    -no-heartbeats \
    -no-srp -no-psk -no-comp \
    -no-dsa -no-deprecated -no-gost
$ make depend
$ make
$ ls lib*.so
libcrypto.so libssl.so
```

Il faut noter que la branche 1.1.1 d'OpenSSL implémente par défaut certaines recommandations émises dans ce guide. C'est le cas par exemple des recommandations **R3**, **R18**, ou encore **R21**.

Application à la configuration de modules applicatifs pour Apache et NGINX

La majorité des applications disposant d'une couche TLS permettent de la configurer au travers de directives qui contrôlent sommairement les options et les extensions relatives au protocole. Il n'est pas strictement nécessaire d'utiliser la bibliothèque générée précédemment pour que ces directives soient respectées. Elles sont cependant très dépendantes de la bibliothèque TLS utilisée ainsi que de sa version.

Nous fournissons ici deux exemples de configuration de modules TLS liés à OpenSSL : `mod_ssl` pour Apache en branche 2.4, et `ngx_http_ssl_module` pour NGINX en branche 1.15. Il est à noter que, en plus des directives de configuration qu'ils fournissent, ces modules peuvent décider d'appliquer eux-mêmes des paramétrages supplémentaires à la couche TLS sans que l'intégrateur ait moyen d'agir dessus, sauf à modifier le code du module directement.

Les extraits de configuration proposés ici peuvent être appliqués à des hôtes virtuels distincts. Ils se concentrent sur les paramètres directement liés à TLS, et doivent être complétés à l'aide de la note technique portant sur la sécurisation des sites web [1].



Apache

La configuration B.2 suppose l'usage d'un serveur Apache en version 2.4.8 ou ultérieure. Elle propose des suites recommandées ainsi que certaines suites dégradées. Plusieurs directives ne sont pas prises en charge par les versions antérieures [81].

Extrait B.2 – Exemple de configuration SSL Apache 2.4.8+

```
# Activer l'usage du protocole TLS
SSLEngine on
# Forcer l'usage de TLS1.3 ou TLS1.2
SSLProtocol +TLSv1.3 +TLSv1.2
# Fournir une chaîne de certification et une clé
SSLCertificateKeyFile "/chemin/vers/cle-privée.pem"
SSLCertificateFile "/chemin/vers/chaîne-certification.pem"
# Désactiver la compression
SSLCompression off
# Désactiver les renégociations non sécurisées
SSLInsecureRenegotiation off
# Activer l'agraphe OCSP
SSLUseStapling on
SSLStaplingCache shmcb:logs/ssl_stapling(32768)
# Utiliser les courbes recommandées pour ECDHE et supported_groups
SSLOpenSSLConfCmd ECDHParameters Automatic
SSLOpenSSLConfCmd Curves secp521r1:secp384r1:prime256v1:X25519:X448:
    brainpoolP512r1:brainpoolP384r1:brainpoolP256r1

# Utiliser le groupe de taille 2048 pour DH défini dans RFC7919
# echo "-----BEGIN DH PARAMETERS-----
# MIIBCAKCAQEA/////////+t+FRYortKmq/cViAnPTzx2LnFg84tNpWp4TZBFGQz
# +8yTnc4kmz75fS/jY2MMddj2gbICrsRhetPfHtXV/WVhJDP1H18GbtCFY2VVPe0a
# 87VXE15/V8k1mE8McODmi3fipona8+/och3xWKE2rec1MKzKT0g6eXq8CrGCsyT7
# YdEIqUuyyOP7uWrat2DX9GgdTOKj3jLN9K5W7edjcrsZCweny04KbXCeAvzhzffi
# 7MAOBMOoNC9hkXL+nOmFg/+OTxIy7vKBg8P+OxtMb61z07X8vC7CIAXFjuGDfRaD
# ssbzSibBsu/6iGtCOGEoXJf/////////wIBAg==
# -----END DH PARAMETERS-----" > /chemin/vers/fichier/ffdhe2048.pem
#
SSLOpenSSLConfCmd DHParameters "/chemin/vers/fichier/ffdhe2048.pem"

SSLHonorCipherOrder on
# La branche 1.1.1 de OpenSSL :
# - ne prend pas en charge les suites CCM pour TLS 1.3;
# Relations d'ordre utilisées :
# - ECDHE > DHE ;
# - AES256 > AES128 > CHACHA20 ;
# - SHA384 > SHA256 ;
# - ECDSA > RSA.
#
TLS_AES_256_GCM_SHA384:TLS_AES_128_GCM_SHA256:TLS_CHACHA20_POLY1305_SHA256:
    ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-
    AES256-GCM-SHA384:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-CHACHA20-
    POLY1305:ECDHE-RSA-CHACHA20-POLY1305:DHE-RSA-AES256-GCM-SHA384:DHE-RSA-
    AES128-GCM-SHA256:DHE-RSA-CHACHA20-POLY1305

# Activer la PFS lors de la reprise de session (pour TLS 1.3)
SSLOpenSSLConfCmd Options -AllowNoDHEKEX
```

Le module applicatif d'apache ne permet pas de désactiver explicitement le mode 0-RTT pour

une session TLS 1.3. Cependant, cette fonctionnalité n'est pour le moment pas supportée par `mod_ssl`.

NGINX

La configuration B.3 suppose l'usage d'un serveur NGINX en version 1.16 ou ultérieure. Elle correspond à une situation où le client est maîtrisé et souhaite négocier une session TLS 1.3 avec la suite cryptographique `TLS_AES_256_GCM_SHA384`. Plusieurs directives ne sont pas prises en charge par les versions antérieures [82].

Extrait B.3 – Exemple de configuration SSL NGINX 1.16+, face à un client maîtrisé

```
# Activer l'usage du protocole TLS
listen 443 ssl;

# Cacher le numéro de version
server_tokens off;

# Forcer l'usage de TLS1.3
ssl_protocols TLSv1.3;

# Fournir la chaîne de certification et les clés
ssl_certificate_key /chemin/vers/cle-privee.pem;
ssl_certificate /chemin/vers/chaine-certification.pem;

# Activer l'agravage OCSP
ssl_stapling on;
ssl_stapling_verify on;

# Utiliser une courbe recommandée, en accord avec le client maîtrisé
ssl_ecdh_curve secp521r1;

# Utiliser une suite recommandée, en accord avec le client maîtrisé
ssl_ciphers TLS_AES_256_GCM_SHA384;

# Désactiver 0-RTT
ssl_early_data off;
```


Annexe C

Liste des recommandations

R1	Restreindre la compatibilité en fonction du profil des clients	10
R2	Utiliser des composants logiciels à jour	17
R3	Privilégier TLS 1.3 et accepter TLS 1.2	17
R4	Ne pas utiliser SSLv2, SSLv3, TLS 1.0 et TLS 1.1	18
R5	Authentifier le serveur à l'échange de clé	19
R6	Échanger les clés en assurant toujours la PFS	20
R7	Échanger les clés avec l'algorithme ECDHE	21
R7-	Échanger les clés avec l'algorithme DHE	21
R8	Authentifier le serveur par certificat	22
R8-	Authentifier le serveur avec un mécanisme symétrique	23
R9	Privilégier AES ou ChaCha20	24
R9-	Tolérer Camellia et ARIA	24
R10	Utiliser un mode de chiffrement intègre	24
R10-	Tolérer le mode CBC avec <code>encrypt_then_mac</code>	25
R11	Utiliser SHA-2 comme fonction de hachage	26
R12	Disposer de plusieurs suites cryptographiques	27
R13	Préférer l'ordre de suites du serveur	27
R14	Utiliser les extensions du tableau 2.4	34
R15	Évaluer l'utilité des extensions du tableau 2.8	36
R16	Ne pas utiliser les extensions du tableau 2.9	36
R17	Utiliser un générateur d'aléa robuste	37
R18	Privilégier l'aléa du serveur avec un suffixe prédictible	38
R18-	Privilégier les aléas sans préfixe prédictible	38
R19	Ne pas utiliser la compression TLS	38
R20	Limiter la durée de vie des tickets	39
R21	Effectuer des reprises de session avec échange de clé	40
R21-	Effectuer des reprises de session avec PFS	40
R22	Toujours activer l'extension pour la renégociation sécurisée	40
R23	Ne pas transmettre de données 0-RTT	41
R24	Présenter un certificat signé avec SHA-2	43
R25	Présenter un certificat valide pendant 3 ans ou moins	44
R26	Utiliser des clés de taille suffisante	44

R27	Présenter un <i>KeyUsage</i> approprié	45
R28	Présenter un <i>ExtendedKeyUsage</i> approprié	45
R29	Présenter un <i>SubjectAlternativeName</i> approprié (côté serveur)	46
R30	Réserver chaque certificat à une seule terminaison TLS	46
R31		46
R32	Présenter un AKI correspondant au SKI défini par l'AC	47
R33	Présenter un certificat avec des sources de révocation	47
R34	Transmettre une chaîne de certificats ordonnée et complète	47
R35	Préférer l'agrafage OCSP	48
R36	Prévoir une redondance des moyens de révocation	48
R37	Réagir en <i>hard-fail</i>	49
R38	Utiliser des certificats enregistrés par CT	49
R39	Rejeter tous les certificats invalidés par CT	49

Bibliographie

- [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 et 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 et 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 et 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 et C. Allen, « The TLS Protocol Version 1.0 », RFC 2246, IETF, jan. 1999.
- [9] T. Dierks et E. Rescorla, « The Transport Layer Security (TLS) Protocol Version 1.1 », RFC 4346, IETF, avril 2006.
- [10] T. Dierks et E. Rescorla, « The Transport Layer Security (TLS) Protocol Version 1.2 », RFC 5246, IETF, août 2008.
- [11] E. Rescorla, « The Transport Layer Security (TLS) Protocol Version 1.3 », RFC 8446, IETF, août 2018.
- [12] R. Barnes, M. Thomson, A. Pironti et A. Langley, « Deprecating Secure Sockets Layer Version 3.0 », RFC 7568, IETF, juin 2015.
- [13] R. Braden, « Requirements for Internet Hosts - Communication Layers », RFC 1122, IETF, oct. 1989.

- 
- [14] E. Rescorla et N. Modadugu, « Datagram Transport Layer Security Version 1.2 », RFC 6347, IETF, jan. 2012.
- [15] E. Rescorla, « Diffie-Hellman Key Agreement Method », RFC 2631, IETF, juin 1999.
- [16] D. E. 3rd, « Transport Layer Security (TLS) Extensions : Extension Definitions », RFC 6066, IETF, jan. 2011.
- [17] H. Krawczyk et P. Eronen, « HMAC-based Extract-and-Expand Key Derivation Function (HKDF) », RFC 5869, IETF, mai 2010.
- [18] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley et W. Polk, « Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile », RFC 5280, IETF, mai 2008.
- [19] Mozilla, « Network Security Services ». <<https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS>>.
- [20] 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>.
- [21] Google, « CRLSets ». <<https://dev.chromium.org/Home/chromium-security/crlsets>>.
- [22] Mozilla, « Revoking Intermediate Certificates : Introducing OneCRL ». <<https://blog.mozilla.org/security/2015/03/03/revoking-intermediate-certificates-introducing-onecrl/>>, March 2015.
- [23] B. Laurie, A. Langley et E. Kasper, « Certificate Transparency », RFC 6962, IETF, juin 2013.
- [24] Common Vulnerabilities and Exposures, « CVE-2009-3555 ». <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3555>>, August 2009.
- [25] J. Rizzo et T. Duong, « Browser Exploit Against SSL/TLS ». <<https://packetstormsecurity.com/files/105499/Browser-Exploit-Against-SSL-TLS.html>>, October 2011.
- [26] Common Vulnerabilities and Exposures, « CVE-2014-0160 ». <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2014-0160>>, April 2014.
- [27] S. Turner et T. Polk, « Prohibiting Secure Sockets Layer (SSL) Version 2.0 », RFC 6176, IETF, mars 2011.
- [28] K. Moriarty et S. Farrell, « Deprecating TLSv1.0 and TLSv1.1 », Internet-Draft draft-ietf-tls-oldversions-deprecate-06.txt, IETF, jan. 2020.
- [29] P. Wouters, H. Tschofenig, J. Gilmore, S. Weiler et T. Kivinen, « Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) », RFC 7250, IETF, juin 2014.

- 
- [30] D. Bleichenbacher, « Chosen ciphertext attacks against protocols based on the rsa encryption standard pkcs 1.5 », in *CRYPTO* (H. Krawczyk, éd.), vol. 1462 in *Lecture Notes in Computer Science*, p. 1–12, Springer, 1998.
- [31] H. Böck, J. Somorovsky et C. Young, « Return of bleichenbacher’s oracle threat (ROBOT) », in *27th USENIX Security Symposium (USENIX Security 18)*, (Baltimore, MD), p. 817–849, USENIX Association, 2018.
- [32] E. Ronen, R. Gillham, D. Genkin, A. Shamir, D. Wong et Y. Yarom, « The 9 lives of bleichenbacher’s CAT : new cache attacks on TLS implementations », *IACR Cryptology ePrint Archive*, vol. 2018, p. 1173, 2018.
- [33] ETSI, « Middlebox security protocol ; part 3 : Profile for enterprise network and data centre access control ». <https://www.etsi.org/deliver/etsi_ts/103500_103599/10352303/01.01.01_60/ts_10352303v010101p.pdf>, October 2018.
- [34] D. Gillmor, « Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS) », RFC 7919, IETF, août 2016.
- [35] 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>.
- [36] P. Eronen et H. Tschofenig, « Pre-Shared Key Ciphersuites for Transport Layer Security (TLS) », RFC 4279, IETF, déc. 2005.
- [37] D. Taylor, T. Wu, N. Mavrogiannopoulos et T. Perrin, « Using the Secure Remote Password (SRP) Protocol for TLS Authentication », RFC 5054, IETF, nov. 2007.
- [38] Common Vulnerabilities and Exposures, « CVE-2006-4339 ». <<https://nvd.nist.gov/vuln/detail/CVE-2006-4339>>, May 2006.
- [39] Mozilla, « RSA Signature Forgery in NSS ». <<https://www.mozilla.org/en-US/security/advisories/mfsa2014-73/>>, September 2014.
- [40] Common Vulnerabilities and Exposures, « CVE-2016-1494 ». <<https://nvd.nist.gov/vuln/detail/CVE-2016-1494>>, January 2016.
- [41] N. J. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering et J. C. Schuldt, « On the Security of RC4 in TLS and WPA ». <<https://cr.yp.to/streamciphers/rc4biases-20130708.pdf>>, July 2013.
- [42] M. Vanhoef et F. Piessens, « All your biases belong to us : Breaking rc4 in wpa-tkip and TLS », 2015.
- [43] A. Popov, « Prohibiting RC4 Cipher Suites », RFC 7465, IETF, fév. 2015.
- [44] A. Langley, W. Chang, N. Mavrogiannopoulos, J. Strombergson et S. Josefsson, « ChaCha20-Poly1305 Cipher Suites for Transport Layer Security (TLS) », RFC 7905, IETF, juin 2016.

- [45] S. Kelly, « Security Implications of Using the Data Encryption Standard (DES) », RFC 4772, IETF, déc. 2006.
- [46] K. Bhargavan et G. Leurent, « On the practical (in-)security of 64-bit block ciphers : Collision attacks on http over tls and openvpn », 2016.
- [47] N. J. AlFardan et 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*, p. 526–540, IEEE Computer Society, 2013.
- [48] Adam Langley, « Lucky Thirteen attack on TLS CBC ». <<https://www.imperialviolet.org/2013/02/04/luckythirteen.html>>, February 2013.
- [49] B. Möller, T. Duong et K. Kotowicz, « This POODLE bites : Exploiting the SSL 3.0 Fallback », rap. tech., September 2014.
- [50] A. Joux, « Authentication Failures in NIST version of GCM », 2006.
- [51] H. Böck, A. Zauner, S. Devlin, J. Somorovsky et P. Jovanovic, « Nonce-disrespecting adversaries : Practical forgery attacks on GCM in TLS », *IACR Cryptology ePrint Archive*, vol. 2016, p. 475, 2016.
- [52] H. Y. Xiaoyun Wang, « Advances in cryptology – crypto 2005 : 25th annual international cryptology conference, santa barbara, california, usa, august 14-18, 2005. proceedings », 2005.
- [53] M. Stevens, P. Karpman et T. Peyrin, « Freestart collision for full SHA-1 ». <<https://eprint.iacr.org/2015/967>>, 2016.
- [54] M. Stevens, E. Bursztein, P. Karpman, A. Albertini et Y. Markov, « The first collision for full SHA-1 », 2017.
- [55] M. J. Dworkin, « SHA-3 Standard : Permutation-Based Hash and Extendable-Output Functions ». <https://www.nist.gov/manuscript-publication-search.cfm?pub_id=919061>, August 2015.
- [56] Internet Assigned Numbers Authority, « Transport Layer Security (TLS) Extensions ». <<https://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xhtml>>.
- [57] S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk et B. Moeller, « Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) », RFC 4492, IETF, mai 2006.
- [58] P. Gutmann, « Encrypt-then-MAC for Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) », RFC 7366, IETF, sept. 2014.
- [59] K. Bhargavan, A. Delignat-Lavaud, A. Pironti, A. Langley et M. Ray, « Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension », RFC 7627, IETF, sept. 2015.

- 
- [60] E. Rescorla, M. Ray, S. Dispensa et N. Oskov, « Transport Layer Security (TLS) Renegotiation Indication Extension », RFC 5746, IETF, fév. 2010.
- [61] D. McGrew et E. Rescorla, « Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP) », RFC 5764, IETF, mai 2010.
- [62] H. Schulzrinne, S. Casner, R. Frederick et V. Jacobson, « RTP : A Transport Protocol for Real-Time Applications », RFC 3550, IETF, juil. 2003.
- [63] S. Friedl, A. Popov, A. Langley et E. Stephan, « Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension », RFC 7301, IETF, juil. 2014.
- [64] A. Langley, « A Transport Layer Security (TLS) ClientHello Padding Extension », RFC 7685, IETF, oct. 2015.
- [65] M. Thomson, « Record Size Limit Extension for TLS », RFC 8449, IETF, août 2018.
- [66] S. Santesson, A. Medvinsky et J. Ball, « TLS User Mapping Extension », RFC 4681, IETF, oct. 2006.
- [67] Y. Pettersen, « The Transport Layer Security (TLS) Multiple Certificate Status Request Extension », RFC 6961, IETF, juin 2013.
- [68] J. Salowey, H. Zhou, P. Eronen et H. Tschafenig, « Transport Layer Security (TLS) Session Resumption without Server-Side State », RFC 5077, IETF, jan. 2008.
- [69] K. G. Paterson, T. Ristenpart et T. Shrimpton, « Tag size does matter : Attacks and proofs for the tls record protocol », in *ASIACRYPT*, vol. 7073 in *Lecture Notes in Computer Science*, p. 372–389, Springer, 2011.
- [70] M. Brown et R. Housley, « Transport Layer Security (TLS) Authorization Extensions », RFC 5878, IETF, mai 2010.
- [71] N. Mavrogiannopoulos et D. Gillmor, « Using OpenPGP Keys for Transport Layer Security (TLS) Authentication », RFC 6091, IETF, fév. 2011.
- [72] J. Callas, L. Donnerhacke, H. Finney, D. Shaw et R. Thayer, « OpenPGP Message Format », RFC 4880, IETF, nov. 2007.
- [73] Y. Nir, S. Josefsson et M. Pegourie-Gonnard, « Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier », RFC 8422, IETF, août 2018.
- [74] S. Hollenbeck, « Transport Layer Security Protocol Compression Methods », RFC 3749, IETF, mai 2004.
- [75] J. Rizzo et T. Duong, « The CRIME attack ». <http://www.ekoparty.org/archive/2012/CRIME_ekoparty2012.pdf>, 2012.

- 
- [76] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Pironti et P.-Y. Strub, « Triple Handshakes and Cookie Cutters : Breaking and Fixing Authentication over TLS ». <<https://www.mitls.org/downloads/tlsauth.pdf>>, May 2014.
 - [77] M. Rex, « MITM attack on delayed TLS-client auth through renegotiation ». <<https://www.ietf.org/mail-archive/web/tls/current/msg03928.html>>, November 2009.
 - [78] 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>.
 - [79] CA/Browser Forum, « Baseline Requirements for the Issuance and Management of Publicly-Trusted Certificates ». <<https://cabforum.org/wp-content/uploads/CA-Browser-Forum-BR-1.6.5.pdf>>, April 2019.
 - [80] M. Prince, « The Hidden Costs of Heartbleed ». <<https://blog.cloudflare.com/the-hard-costs-of-heartbleed/>>, April 2014.
 - [81] The Apache Software Foundation, « Apache Module mod_ssl ». <https://httpd.apache.org/docs/2.4/en/ssl/ssl_howto.html>.
 - [82] Nginx, « Module ngx_http_ssl_module ». <http://nginx.org/en/security_advisories.html>.

Acronymes

AC	Autorité de Certification
AES	Advanced Encryption Standard
AIA	Authority Information Access
AKI	Authority Key Identifier
CRL	Certificate Revocation List
CRLDP	Certificate Revocation List Distribution Point
CT	Certificate Transparency
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
ETSI	European Telecommunications Standards Institute
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
IGC	Infrastructure de Gestion de Clés
LTS	Long Term Support
OCSP	Online Certificate Status Protocol
PFS	Perfect Forward Secrecy
PSK	Pre-Shared Key



RGS	Référentiel Général de Sécurité
RTP	Real-time Transport Protocol
SAN	Subject Alternative Name
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
URL	Uniform Resource Locator

À propos de l'ANSSI

L'Agence nationale de la sécurité des systèmes d'information (ANSSI) a été créée le 7 juillet 2009 sous la forme d'un service à compétence nationale.

En vertu du décret n° 2009-834 du 7 juillet 2009 modifié par le décret n° 2011-170 du 11 février 2011, l'agence assure la mission d'autorité nationale en matière de défense et de sécurité des systèmes d'information. Elle est rattachée au Secrétaire général de la défense et de la sécurité nationale, sous l'autorité du Premier ministre.

Pour en savoir plus sur l'ANSSI et ses missions, rendez-vous sur www.ssi.gouv.fr.

Juillet 2019

Licence ouverte / Open Licence (Etalab v1)

Agence nationale de la sécurité des systèmes d'information
ANSSI - 51 boulevard de La Tour-Maubourg - 75700 PARIS 07 SP
Site internet : www.ssi.gouv.fr
Messagerie : communication@ssi.gouv.fr