

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 \LaTeX .

Version 1.1 : 19/08/2016

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 | 13 |
| 2 Négociation des paramètres TLS | 17 |
| 2.1 Versions de protocole | 17 |
| 2.2 Suites cryptographiques | 19 |
| 2.3 Extensions | 26 |
| 2.4 Considérations additionnelles | 33 |
| 3 Mise en place de l'IGC | 37 |
| 3.1 Attributs des certificats X.509 | 37 |
| 3.2 Contrôle de validité | 41 |
| A Référentiel des suites cryptographiques | 45 |
| A.1 Suites recommandées | 45 |
| A.2 Suites dégradées | 47 |
| B Exemples d'application des recommandations | 49 |
| C Liste des recommandations | 53 |
| Bibliographie | 55 |
| Acronymes | 61 |

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 et l’intégrité des échanges. Le serveur est nécessairement authentifié, et des fonctions additionnelles permettent l’authentification du client lorsqu’un tel besoin a été identifié.

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 vocation à servir de registre commun de recommandations qui puisse être adapté en fonction du périmètre de chaque projet cherchant à respecter, ou à permettre de respecter, les bonnes pratiques liées 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.

En effet, l'état d'une connexion TLS résulte d'un ensemble de facteurs qui sont rarement sous le contrôle intégral d'une seule et même entité. Cet état dépend globalement :

- des capacités de la pile TLS, en charge notamment de la logique d'automate et des calculs cryptographiques, telle que OpenSSL, GnuTLS ou encore miTLS ;
- des capacités du 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 et TLS 1.2, 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.

À défaut de correspondance exacte, les capacités successives de la liste précédente sont strictement inclusives. Un logiciel ne dispose pas de plus de capacités que la pile TLS sur laquelle il s'appuie. De même, un fichier de configuration ne permet pas de déployer des capacités qui ne seraient pas prises en charge par le logiciel en question.

Contrôler un élément de cette chaîne permet donc d'exclure certaines capacités indésirables. Par contre, il n'est pas toujours possible pour une entité isolée de déployer ou d'utiliser les paramètres optimaux vis-à-vis des recommandations de sécurité.

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. Ce travail de transposition, dont les circonstances sont trop variables pour être intégralement couvertes, fait néanmoins l'objet de publications additionnelles de la part de l'ANSSI [1, 2]. Des exemples courants 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 **R3** préconise d'utiliser uniquement TLS 1.2, tandis que la **R3-** tolère les versions TLS 1.1 et TLS 1.0. Certains intégrateurs peuvent choisir d'exclure de leurs logiciels produits, dès la compilation de la pile TLS, les versions TLS 1.1 et TLS 1.0. Pour d'autres, il peut être nécessaire d'inclure ces versions dégradées afin de répondre aux besoins en compatibilité de certains clients, quitte à permettre aux clients plus respectueux des recommandations d'exclure ces versions par l'intermédiaire d'options de configuration.

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 . |
| Rx-- | Cette dérogation représente le plus faible niveau de confiance admis pour rester en adéquation avec le guide. |

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 **R3**, la **R3-** et la **R3--**, c'est la **R3** qui devrait être suivie : le serveur et ses clients doivent être configurés pour utiliser TLS 1.2, et rejeter toute tentative de connexion avec une version antérieure du protocole.

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. Typiquement, cherchant à communiquer avec un serveur web public, plusieurs logiciels clients datés risquent de ne pas prendre en charge une partie des fonctions les plus recommandées.

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-** ou **Rx--**) s'appuie sur ce profil.

Chapitre 1

Présentation du protocole TLS

1.1 Déroutement des sessions TLS

Le développement du protocole TLS a suivi plusieurs itérations [8, 9, 10] depuis la conception du protocole SSL, désormais obsolète [11]. 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 [12]. Pour les protocoles à datagrammes, comme UDP⁸, une variante DTLS⁹ a été définie [13]. 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 communément. Ce paramètre est établi au cours d'une phase *TLS handshake* qui précède la protection effective des échanges. 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`, en complément des messages de type `change_cipher_spec` qui signalent l'échéance de sa mise en œuvre.

La figure 1.1 illustre la négociation de ces paramètres dans un cas générique. Elle fait intervenir les échanges suivants entre le client et le serveur :

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 éphémère qu'il signe à l'aide de la clé privée associée à la clé publique précédente ;

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*.

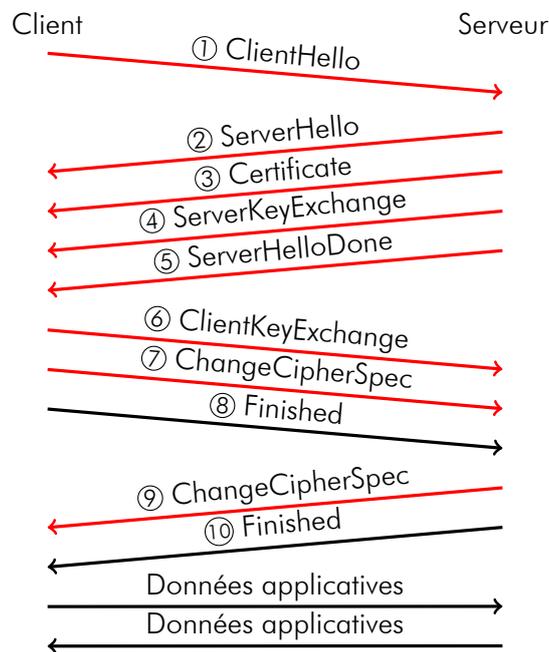


Figure 1.1 – Initiation générique d’une session TLS

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 clés en choisissant à son tour une valeur éphémère 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 clés éphémères 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 prévenir le déchiffrement de messages de sessions passées quand bien même la clé privée liée au certificat du serveur serait compromise, en négociant un secret éphémère à l’aide d’un échange de clés Diffie–Hellman [14]. 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, 15]. 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,

11. Perfect Forward Secrecy.



un `ClientHello` peut par exemple contenir des informations additionnelles relatives aux courbes elliptiques prises en charge par le client pour effectuer des calculs ECC ¹².

À l'issue du *handshake*, le client et le serveur disposent d'un secret partagé, le *premaster secret*, calculé grâce aux éléments contenus dans le `ServerKeyExchange` et le `ClientKeyExchange`. Le *premaster secret* est dérivé de part et d'autre en un même *master secret* qui prend en compte les aléas contenus dans le `ClientHello` et le `ServerHello`. Enfin, le *master secret* est à son tour dérivé afin de générer les 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 redéfinir les paramètres de sécurité ou encore de rafraîchir les clés de chiffrement sans interrompre la session TLS, à l'aide d'un nouveau *handshake*. 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`.

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 [16] 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 sujet (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.

12. Elliptic Curve Cryptography.

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 [17].

Les certificats contiennent plusieurs attributs, tels qu'une clé publique et une période de validité, qui sont habituellement complétés par des extensions X.509v3. L'ANSSI recommande à cet effet le suivi de l'annexe A4 du RGS¹³ [18]. Les extensions permettent notamment de préciser le cadre d'utilisation d'un certificat et de renforcer les assurances de l'IGC¹⁴. Par exemple, la présence d'une extension EV¹⁵ signale que des exigences additionnelles ont été portées au processus de vérification d'identité mené par l'AC¹⁶ 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 CRL¹⁷ : 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 CRLDP¹⁸ de chaque certificat émis par cette AC ;
- le protocole OCSP¹⁹ : 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 AIA²⁰ de chaque certificat émis par l'AC.

Des initiatives récentes 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 constituer une nuisance vis-à-vis 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 [19]. Mozilla a lancé un projet similaire pour son navigateur Firefox, sous le nom de OneCRL [20].

13. Référentiel Général de Sécurité.

14. Infrastructure de Gestion de Clés.

15. Extended Validation.

16. Autorité de Certification.

17. Certificate Revocation List.

18. Certificate Revocation List Distribution Point.

19. Online Certificate Status Protocol.

20. Authority Information Access.



L'agrafage OCSP²¹ 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 requêter les répondeurs lui-même [15].

En complément des mécanismes de révocation, le programme Certificate Transparency, initié par Google et standardisé par l'IETF [21], 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 SCT²² 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.

21. En anglais, *OCSP Stapling*.

22. 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. [22, 23, 24, 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 solutions dont dépend le déploiement de TLS doivent être tenues 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 cinq déclinaisons du protocole exploitées : SSLv2, SSLv3, TLS 1.0, TLS 1.1 et TLS 1.2. Les spécifications de TLS 1.3 sont en cours d'élaboration par l'IETF [25].

La version utilisée au cours d'une session est négociée pendant le *handshake*. Le client signale dans son `ClientHello` la plus haute version qu'il prend en charge, et le serveur répond dans son `ServerHello` avec la plus haute version qu'il prend en charge et qui soit au plus égale à celle préférée par le client.

À ce jour, sous certaines conditions précisées dans la suite de cette section, TLS 1.0 reste considéré d'usage sûr. Cependant, TLS 1.1 résout une partie des attaques établies contre TLS 1.0, et à son tour TLS 1.2 a été défini de façon plus robuste que TLS 1.1.

S'il apparaît nécessaire de prendre en charge d'autres versions que TLS 1.2, il est recommandé pour leur utilisation de prendre en charge la suite `TLS_FALLBACK_SCSV` définie ci-dessous. En particulier, bien que l'usage de TLS 1.1 ou TLS 1.0 ne soit pas encouragé dans le cas général, ces versions peuvent être tolérées s'il est nécessaire de communiquer avec des composants (non conformes aux recommandations) qui ne prennent pas en charge TLS 1.2.

R3 Utiliser uniquement TLS 1.2

La connexion doit être réalisée avec la version TLS 1.2. Toute tentative de connexion avec une version antérieure doit être rejetée.

R3 – Privilégier TLS 1.2 et tolérer TLS 1.1 et TLS 1.0

La version TLS 1.2 doit être prise en charge et privilégiée. Les versions TLS 1.1 et TLS 1.0, avec prise en charge de TLS_FALLBACK_SCSV, sont tolérées.

La publication de SSLv3 a répondu à plusieurs problèmes de conception identifiés pour SSLv2. Cependant, cette version du protocole n'est à ce jour plus considérée comme sûre. En juin 2015, l'IETF a fait état des faiblesses de SSLv3 et l'a formellement déclaré obsolète [11].

En avril 2015, la suite cryptographique factice TLS_FALLBACK_SCSV a été définie afin de limiter les risques d'attaque par dégradation de version, notamment liés aux implémentations tentant d'établir une connexion SSLv3 suite à l'échec des connexions TLS [26]. Suite à un premier échec de connexion, un client qui met en œuvre TLS_FALLBACK_SCSV et souhaite essayer un ClientHello de version dégradée est tenu d'y ajouter la SCSV²³ précédente. De cette façon, si un serveur observe la SCSV dans un ClientHello qui annonce une version inférieure à la plus récente version que lui-même prend en charge, il sait qu'un premier échange a échoué. Il peut, en fonction du contexte, en inférer une tentative d'attaque, et alerter le client.

R3 -- Privilégier TLS 1.2 et tolérer TLS 1.1, TLS 1.0 et SSLv3

La version SSLv3 est fortement déconseillée dans le cas général. En complément de la R3–, dans le cas où un besoin fort de compatibilité a été identifié, et seulement dans ce cas, la version SSLv3 accompagnée de TLS_FALLBACK_SCSV est tolérée.

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. 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. Cette version du protocole est donc à bannir.

23. Signaling Cipher Suite Value.

R4 Ne pas utiliser SSLv2

La version SSLv2 est fortement déconseillée en toutes circonstances. De plus, il faut privilégier l'usage de composants logiciels qui ne prennent pas en charge cette version du protocole.

2.2 Suites cryptographiques

Le `ClientHello` contient un ensemble de suites cryptographiques que le client est prêt à utiliser au cours de la session. Il est attendu du serveur qu'il en sélectionne une parmi celles-ci, après comparaison avec celles 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 de la suite retenue.

Chaque suite constitue une combinaison des mécanismes cryptographiques suivants :

- un mécanisme d'échange de clés, qui précise un algorithme d'échange et éventuellement l'algorithme de signature utilisé pour authentifier les échanges. `RSA`, `ECDHE_RSA` et `PSK` en sont quelques exemples ;
- des mécanismes assurant la confidentialité et l'intégrité des données échangées après le *handshake*, définis :
 - soit comme la composition d'un algorithme de chiffrement et d'une fonction de hachage utilisée en mode HMAC, telle `AES_256_CBC_SHA384` ;
 - soit comme un mode de chiffrement intègre, aussi appelé mode combiné, offrant simultanément chiffrement et intégrité, tel `AES_256_GCM` ;
- de manière optionnelle, pour des suites définies pour la version TLS 1.2, une fonction de hachage utilisée pour la dérivation des secrets à partir du *premaster secret*. Cette option est notamment utilisée par les suites utilisant le chiffrement intègre `AES_GCM`.

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 clés `ECDHE_RSA` avec le mode de chiffrement intègre `AES_256_GCM` complété de `SHA384` pour la dérivation des secrets.

Parmi les algorithmes standardisés avant juin 2016 pour un usage dans le cadre de TLS, les recommandations de la présente section dressent une liste blanche des algorithmes souhaitables : tout ce qui n'est pas recommandé est implicitement déconseillé. En particulier, l'usage de la fonction de chiffrement de flux RC4 et de la fonction de hachage MD5 est déconseillé.

À ce jour, aucune recommandation n'est émise quant au mode de chiffrement intègre `CHACHA20_POLY1305`, dont l'usage avec TLS a été standardisé en juin 2016 [27].

Échange de clés

Il existe plusieurs méthodes d'échange de clés, dont certaines n'exigent pas l'authentification du serveur. Cependant, en l'absence de cette protection, l'échange de clés 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.

R5 Authentifier le serveur à l'échange de clés

Au cours d'un échange de clés, le serveur doit être authentifié par le client. Les alternatives anonymisées de ces échanges sont fortement déconseillées.

Dans le cas général, l'échange de clés s'appuie, ou bien sur le chiffrement asymétrique d'un secret à l'aide de la clé publique du serveur, 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 DH²⁴, parfois désigné par FFDH²⁵, représente l'échange Diffie–Hellman historique dont l'arithmétique sous-jacente est relative à un groupe multiplicatif. L'arithmétique de l'algorithme ECDH²⁶ 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 (PFS). Celle-ci est assurée par les variantes éphémères des algorithmes précédents, DHE²⁷ et ECDHE²⁸, pour lesquelles les clés Diffie–Hellman sont générées à chaque nouvelle session. Dans ce contexte, l'authentification de l'échange de clés passe par la signature du `ServerKeyExchange` à l'aide de la clé privée du serveur.

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).

24. Diffie–Hellman.

25. Finite Field Diffie–Hellman.

26. Elliptic Curve Diffie–Hellman.

27. Diffie–Hellman Ephemeral.

28. Elliptic Curve Diffie–Hellman Ephemeral.

R6 – Échanger les clés sans toujours assurer la PFS

Les échanges Diffie–Hellman éphémères doivent être pris en charge et privilégiés. Les échanges basés sur le chiffrement d'un secret à l'aide de la clé du serveur, qui n'assurent pas la confidentialité persistante, sont tolérés.

Concernant 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 [28]. Parmi les courbes enregistrées auprès de l'IANA [29], les courbes éprouvées retenues sont `secp256r1`, `secp384r1`, `secp521r1` (aussi appelées P-256, P-384 et P-521), ainsi que `brainpoolP256r1`, `brainpoolP384r1` et `brainpoolP512r1`.

Il est à noter que la négociation de groupe DHE n'est, à ce jour, permise par le protocole. Lorsque cette solution est privilégiée, le groupe est en effet imposé par le serveur dans un `ServerKeyExchange`. De fait, un client qui souhaite assurer la PFS avec DHE, mais se verrait présenter un groupe jugé insatisfaisant, n'aurait d'autre solution que d'interrompre la session.

Par contraste, la négociation des paramètres ECDHE est permise par l'extension `supported_groups`. Celle-ci rend en effet possible la sélection de courbes de paramètres implicites, enregistrées auprès de l'IANA. L'usage de ECDHE est donc préférable à celui de DHE dès lors qu'une des parties communicantes n'est pas contrôlée.

R7 Échanger les clés avec ECDHE

Les échanges de clés ECDHE doivent être privilégiés, à l'aide des courbes `secp256r1`, `secp384r1`, `secp521r1`, `brainpoolP256r1`, `brainpoolP384r1` ou `brainpoolP512r1`.

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 pour les échanges les plus sensibles. 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 DHE

Les échanges DHE sont tolérés avec des groupes 2048-bits ou plus (3072-bits ou plus si l'information doit être protégée au-delà de 2030).



D'autres méthodes d'échange de clés telles que PSK²⁹ [30] et SRP³⁰ [31] reposent sur le partage au préalable d'un secret entre le client et le serveur. Plus complexes à déployer et à maintenir, elles ne constituent des alternatives valables que dans les environnements maîtrisés, notamment pour des applications d'infrastructure.

Chiffrement symétrique

Le *premaster secret* négocié à l'aide de l'échange précédent est dérivé de part et d'autre en un *master secret*, duquel est notamment dérivée la clé de chiffrement symétrique utilisée pour la protection en confidentialité des échanges qui suivent la phase de négociation. L'algorithme de chiffrement en jeu est cependant fixé dès l'élection d'une suite cryptographique.

Chiffrement de flux

Actuellement, RC4 est la seule fonction de chiffrement de flux proposée parmi les suites standardisées. Cependant, suite à des travaux [32] 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 interdit l'utilisation de cette fonction [33].

Chiffrement par bloc

Par opposition, plusieurs fonctions de chiffrement par bloc peuvent être utilisées pour sécuriser des connexions TLS. Publié en 1977, DES³¹ n'est à ce jour plus sûr [34]. Il faut lui privilégier son successeur, AES³², avec une taille de clé de 128 bits ou plus, en accord avec le RGS [28]. Bien que soumis à moins d'examens, Camellia et ARIA offrent à ce jour une sécurité comparable à AES et peuvent être envisagés comme alternatives.

L'algorithme Triple DES constitue un dernier recours. Cependant, comme l'algorithme DES sous-jacent, il souffre d'une taille de bloc réduite. De ce fait, afin d'éviter leur compromission, il est nécessaire de renouveler les clés de chiffrement de Triple DES au minimum à chaque gigaoctet de données applicatives échangées.

Concernant le choix entre une clé de 128 bits ou de 256 bits pour AES, 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.

29. Pre-Shared Key.

30. Secure Remote Password.

31. Data Encryption Standard.

32. Advanced Encryption Standard.

R8 Chiffrer avec AES

Les suites mettant en œuvre l'algorithme de chiffrement par bloc AES-256 sont à privilégier. L'algorithme AES-128 constitue une alternative acceptable.

R8 – Chiffrer avec Camellia ou 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.

R8 – Privilégier AES et tolérer 3DES avec rafraîchissement des clés

Le chiffrement AES doit être pris en charge et privilégié. Les suites mettant en œuvre l'algorithme de chiffrement par bloc Triple DES sont tolérées, sous réserve que les clés de chiffrement associées soient renouvelées au minimum à chaque gigaoctet de données applicatives échangées.

Les modes de chiffrement mettant en œuvre l'algorithme de chiffrement par bloc sélectionné sont décrits après l'introduction du mécanisme d'intégrité reposant sur une fonction de hachage.

Code d'authentification

Une fois la session TLS établie, chaque *record* envoyé est protégé en intégrité. Lorsque la suite cryptographique n'utilise pas de mode de chiffrement intègre, il s'agit d'un motif d'intégrité 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 d'un élément de la famille SHA-2.

La fonction MD5 a fait l'objet de nombreuses attaques, qui ont motivé son exclusion par le RGS. Depuis 2005, plusieurs études ont aussi altéré la robustesse de SHA-1 [35, 36]. Bien que HMAC-SHA1 ne soit pas directement remis en question à ce jour, en accord avec le RGS, son utilisation est déconseillée. Il faut préférer l'usage des deux déclinaisons de SHA-2 : SHA-256 et SHA-384. Standardisée en 2015, l'usage de la fonction SHA-3 [37] pour le protocole TLS n'a pas été spécifié à ce jour.

R9 Construire le HMAC avec SHA-2

Le HMAC permettant l'authentification doit être construit à l'aide d'un représentant de la famille SHA-2 : SHA-256 ou bien SHA-384.

R9 – Privilégier les HMAC avec SHA-2 et tolérer les HMAC avec SHA-1

La construction de HMAC avec SHA-2 doit être prise en charge et privilégiée. L'usage de HMAC construits à l'aide de SHA-1 est toléré.

Mode de chiffrement et d'intégrité

Pour les suites cryptographiques définies par les versions du protocole TLS antérieures à la version 1.2, un algorithme de chiffrement par bloc est utilisé dans le mode de chiffrement CBC pour assurer le chiffrement, et combiné avec un 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 à travers un grand nombre de sessions [38]. De telles fuites d'information ne peuvent être complètement évitées qu'au prix d'un gros effort d'implémentation [39].

Si l'extension `encrypt_then_mac` est utilisée, l'ordre des opérations est inversé : le HMAC porte sur les données déjà chiffrées. Ceci permet d'éviter les fuites d'information lors du déchiffrement, et de prévenir certaines vulnérabilités [40, 38].

La version 1.2 du protocole TLS 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. Des suites offrant les modes d'opération GCM et CCM ont ainsi été standardisées.

R10 Utiliser un mode de chiffrement robuste

La suite cryptographique retenue doit mettre en œuvre un mode de chiffrement intègre, ou bien la combinaison CBC + HMAC en conjonction avec l'extension `encrypt_then_mac`.

R10- Utiliser le mode CBC sans `encrypt_then_mac`

Les suites cryptographiques mettant en œuvre le mode CBC + HMAC en l'absence de l'extension `encrypt_then_mac` sont tolérées. La prise en charge de modes de chiffrement intègre est conseillée, mais pas obligatoire.

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 et 2.2. Pour rappel, l'extension `encrypt_then_mac` est recommandée pour l'utilisation des suites exploitant le mode de chiffrement CBC.

| 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 |
| 0xC024 | TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 |
| 0xC023 | TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 |

Table 2.1 – Suites TLS 1.2 recommandées avec un serveur disposant d'une clé ECDSA

| Code TLS | Suite cryptographique |
|----------|---------------------------------------|
| 0xC030 | TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 |
| 0xC02F | TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 |
| 0xC028 | TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 |
| 0xC027 | TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 |

Table 2.2 – Suites TLS 1.2 recommandées avec un serveur disposant d'une clé RSA

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.

Contextes de déploiement

Dans la situation où les profils de serveur et de client sont contrôlés, si le serveur dispose d'un certificat pour une clé ECDSA, alors il est suffisant pour le serveur et le client de n'utiliser qu'une seule des suites cryptographiques du tableau 2.1, par exemple `TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384` avec la courbe elliptique `secp256r1`.

Dans l'éventualité où celle-ci était compromise, il reste toutefois préférable que chacune des parties communicantes dispose d'une implémentation des autres suites du tableau 2.1 qui puisse être exploitée ultérieurement. Ce principe s'applique aussi aux six courbes elliptiques recommandées précédemment.

De même, si le serveur dispose d'un certificat pour une clé RSA, alors il est conseillé au serveur et au client de disposer de plusieurs des suites cryptographiques du tableau 2.2, et il est suffisant de n'en mettre en œuvre qu'une seule, par exemple `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`.

R11 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.

En l'absence de contrôle sur les clients, le serveur doit accorder plus de confiance en son propre traitement qu'en le leur. En particulier, le serveur doit privilégier les préférences entre suites cryptographiques établies dans sa propre configuration, plutôt que l'ordre des suites qui figure dans le `ClientHello`.

R12 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 [41].

Selon le contexte, ces extensions peuvent être purement informatives, ou bien indispensables au déroulement de la session TLS. Leur utilisation est souvent conditionnée à une application particulière. De ce fait, les extensions dont l'utilisation est acceptable pour certains contextes n'ont pas valeur à être utilisées pour toutes les sessions. 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 devrait 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) [42]

Cette extension informe le serveur des courbes elliptiques prises en charge par le client (s'il en existe), en accord avec le registre maintenu par l'IANA. Sa prise en charge et son usage sont obligatoires dès lors que le client ou le serveur souhaitent exploiter des fonctions ECC. L'ordre dans lequel les courbes sont présentées 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 de futurs messages du *handshake*, notamment le `ServerKeyExchange`. Dans le cadre recommandé d'une session TLS 1.2 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. Pour les versions antérieures du protocole, le mécanisme d'authentification est différent et cette extension ne doit pas être utilisée.

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

Cette extension signale la prise en charge de SCT. Dans le modèle CT³³ 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.

- `encrypt_then_mac` (0x0016) [43]

Cette extension signale la prise en charge de la construction cryptographique *encrypt-then-mac*, en complément 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) [44]

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 clés, certificats), à travers un condensat de tous les messages échangés au cours du *handshake*.

- `renegotiation_info` (0xFF01) [45]

Le mécanisme de renégociation et la nécessité de cette extension sont discutés en section 2.4.

33. Certificate Transparency.

Extensions acceptables, relatives à un contexte spécifique

- `server_name` (0x0000) [15]

Cette extension signale la prise en charge du mécanisme SNI³⁴. 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.

- `max_fragment_length` (0x0001) [15]

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*.

- `trusted_ca_keys` (0x0003) [15]

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 en lesquelles il a confiance, ou bien les condensats des certificats correspondants.

- `status_request` (0x0005) [15]

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 *CertificateStatus* contenant une réponse OCSP.

- `user_mapping` (0x0006) [46]

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.

- `cert_type` (0x0009) [47]

Cette extension permet de signaler la prise en charge de certificats OpenPGP [48], qui diffèrent des certificats X.509 historiques. 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) [42]

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.

34. Server Name Indication.

- `srp` (0x000C) [31]

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`.

- `use_srtp` (0x000E) [49]

Cette extension signale la prise en charge de DTLS-SRTP. SRTP est une variante sécurisée de RTP³⁵, un protocole optimisé pour le transfert de données en temps réel, telles que des flux vidéo [50]. 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) [51]

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.

- `status_request_v2` (0x0011) [52]

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`.

- `padding` (0x0015) [53]

Cette extension permet au client d'ajouter du bourrage³⁶ 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.

35. Real-time Transport Protocol.

36. En anglais, *padding*.

- `session_ticket` (0x0023) [54]

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

- `client_certificate_url` (0x0002) [15]

Cette extension signale la prise en charge de l'authentification client par l'intermédiaire de certificats distants, localisés par une ou plusieurs URL³⁷. 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) [15]

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.

- `client_authz` (0x0007) [55]

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) [55]

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.

37. Uniform Resource Locator.

- heartbeat (0x000F) [56]

Cette extension signale la prise en charge du mécanisme de *heartbeats*. Les *heartbeats* visent à maintenir une connexion TLS active, en forçant l'envoi immédiat d'un message *HeartbeatResponse* pour tout *HeartbeatRequest* reçu. Cette extension n'est généralement pas nécessaire et son implémentation défectueuse dans OpenSSL (versions 1.0.1 à 1.0.1f incluse) est à la source de la vulnérabilité Heartbleed [24]. Sa prise en charge est fortement déconseillée.

- client_certificate_type (0x0013) [57]

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) [57]

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.

Synthèse

R13 Utiliser les extensions du tableau 2.3

Les extensions recommandées dans le cadre général de l'utilisation du protocole TLS figurent dans le tableau 2.3. Elles doivent être prises en charge par les équipements maîtrisés et utilisées dans les contextes précisés.

| Code | Intitulé | Contexte d'utilisation |
|--------|------------------------|---------------------------|
| 0x000A | supported_groups | En cas de calculs ECC |
| 0x000D | signature_algorithms | En cas de version TLS 1.2 |
| 0x0012 | sct | En cas de certificat EV |
| 0x0016 | encrypt_then_mac | Toujours recommandé |
| 0x0017 | extended_master_secret | Toujours recommandé |
| 0xFF01 | renegotiation_info | En cas de renégociation |

Table 2.3 – Extensions TLS recommandées dans le cadre général

R14 Évaluer l'utilité des extensions du tableau 2.4

Les extensions du tableau 2.4 sont relatives à des déploiements spécifiques. Seules celles évaluées nécessaires doivent être implémentées et utilisées.

Les extensions les plus couramment utiles sont `server_name`, `status_request_v2`, `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> | En cas d'hébergement mutualisé |
| 0x0001 | <code>max_fragment_length</code> | En cas de contraintes réseau (rare) |
| 0x0003 | <code>trusted_ca_keys</code> | En cas de racines alternatives (rare) |
| 0x0005 | <code>status_request</code> | En cas d'agrafage OCSP |
| 0x0006 | <code>user_mapping</code> | En cas de clients indexés (rare) |
| 0x0009 | <code>cert_type</code> | En cas de certificats OpenPGP (rare) |
| 0x000B | <code>ec_point_formats</code> | En cas de points ECC compressés |
| 0x000C | <code>srp</code> | En cas de suites SRP |
| 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 |
| 0x0015 | <code>padding</code> | En cas de serveur défectueux (rare) |
| 0x0023 | <code>session_ticket</code> | En cas de reprises de session |

Table 2.4 – Extensions TLS relatives à un contexte spécifique

R15 Ne pas utiliser les extensions du tableau 2.5

Les extensions du tableau 2.5 sont toujours déconseillées.

| Code | Intitulé |
|--------|--------------------------------------|
| 0x0002 | <code>client_certificate_url</code> |
| 0x0004 | <code>truncated_hmac</code> |
| 0x0007 | <code>client_authz</code> |
| 0x0008 | <code>server_authz</code> |
| 0x000F | <code>heartbeat</code> |
| 0x0013 | <code>client_certificate_type</code> |
| 0x0014 | <code>server_certificate_type</code> |

Table 2.5 – Extensions TLS 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 `ServerHello`. 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`. Lorsque leur source est modifiable, l'utilisation de valeurs non prédictibles constitue de ce fait une protection essentielle contre les attaques par rejeu.

La création de ces valeurs doit faire appel à un générateur d'aléa de qualité, tel que défini par le RGS [28]. 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.

R16 Utiliser un générateur d'aléa robuste

Les aléas utilisés dans le `ClientHello` et le `ServerHello` doivent provenir de générateurs d'aléa fiables.

R17 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

La phase de négociation, en plus des paramètres pour la protection des *records*, détermine aussi un algorithme de compression utilisé sur les données applicatives avant leur chiffrement. En pratique, la fonction généralement retenue est celle qui laisse inchangées ces données. Il reste toutefois possible d'exploiter l'algorithme Deflate [58].

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 [59]. 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, en dehors de la consultation d'un serveur de contenu statique, la compression (autre que la transformation identité établie par défaut) reste à éviter.

R18 Ne pas utiliser la compression TLS

En dehors du cadre de la consultation d'un serveur de contenu statique, l'utilisation du mécanisme de compression TLS est déconseillée.

Reprise de session

Il existe deux méthodes de reprise de session, permettant d'écourter la phase de négociation TLS en restaurant des secrets précédemment établis.

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 clés 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 [54].

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* [60] est immédiatement contrée par l'absence de prise en charge du mécanisme de reprise de session.

Bien que les implémentations le permettent encore rarement, 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.

R19 Ne pas effectuer de reprise de session

L'utilisation du mécanisme de reprise de session est déconseillée.

R19 – 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.

R19 – Effectuer des reprises de session sans PFS

L'utilisation de tickets de session ou d'identifiants de session sans la suppression régulière des données associées est tolérée.

Renégociation

Le mécanisme de renégociation initialement conçu pour TLS 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 [22, 61].

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*.

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 souhaitait jamais effectuer de renégociation.

R20 Sécuriser les renégociations

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`.

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 [62].

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 [18]. Des informations complémentaires sur leur utilisation et leur structure figurent dans la RFC 5280 [16].

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 [28] est recommandé à leur sujet.

R21 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.

R22 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.

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.

R23 Utiliser des clés de taille suffisante

Pour une utilisation 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*.

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.

R24 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* n'interviennent pas dans le cadre de TLS.

R25 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 SAN³⁸ 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

38. Subject Alternative Name.

des adresses de messagerie électronique (*rfc822Name*). Ainsi, à titre d'exemple, un certificat de serveur web doit contenir le nom de domaine consulté par l'utilisateur dans la partie *Common Name* du champ *subject*, ou bien parmi les FQDN³⁹ des entrées *dNSName* de l'extension SAN.

R26 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 ubiquitaire 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.

R27 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.

Pour aller plus loin

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 AKI⁴⁰ 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 SKI⁴¹ du certificat de l'AC relatif à la clé de signa-

39. Fully Qualified Domain Name.

40. Authority Key Identifier.

41. Subject Key Identifier.

ture, 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.

R28 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.

R29 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 ordonnée et dans son intégralité, depuis le certificat terminal jusqu'au certificat intermédiaire signé par la racine de confiance. 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. Toutefois, dans le cas où le certificat terminal et le certificat racine sont un seul et même certificat auto-signé, il reste nécessaire de l'envoyer afin de faire connaître l'identité du serveur.

R30 Transmettre une chaîne de certificats ordonnée et complète

Les chaînes de certificats transmises à l'aide des messages `Certificate` doivent 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 [63].

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.

R31 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 seraient mis en œuvre, partageant les mêmes informations de révocation.

R32 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. Dans l'éventualité où aucun d'entre eux ne serait 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.

R33 Réagir en *hard-fail*

Les composants logiciels TLS 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. Actuellement, seuls les certificats 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.

R34 Utiliser des certificats enregistrés par CT

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

R35 Rejeter tous les certificats invalidés par CT

Les clients TLS doivent rejeter tous les certificats qui sont accompagnés d'un SCT invalide, et les certificats EV qui ne sont pas accompagnés d'un SCT.

R35 – Rejeter les certificats EV invalidés par CT

Les clients TLS doivent rejeter les certificats EV qui sont accompagnés d'un SCT invalide, ou 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 tableaux A.1 et A.2 listent les suites cryptographiques recommandées dans le cas général, standardisées pour l'usage avec 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`.

| 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 |
| 0xC024 | TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 |
| 0xC023 | TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 |

Table A.1 – Suites recommandées pour un serveur disposant d'une clé ECDSA

| Code TLS | Suite cryptographique |
|----------|---------------------------------------|
| 0xC030 | TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 |
| 0xC02F | TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 |
| 0xC028 | TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 |
| 0xC027 | TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 |

Table A.2 – Suites recommandées pour un serveur disposant d'une clé RSA

Bien que l'usage du chiffrement AES, 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.3 et A.4. Enfin, dans le cadre des déploiements avec *pre-shared keys*, les suites recommandées figurent dans le tableau A.5.

| Code TLS | Suite cryptographique |
|----------|--|
| 0xC087 | TLS_ECDHE_ECDSA_WITH_CAMELLIA_256_GCM_SHA384 |
| 0xC086 | TLS_ECDHE_ECDSA_WITH_CAMELLIA_128_GCM_SHA256 |
| 0xC073 | TLS_ECDHE_ECDSA_WITH_CAMELLIA_256_CBC_SHA384 |
| 0xC072 | TLS_ECDHE_ECDSA_WITH_CAMELLIA_128_CBC_SHA256 |
| 0xC08B | TLS_ECDHE_RSA_WITH_CAMELLIA_256_GCM_SHA384 |
| 0xC08A | TLS_ECDHE_RSA_WITH_CAMELLIA_128_GCM_SHA256 |
| 0xC077 | TLS_ECDHE_RSA_WITH_CAMELLIA_256_CBC_SHA384 |
| 0xC076 | TLS_ECDHE_RSA_WITH_CAMELLIA_128_CBC_SHA256 |

Table A.3 – Suites recommandées 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 |
| 0xC049 | TLS_ECDHE_ECDSA_WITH_ARIA_256_CBC_SHA384 |
| 0xC048 | TLS_ECDHE_ECDSA_WITH_ARIA_128_CBC_SHA256 |
| 0xC04D | TLS_ECDHE_RSA_WITH_ARIA_256_CBC_SHA384 |
| 0xC04C | TLS_ECDHE_RSA_WITH_ARIA_128_CBC_SHA256 |

Table A.4 – Suites recommandées pour l'usage d'ARIA

| Code TLS | Suite cryptographique |
|----------|---------------------------------------|
| 0xC038 | TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA384 |
| 0xC037 | TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256 |

Table A.5 – Suites recommandées pour l'usage de PSK

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.

Suites dégradées définies pour TLS 1.2

En l'absence de prise en charge de ECC, le tableau A.6 liste les suites tolérées.

| Code TLS | Suite cryptographique |
|----------|-------------------------------------|
| 0x009F | TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 |
| 0x009E | TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 |
| 0xC09F | TLS_DHE_RSA_WITH_AES_256_CCM |
| 0xC09E | TLS_DHE_RSA_WITH_AES_128_CCM |
| 0x006B | TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 |
| 0x0067 | TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 |

Table A.6 – Suites TLS 1.2 tolérées en l'absence de prise en charge de ECC

En l'absence de prise en charge des échanges Diffie–Hellman, le tableau A.7 liste les suites tolérées. Elles ne vérifient pas la propriété de PFS.

| Code TLS | Suite cryptographique |
|----------|---------------------------------|
| 0x009D | TLS_RSA_WITH_AES_256_GCM_SHA384 |
| 0x009C | TLS_RSA_WITH_AES_128_GCM_SHA256 |
| 0xC09D | TLS_RSA_WITH_AES_256_CCM |
| 0xC09C | TLS_RSA_WITH_AES_128_CCM |
| 0x003D | TLS_RSA_WITH_AES_256_CBC_SHA256 |
| 0x003C | TLS_RSA_WITH_AES_128_CBC_SHA256 |

Table A.7 – Suites TLS 1.2 tolérées en l'absence de prise en charge de DH

Dans le cadre de SRP, le tableau A.8 liste les suites tolérées. Celles-ci s'appuient sur un HMAC-SHA1 pour le contrôle d'intégrité des messages, et restent donc pertinentes pour TLS 1.0.

| Code TLS | Suite cryptographique |
|----------|--------------------------------------|
| 0xC021 | TLS_SRP_SHA_RSA_WITH_AES_256_CBC_SHA |
| 0xC01E | TLS_SRP_SHA_RSA_WITH_AES_128_CBC_SHA |

Table A.8 – Suites TLS tolérées pour authentification par mot de passe

Suites dégradées définies pour TLS 1.0

Lorsque la prise en charge de TLS 1.0 est indispensable, les suites du tableau A.9 sont à privilégier.

| Code TLS | Suite cryptographique |
|----------|--------------------------------------|
| 0xC00A | TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA |
| 0xC009 | TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA |
| 0xC014 | TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA |
| 0xC013 | TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA |

Table A.9 – Suites tolérées pour TLS 1.0

En ultime recours, les suites du tableau A.10 sont suggérées. Leur utilisation est déconseillée dans le cas général, pour les raisons précédemment exposées.

| Code TLS | Suite cryptographique |
|----------|---------------------------------------|
| 0x0039 | TLS_DHE_RSA_WITH_AES_256_CBC_SHA |
| 0x0033 | TLS_DHE_RSA_WITH_AES_128_CBC_SHA |
| 0xC008 | TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA |
| 0xC012 | TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA |
| 0x0016 | TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA |
| 0x0035 | TLS_RSA_WITH_AES_256_CBC_SHA |
| 0x002F | TLS_RSA_WITH_AES_128_CBC_SHA |
| 0x000A | TLS_RSA_WITH_3DES_EDE_CBC_SHA |

Table A.10 – Suites déconseillées pour TLS 1.0

Les suites dégradées ayant pour méthode d'échange PSK ou SRP, ainsi que celles reposant sur Camellia ou ARIA pour le chiffrement symétrique, ne sont pas représentées.

Les suites non mentionnées restantes, impliquant notamment des certificats DSA, des certificats de clés statiques DH ou ECDH, des HMAC-MD5, des échanges DH anonymisés, ou encore des algorithmes EXPORT, sont très fortement déconseillées.

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 1.0.2. Il s'agit d'une version LTS⁴² dont la maintenance sera assurée jusqu'au 31 décembre 2019.

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.

Afin d'être exploitées par d'autres programmes, ces bibliothèques pourront être sélectionnées au travers de `LD_LIBRARY_PATH`, ou bien explicitement à la compilation, lors de l'étape d'édition de liens.

42. Long Term Support.

Extrait B.1 – Exemple de compilation OpenSSL 1.0.2

```
# Pensez à télécharger la dernière version stable de la branche 1.0.2
$ tar -xzf openssl-1.0.2h.tar.gz
$ cd openssl-1.0.2h
$ ./config shared -D_FORTIFY_SOURCE=2 -fstack-protector-all \
  -no-ssl2 -no-ssl3 -no-ssl2-method -no-ssl3-method \
  -no-ec2m \
  -no-weak-ssl-ciphers \
  -no-seed -no-idea \
  -no-mdc2 -no-md2 -no-md4 -no-whirlpool \
  -no-rc2 -no-rc4 -no-rc5 -no-blowfish -no-cast \
  -no-heartbeats \
  -no-srp -no-psk -no-comp
$ make depend
$ make
$ ls lib*.so
libcrypto.so libssl.so
```

Application 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 1.10. Il est à noter que, en plus des directives de configuration qu'ils fournissent, ces modules peuvent décider d'appliquer eux-mêmes des paramètres 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 [64].

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

```
# Activer l'usage du protocole TLS
SSLEngine on
# Forcer l'usage de TLS1.2
SSLProtocol all -SSLv2 -SSLv3 -TLSv1 -TLSv1.1
# Fournir une chaîne de certification et une clé
SSLCertificateKeyFile "/chemin/vers/cle-privee.pem"
SSLCertificateFile "/chemin/vers/chaine-certification.pem"
# Désactiver le cache de session et les tickets
SSLSessionCache none
SSLSessionTickets off
# Désactiver la compression
SSLCompression off
# Désactiver les renégociations non sécurisées
SSLInsecureRenegotiation off
# Activer l'agravage OCSP
SSLUseStapling on
SSLStaplingCache shmcb:logs/ssl_stapling(32768)
# Utiliser les courbes recommandées pour ECDHE et supported_groups
SSLOpenSSLConfCmd ECDHParameters secp521r1:secp384r1:prime256v1:
    brainpoolP512r1:brainpoolP384r1:brainpoolP256r1
SSLOpenSSLConfCmd Curves secp521r1:secp384r1:prime256v1:brainpoolP512r1:
    brainpoolP384r1:brainpoolP256r1
# Utiliser un groupe de taille 2048 pour DH, généré par OpenSSL avec
# openssl dhparam 2048 > /chemin/vers/fichier/dhparams.pem
SSLOpenSSLConfCmd DHParameters "/chemin/vers/fichier/dhparams.pem"

SSLHonorCipherOrder on
# La branche 1.0.2 de OpenSSL :
# - ne prend pas en charge les suites CCM ;
# - ne prend pas en charge ARIA ;
# - prend en charge CAMELLIA (en revision `h'), mais uniquement combiné avec
# SHA-1. L'implémentation de SHA256 avec CAMELLIA128 est anticipée.
# Relations d'ordre utilisées :
# - ECDHE > DHE > chiffrement RSA ;
# - GCM > CBC ;
# - AES256 > AES128 > CAMELLIA256 > CAMELLIA128 ;
# - SHA384 > SHA256 ;
# - ECDSA > RSA.
#
# Avec le support actuel et futur prévu, en branche 1.0.2:
SSLCipherSuite ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:
    ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-
    AES256-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-
    AES128-SHA256:ECDHE-ECDSA-CAMELLIA256-SHA384:ECDHE-RSA-CAMELLIA256-SHA384:
    ECDHE-ECDSA-CAMELLIA128-SHA256:ECDHE-RSA-CAMELLIA128-SHA256:DHE-RSA-AES256
    -GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-SHA256:DHE-RSA-AES128
    -SHA256:AES256-GCM-SHA384:AES128-GCM-SHA256:AES256-SHA256:AES128-SHA256:
    CAMELLIA128-SHA256
```

NGINX

La configuration B.3 suppose l'usage d'un serveur NGINX en version 1.6 ou ultérieure. Elle correspond à une situation où le client est maîtrisé et souhaite négocier une session TLS 1.2 avec la suite cryptographique TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384. Plusieurs directives ne sont pas prises en charge par les versions antérieures [65].

Extrait B.3 – Exemple de configuration SSL NGINX 1.6+, face à un client maîtrisé

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

# Forcer l'usage de TLS1.2
ssl_protocols        TLSv1.2;

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

# Désactiver le cache de session et les tickets
ssl_session_cache    none;
ssl_session_tickets  off;

# (La compression est désactivée par défaut depuis la version 1.1.6)
# (Les renégociations sont désactivées par défaut depuis la version 0.8.24)

# Activer l'agrappage 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           ECDHE-ECDSA-AES256-GCM-SHA384;
```

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 | Utiliser uniquement TLS 1.2 | 18 |
| R3- | Privilégier TLS 1.2 et tolérer TLS 1.1 et TLS 1.0 | 18 |
| R3-- | Privilégier TLS 1.2 et tolérer TLS 1.1, TLS 1.0 et SSLv3 | 18 |
| R4 | Ne pas utiliser SSLv2 | 19 |
| R5 | Authentifier le serveur à l'échange de clés | 20 |
| R6 | Échanger les clés en assurant toujours la PFS | 20 |
| R6- | Échanger les clés sans toujours assurer la PFS | 21 |
| R7 | Échanger les clés avec ECDHE | 21 |
| R7- | Échanger les clés avec DHE | 21 |
| R8 | Chiffrer avec AES | 23 |
| R8- | Chiffrer avec Camellia ou ARIA | 23 |
| R8-- | Privilégier AES et tolérer 3DES avec rafraîchissement des clés | 23 |
| R9 | Construire le HMAC avec SHA-2 | 23 |
| R9- | Privilégier les HMAC avec SHA-2 et tolérer les HMAC avec SHA-1 | 24 |
| R10 | Utiliser un mode de chiffrement robuste | 24 |
| R10- | Utiliser le mode CBC sans <code>encrypt_then_mac</code> | 24 |
| R11 | Disposer de plusieurs suites cryptographiques | 26 |
| R12 | Préférer l'ordre de suites du serveur | 26 |
| R13 | Utiliser les extensions du tableau 2.3 | 31 |
| R14 | Évaluer l'utilité des extensions du tableau 2.4 | 32 |
| R15 | Ne pas utiliser les extensions du tableau 2.5 | 32 |
| R16 | Utiliser un générateur d'aléa robuste | 33 |
| R17 | Privilégier les aléas sans préfixe prédictible | 33 |
| R18 | Ne pas utiliser la compression TLS | 34 |
| R19 | Ne pas effectuer de reprise de session | 34 |
| R19- | Effectuer des reprises de session avec PFS | 35 |
| R19-- | Effectuer des reprises de session sans PFS | 35 |
| R20 | Sécuriser les renégociations | 35 |

| | | |
|------|---|----|
| R21 | Présenter un certificat signé avec SHA-2 | 37 |
| R22 | Présenter un certificat valide pendant 3 ans ou moins | 38 |
| R23 | Utiliser des clés de taille suffisante | 38 |
| R24 | Présenter un <i>KeyUsage</i> approprié | 39 |
| R25 | Présenter un <i>ExtendedKeyUsage</i> approprié | 39 |
| R26 | Présenter un <i>SubjectAlternativeName</i> approprié (côté serveur) | 40 |
| R27 | Réserver chaque certificat à une seule terminaison TLS | 40 |
| R28 | Présenter un AKI correspondant au SKI défini par l'AC | 41 |
| R29 | Présenter un certificat avec des sources de révocation | 41 |
| R30 | Transmettre une chaîne de certificats ordonnée et complète | 41 |
| R31 | Préférer l'agrafage OCSP | 42 |
| R32 | Prévoir une redondance des moyens de révocation | 42 |
| R33 | Réagir en <i>hard-fail</i> | 43 |
| R34 | Utiliser des certificats enregistrés par CT | 43 |
| R35 | Rejeter tous les certificats invalidés par CT | 43 |
| R35- | Rejeter les certificats EV invalidés par CT | 43 |

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. Cohnen, 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 (Proposed Standard), jan. 1999. Obsoleted by RFC 4346, updated by RFCs 3546, 5746, 6176, 7465, 7507.
- [9] T. Dierks et E. Rescorla, « The Transport Layer Security (TLS) Protocol Version 1.1 ». RFC 4346 (Proposed Standard), avril 2006. Obsoleted by RFC 5246, updated by RFCs 4366, 4680, 4681, 5746, 6176, 7465, 7507.
- [10] T. Dierks et E. Rescorla, « The Transport Layer Security (TLS) Protocol Version 1.2 ». RFC 5246 (Proposed Standard), août 2008. Updated by RFCs 5746, 5878, 6176, 7465, 7507, 7568, 7627, 7685, 7905.

- [11] R. Barnes, M. Thomson, A. Pironti et A. Langley, « Deprecating Secure Sockets Layer Version 3.0 ». RFC 7568 (Proposed Standard), juin 2015.
- [12] R. Braden, « Requirements for Internet Hosts - Communication Layers ». RFC 1122 (INTERNET STANDARD), oct. 1989. Updated by RFCs 1349, 4379, 5884, 6093, 6298, 6633, 6864.
- [13] E. Rescorla et N. Modadugu, « Datagram Transport Layer Security Version 1.2 ». RFC 6347 (Proposed Standard), jan. 2012. Updated by RFCs 7507, 7905.
- [14] E. Rescorla, « Diffie-Hellman Key Agreement Method ». RFC 2631 (Proposed Standard), juin 1999.
- [15] D. E. 3rd, « Transport Layer Security (TLS) Extensions : Extension Definitions ». RFC 6066 (Proposed Standard), jan. 2011.
- [16] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley et W. Polk, « Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile ». RFC 5280 (Proposed Standard), mai 2008. Updated by RFC 6818.
- [17] Mozilla, « Network Security Services ». <<https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS>>.
- [18] Agence nationale de la sécurité des systèmes d'information (ANSSI), « Référentiel Général de Sécurité - Annexe A4 ». <http://www.ssi.gouv.fr/uploads/2014/11/RGS_v-2-0_A4.pdf>.
- [19] Google, « CRLSets ». <<https://dev.chromium.org/Home/chromium-security/crlsets>>.
- [20] Mozilla, « Revoking Intermediate Certificates : Introducing OneCRL ». <<https://blog.mozilla.org/security/2015/03/03/revoking-intermediate-certificates-introducing-onecrl/>>, March 2015.
- [21] B. Laurie, A. Langley et E. Kasper, « Certificate Transparency ». RFC 6962 (Experimental), juin 2013.
- [22] Common Vulnerabilities and Exposures, « CVE-2009-3555 ». <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3555>>, August 2009.
- [23] J. Rizzo et T. Duong, « Browser Exploit Against SSL/TLS ». <<https://packetstormsecurity.com/files/105499/Browser-Exploit-Against-SSL-TLS.html>>, October 2011.
- [24] Common Vulnerabilities and Exposures, « CVE-2014-0160 ». <<https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2014-0160>>, April 2014.

- 
- [25] E. Rescorla, « The Transport Layer Security (TLS) Protocol Version 1.3 – Version 12 ». <<https://tools.ietf.org/html/draft-ietf-tls-tls13-12>>, March 2016.
- [26] B. Moeller et A. Langley, « TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks ». RFC 7507 (Proposed Standard), avril 2015.
- [27] A. Langley, W. Chang, N. Mavrogiannopoulos, J. Strombergson et S. Josefsson, « ChaCha20-Poly1305 Cipher Suites for Transport Layer Security (TLS) ». RFC 7905 (Proposed Standard), juin 2016.
- [28] Agence nationale de la sécurité des systèmes d'information (ANSSI), « Référentiel Général de Sécurité - Annexe B1 ». <http://www.ssi.gouv.fr/uploads/2014/11/RGS_v-2-0_B1.pdf>.
- [29] Internet Assigned Numbers Authority, « Transport Layer Security (TLS) Parameters ». <<https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml>>.
- [30] P. Eronen et H. Tschofenig, « Pre-Shared Key Ciphersuites for Transport Layer Security (TLS) ». RFC 4279 (Proposed Standard), déc. 2005.
- [31] D. Taylor, T. Wu, N. Mavrogiannopoulos et T. Perrin, « Using the Secure Remote Password (SRP) Protocol for TLS Authentication ». RFC 5054 (Informational), nov. 2007.
- [32] N. J. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering et J. C. Schuldt, « On the Security of RC4 in TLS and WPA ». <<https://cr.yp.to/streamciphers/rc4biases-20130708.pdf>>, July 2013.
- [33] A. Popov, « Prohibiting RC4 Cipher Suites ». RFC 7465 (Proposed Standard), fév. 2015.
- [34] S. Kelly, « Security Implications of Using the Data Encryption Standard (DES) ». RFC 4772 (Informational), déc. 2006.
- [35] H. Y. Xiaoyun Wang, « Advances in cryptology – crypto 2005 : 25th annual international cryptology conference, santa barbara, california, usa, august 14-18, 2005. proceedings », 2005.
- [36] M. Stevens, P. Karpman et T. Peyrin, « Freestart collision for full SHA-1 ». <<https://eprint.iacr.org/2015/967>>, 2016.
- [37] M. J. Dworkin, « SHA-3 Standard : Permutation-Based Hash and Extendable-Output Functions ». <https://http://www.nist.gov/manuscript-publication-search.cfm?pub_id=919061>, August 2015.

- [38] N. J. AlFardan 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.
- [39] Adam Langley, « Lucky Thirteen attack on TLS CBC ». <<https://www.imperialviolet.org/2013/04/luckythirteen.html>>, February 2013.
- [40] B. Möller, T. Duong et K. Kotowicz, « This POODLE bites : Exploiting the SSL 3.0 Fallback », rap. tech., September 2014.
- [41] Internet Assigned Numbers Authority, « Transport Layer Security (TLS) Extensions ». <<https://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xhtml>>.
- [42] S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk et B. Moeller, « Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) ». RFC 4492 (Informational), mai 2006. Updated by RFCs 5246, 7027.
- [43] P. Gutmann, « Encrypt-then-MAC for Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) ». RFC 7366 (Proposed Standard), sept. 2014.
- [44] K. Bhargavan, A. Delignat-Lavaud, A. Pironti, A. Langley et M. Ray, « Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension ». RFC 7627 (Proposed Standard), sept. 2015.
- [45] E. Rescorla, M. Ray, S. Dispensa et N. Oskov, « Transport Layer Security (TLS) Renegotiation Indication Extension ». RFC 5746 (Proposed Standard), fév. 2010.
- [46] S. Santesson, A. Medvinsky et J. Ball, « TLS User Mapping Extension ». RFC 4681 (Proposed Standard), oct. 2006.
- [47] N. Mavrogiannopoulos et D. Gillmor, « Using OpenPGP Keys for Transport Layer Security (TLS) Authentication ». RFC 6091 (Informational), fév. 2011.
- [48] J. Callas, L. Donnerhacke, H. Finney, D. Shaw et R. Thayer, « OpenPGP Message Format ». RFC 4880 (Proposed Standard), nov. 2007. Updated by RFC 5581.
- [49] D. McGrew et E. Rescorla, « Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP) ». RFC 5764 (Proposed Standard), mai 2010.
- [50] H. Schulzrinne, S. Casner, R. Frederick et V. Jacobson, « RTP : A Transport Protocol for Real-Time Applications ». RFC 3550 (INTERNET STANDARD), juil. 2003. Updated by RFCs 5506, 5761, 6051, 6222, 7022, 7160, 7164.
- [51] S. Friedl, A. Popov, A. Langley et E. Stephan, « Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension ». RFC 7301 (Proposed Standard), juil. 2014.

- 
- [52] Y. Pettersen, « The Transport Layer Security (TLS) Multiple Certificate Status Request Extension ». RFC 6961 (Proposed Standard), juin 2013.
 - [53] A. Langley, « A Transport Layer Security (TLS) ClientHello Padding Extension ». RFC 7685 (Proposed Standard), oct. 2015.
 - [54] J. Salowey, H. Zhou, P. Eronen et H. Tschofenig, « Transport Layer Security (TLS) Session Resumption without Server-Side State ». RFC 5077 (Proposed Standard), jan. 2008.
 - [55] M. Brown et R. Housley, « Transport Layer Security (TLS) Authorization Extensions ». RFC 5878 (Experimental), mai 2010.
 - [56] R. Seggelmann, M. Tuexen et M. Williams, « Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension ». RFC 6520 (Proposed Standard), fév. 2012.
 - [57] 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 (Proposed Standard), juin 2014.
 - [58] S. Hollenbeck, « Transport Layer Security Protocol Compression Methods ». RFC 3749 (Proposed Standard), mai 2004.
 - [59] J. Rizzo et T. Duong, « The CRIME attack ». <http://www.ekoparty.org/archive/2012/CRIME_ekoparty2012.pdf>, 2012.
 - [60] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Pironti et P.-Y. Strub, « Triple Handshakes and Cookie Cutters : Breaking and Fixing Authentication over TLS ». <<https://www.mitls.org/downloads/tlsauth.pdf>>, May 2014.
 - [61] M. Rex, « MITM attack on delayed TLS-client auth through renegotiation ». <<https://www.ietf.org/mail-archive/web/tls/current/msg03928.html>>, November 2009.
 - [62] Agence nationale de la sécurité des systèmes d'information (ANSSI), « Référentiel Général de Sécurité - Annexe B2 ». <http://www.ssi.gouv.fr/uploads/2014/11/RGS_v-2-0_B2.pdf>.
 - [63] M. Prince, « The Hidden Costs of Heartbleed ». <<https://blog.cloudflare.com/the-hard-costs-of-heartbleed/>>, April 2014.
 - [64] The Apache Software Foundation, « Apache Module mod_ssl ». <https://httpd.apache.org/docs/2.4/en/ssl/ssl_howto.html>.
 - [65] Nginx, « Module ngx_http_ssl_module ». <http://nginx.org/en/security_advisories.html>.

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 |
| DES | Data Encryption Standard |
| DH | Diffie–Hellman |
| DHE | Diffie–Hellman Ephemeral |
| DTLS | Datagram Transport Layer Security |
| ECC | Elliptic Curve Cryptography |
| ECDH | Elliptic Curve Diffie–Hellman |
| ECDHE | Elliptic Curve Diffie–Hellman Ephemeral |
| EV | Extended Validation |
| FFDH | Finite Field Diffie–Hellman |
| FQDN | Fully Qualified Domain Name |
| HTTP | Hypertext Transfer Protocol |
| HTTPS | HTTP Secure |
| IANA | Internet Assigned Numbers Authority |
| IETF | Internet Engineering Task Force |
| 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 |
| SCSV | Signaling Cipher Suite Value |
| SCT | Signed Certificate Timestamp |
| SKI | Subject Key Identifier |
| SNI | Server Name Indication |
| SRP | Secure Remote Password |
| SSL | Secure Sockets Layer |
| TCP | Transmission Control Protocol |
| TLS | Transport Layer Security |
| UDP | User Datagram Protocol |
| 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 2016

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