HTTPS : de SSL à TLS 1.3

[Openweb.eu.org](https://openweb.eu.org/) > [Articles](https://openweb.eu.org/articles/)  > HTTPS : de SSL à TLS 1.3

Abstract

Un changement de numéro de version, même mineur, est rarement anodin lorsqu’il s’agit d’un protocole de sécurité. Alors qu’une importante partie de l’écosystème qui gravite autour de SSL/TLS persiste à l’appeler SSL, aujourd’hui, c’est bel et bien uniquement TLS qu’il faut utiliser et avec une large préférence pour TLS 1.2 en attendant TLS 1.3 prochainement.

Article

HTTPS est apparu dans [Netscape Navigator](https://fr.wikipedia.org/wiki/Netscape_Navigator) en 1995 pour accompagner l’essor du commerce en ligne alors naissant. Initialement c’est le protocole SSL —*Secure Sockets Layer*— qui était intercalé entre TCP et HTTP. Depuis lors, SSL a bien évolué et a même changé de nom pour devenir TLS —*Transport Layer Security*—, on dénombre désormais six versions : SSLv2, SSLv3, TLS 1.0, TLS 1.1, TLS 1.2 et TLS 1.3. Elles apportent chacune des améliorations parfois subtiles mais qui peuvent se révéler cruciales. 2017 sera marquée par l’avènement de TLS 1.3. Neuf ans après TLS 1.2, quatre ans après les révélations d’Edward Snowden, cette nouvelle version apporte plus de simplicité, de vitesse et de sécurité.

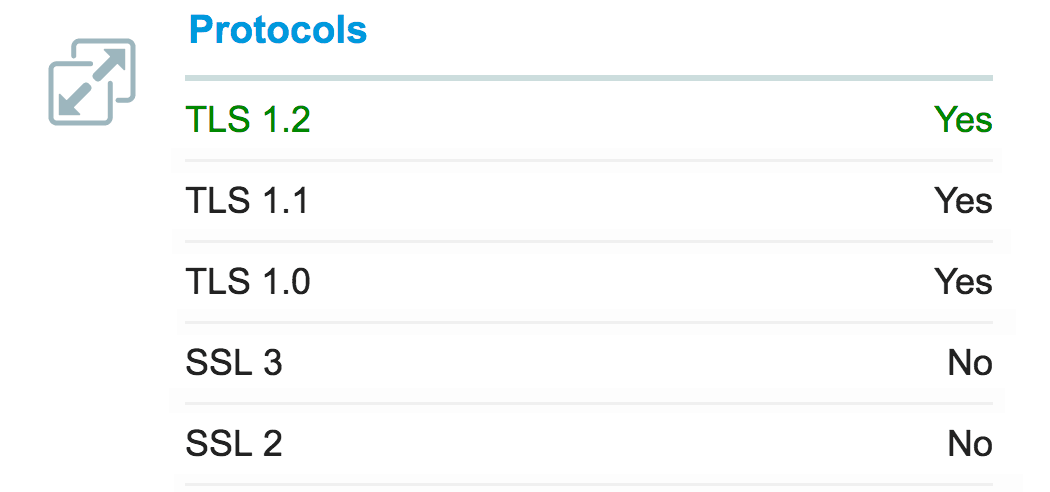
Cet article est relativement long car il aborde des aspects historiques en plus de la technique, mais d’un point de vue purement pratique il peut se résumer en quelques points :

* TLS 1.2 est une nécessité, à activer si ce n’est déjà fait ;
* SSLv2 et SSLv3 sont des calamités, à désactiver ;
* TLS 1.0 est en sursis, planifier sa désactivation ;
* TLS 1.1 ne sert pratiquement à rien ;
* TLS 1.3 est l’avenir, considérer son adoption une fois disponible.

Ces recommandations sont en phase avec celles de l’ANSSI —Agence Nationale de la Sécurité des Systèmes d’Information— : [Recommandations de sécurité relatives à TLS](https://www.ssi.gouv.fr/guide/recommandations-de-securite-relatives-a-tls/) (si ce n’est que l’agence française n’aborde pas encore TLS 1.3) ou le wiki de Mozilla concernant la configuration TLS : [*Security/Server Side TLS*](https://wiki.mozilla.org/Security/Server_Side_TLS) (destiné à la base au seul usage interne au sein de Mozilla, mais le guide [Sécuriser votre site à l’aide du protocole HTTPS](https://support.google.com/webmasters/answer/6073543?hl=fr) de Google et le [*National Cyber Security Centre*](https://www.ncsc.gov.uk/guidance/tls-external-facing-services) britannique le recommandent).

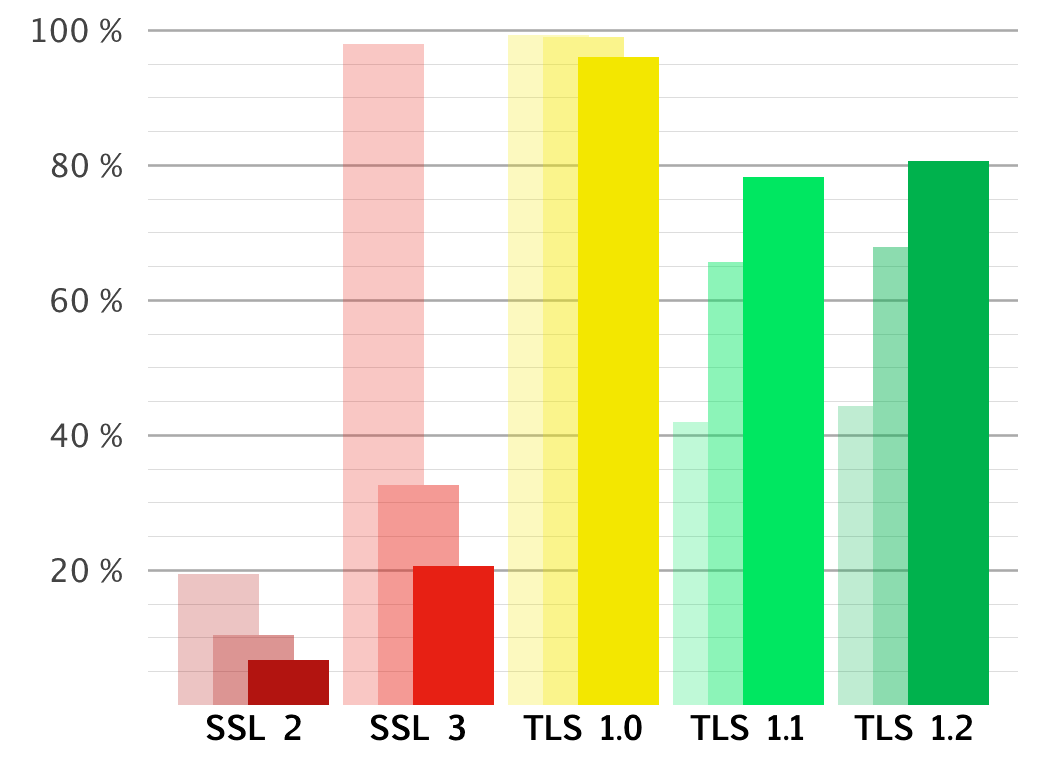
Il arrive qu’on ne dispose que d’un contrôle partiel sur le paramétrage de TLS qui est souvent lié à un composant proche du système d’exploitation comme [OpenSSL](https://www.openssl.org/) ou [*Secure Channel*](https://msdn.microsoft.com/en-us/library/windows/desktop/aa380123(v=vs.85).aspx) [[1](https://openweb.eu.org/articles/https-de-ssl-a-tls-1-3#nb1)], il faut donc éventuellement s’adresser à son administrateur système ou son hébergeur pour parvenir à mettre en place certaines évolutions.

Il est en revanche assez facile de s’équiper d’outils pour analyser la configuration SSL/TLS d’un serveur ou d’un client web. [*SSL Labs*](https://www.ssllabs.com/ssltest/) (en ligne) et son quasi équivalent pour *shell bash* [testssl.sh](https://github.com/drwetter/testssl.sh) génèrent un rapport relativement complet sur la configuration d’un serveur web sécurisé quelconque et indiquent donc quelles versions de SSL/TLS sont actives, on trouve par exemple pour [un domaine de la DGFiP](https://www.ssllabs.com/ssltest/analyze.html?d=cfspart.impots.gouv.fr).



Avoir TLS 1.0, 1.1 et 1.2 comme seules versions actives est une situation très courante actuellement. En effet, tous les navigateurs sont dotés de TLS 1.2 depuis plusieurs années (voir [Can I use : TLS 1.2](http://caniuse.com/#feat=tls1-2)), mais le parc mondial n’est malheureusement pas constitué uniquement de navigateurs récents, c’est pourquoi TLS 1.0 reste encore souvent activé aux côtés de TLS 1.2 et 1.1 sur de nombreux serveurs. Quant à TLS 1.3 il faut encore attendre sa finalisation puis la diffusion de bibliothèques adaptées.

On trouve sur [*SSL Pulse*](https://www.trustworthyinternet.org/ssl-pulse/) un ensemble de statistiques relatives aux éléments clés de la configuration SSL/TLS côté serveur, celles-ci proviennent de l’analyse mensuelle d’un large panel de sites web. Ce qui permet de prendre régulièrement le pouls de l’état de la crypto sur le web et d’éventuellement se situer par rapport aux autres sites.  
Le graphique ci-dessous représente l’évolution des versions de SSL/TLS acceptées par les serveurs scrutés par *SSL Pulse* sur une période de deux ans (les colonnes représentent de gauche à droite octobre 2014, 2015 et 2016).

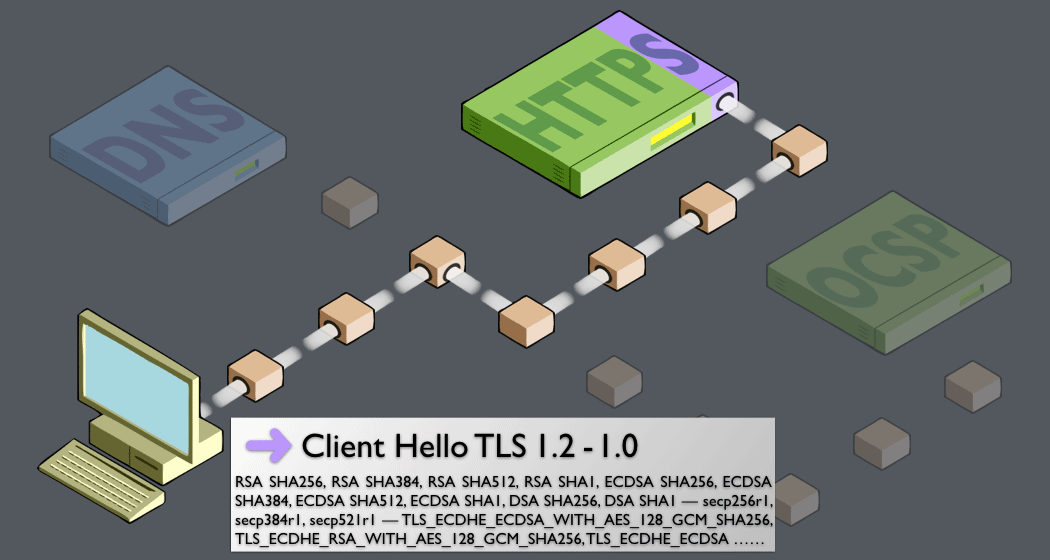


On observe la disparition (toute relative) de SSLv2 et v3, un début de retrait de TLS 1.0 et pour finir l’essor rapide (mais tardif) de TLS 1.1 et 1.2. En se basant sur les tendances on peut déduire que les sites en pointe acceptent TLS 1.2 et 1.1 mais refusent d’ores et déjà TLS 1.0, que le gros des troupes accepte indifféremment TLS 1.2, 1.1 et 1.0 et que les sites à la traîne ne peuvent traiter que TLS 1.0 parfois encore secondé par SSLv3. Ce découpage en trois populations correspond grosso modo à ce que propose Mozilla sur [*Security/Server Side TLS*](https://wiki.mozilla.org/Security/Server_Side_TLS) avec ses trois configurations : moderne, intermédiaire et vieillot ou la classification R3, R3− et R3−− de l’ANSSI.

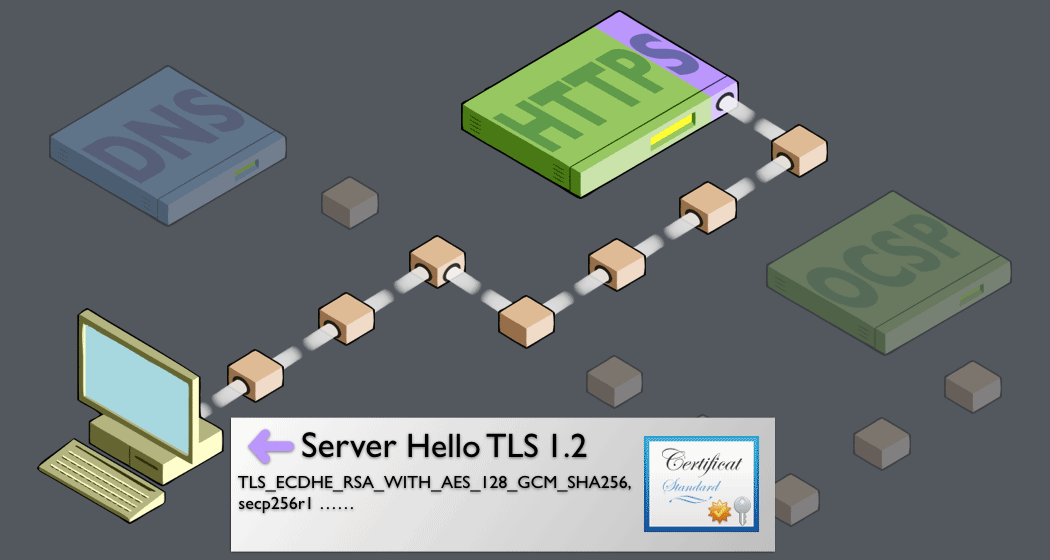
Il faut en effet faire preuve de pragmatisme, la même configuration SSL/TLS ne peut pas convenir à tout le monde. De façon un peu caricaturale, un site orienté *B2B* sera certainement bien plus confronté à de vieux Internet Explorer pour qui TLS 1.0 reste nécessaire, alors qu’un site dont la confidentialité est une priorité ou doté d’un design qui repose sur des fonctionnalités CSS ou Flexbox récentes pourrait, au contraire, se passer de TLS 1.0 et des vieux clients web qui en dépendent. Une refonte peut être une occasion intéressante de réviser une configuration TLS, à condition de bien en mesurer les conséquences —ce qui implique que le front-end discute avec le back-end et le système. Si un site est prévu pour au minimum IE 11 et que son niveau de dégradation avec IE 9 ou plus ancien est tel qu’il vaudrait mieux ne pas l’afficher… la configuration TLS offre une solution radicale.

**Le client web propose, le serveur web dispose**

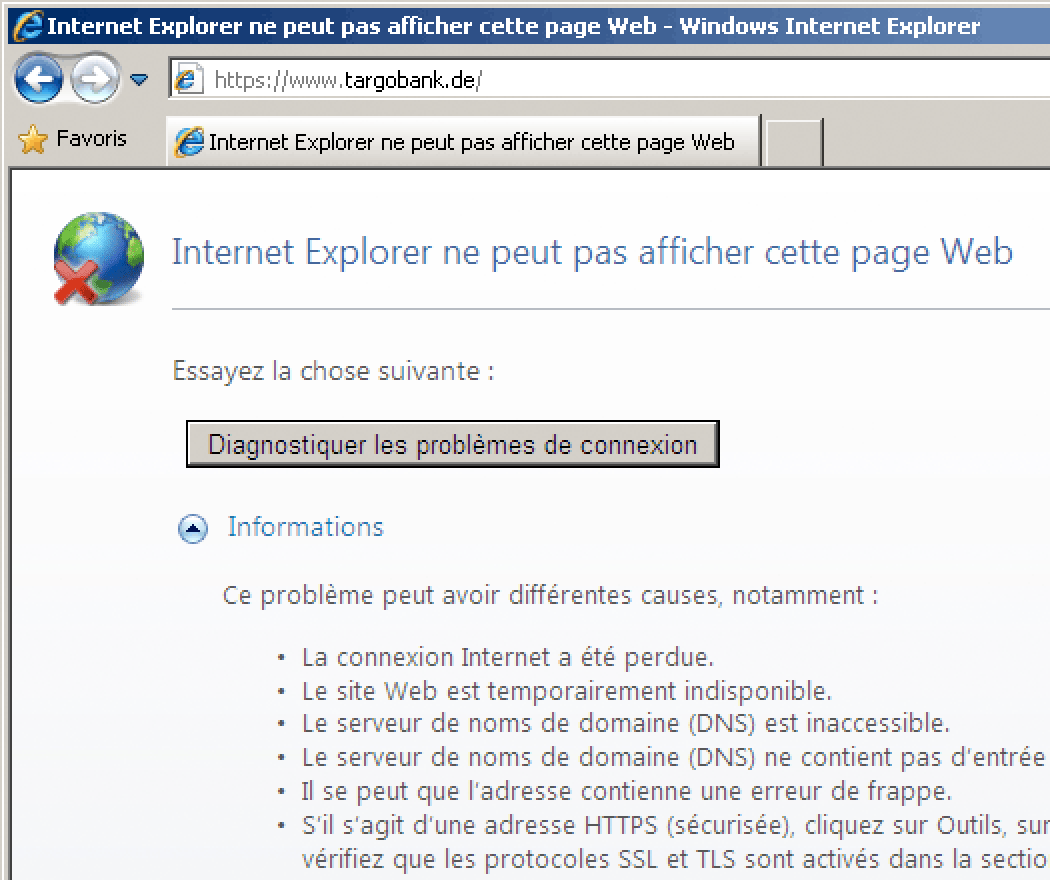
Le client et le serveur web négocient certains éléments lors de l’établissement de la liaison sécurisée. Il faut en effet qu’ils s’assurent de disposer d’algorithmes en commun pour signer, échanger des clés et chiffrer, mais la version du protocole SSL/TLS à utiliser est le tout premier point sur lequel ils doivent nécessairement s’entendre, car le reste de l’établissement de la liaison sécurisée (le *handshake*) varie légèrement d’une version à l’autre, ce sont surtout des initialisations, transmissions d’aléas et calculs de sommes de contrôle qui changent.



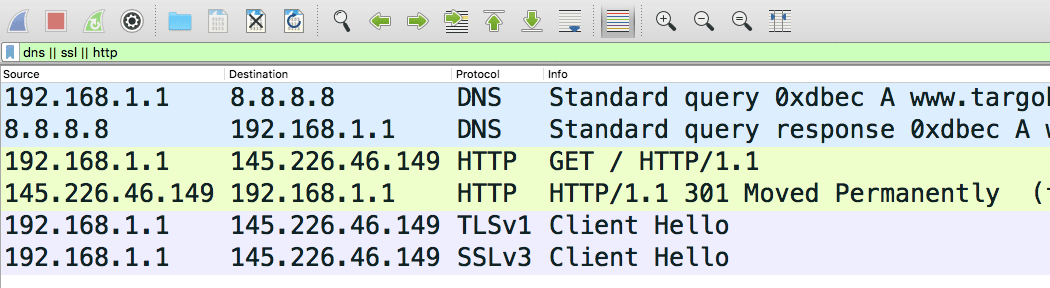
La règle de base est que le client propose, il va donc faire étalage de toutes ses capacités, parfois dans un ordre préférentiel. Puis c’est le serveur qui dispose vu que c’est lui qui va retenir quels algorithmes seront utilisés en fonction de ses propres capacités.



Il va sans dire qu’il se produit quelques problèmes intergénérationnels lorsque de vieux clients essayent de communiquer avec des serveurs qui cherchent à renforcer la sécurité des données en transit, car ceci s’oppose à une rétrocompatibilité très poussée, ou inversement lorsque des clients récents croisent un serveur resté figé à une autre époque.



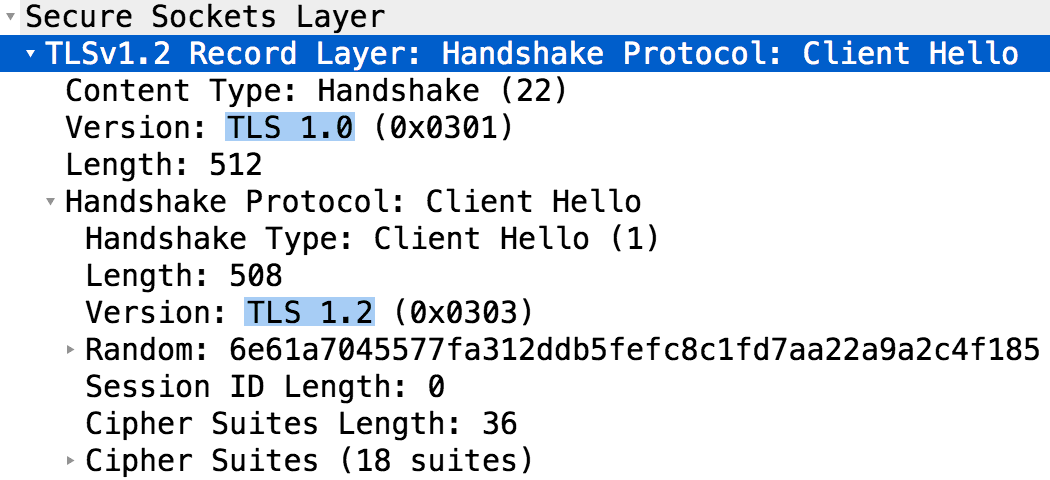
IE 8 sur Windows XP n’arrive pas à négocier l’ouverture de la connexion sécurisée avec un site qui nécessite au minimum TLS 1.1.



Les traces réseau révèlent que le navigateur a fait deux tentatives, une fois en TLS 1.0 et une fois en SSLv3, le serveur n’a pas même daigné lui répondre, le navigateur se retrouve alors déboussolé et ne peut afficher qu’un message d’erreur assez générique. Ce point est à prendre en compte lorsque l’on planifie de désactiver TLS 1.0 sur un site existant, il faut en effet prévenir les derniers utilisateurs qui se présentent encore avec de vieux clients web du changement à venir bien avant la désactivation effective.

**Le client web propose, le serveur web… explose**

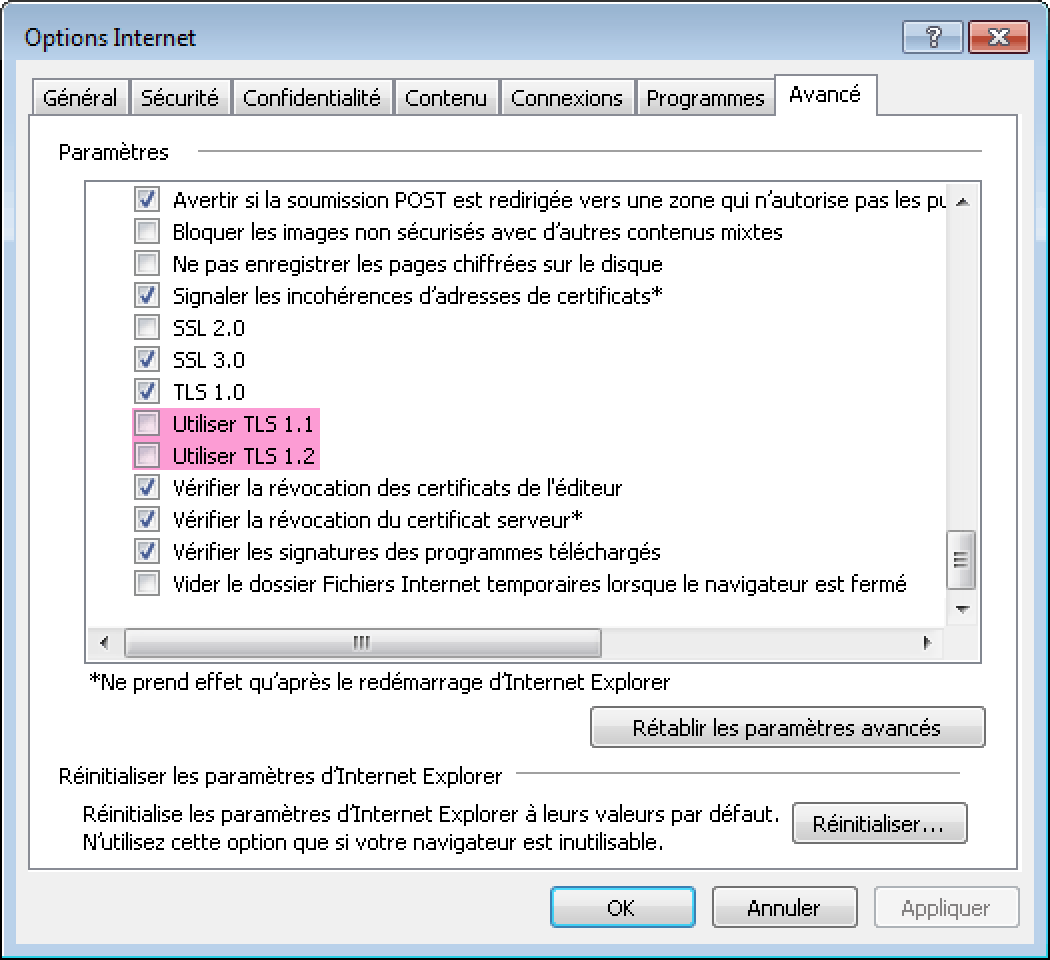
Dans le « *Client Hello* », qui correspond au premier message adressé par le client au serveur, figurent la version la plus élevée et la plus basse du protocole que le client propose d’utiliser, de nos jours on y trouve très souvent TLS 1.2 et TLS 1.0, mais ceci n’est qu’une convention qui laisse d’ailleurs planer un doute quant au fait que TLS 1.1 puisse être utilisé ou non.



Pour des raisons historiques et de nombreuses mises en œuvres bancales, cet aspect est un brin moins trivial qu’il ne devrait l’être. En effet lorsque les premiers clients aptes à utiliser TLS 1.0 ont entamé le dialogue avec des serveurs qui ne connaissaient que SSLv3, les négociations ne se sont pas déroulées exactement comme prévues… En théorie les serveurs en question auraient dû décliner la proposition d’utiliser TLS 1.0 et se rabattre sur SSLv3 à la place, en pratique certains serveurs se sont figés d’effroi et dans le mutisme le plus total.

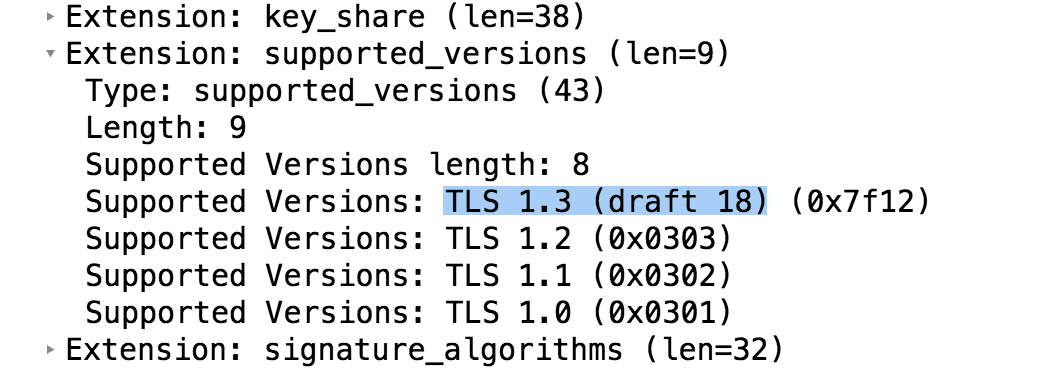
Ce phénomène porte le nom d’intolérance de version et a forcé les clients web à émettre une première tentative de négociation pour TLS 1.0 puis, en cas d’échec ou en l’absence de réponse, une seconde tentative pour SSLv3, exactement comme on le voit sur les traces réseau d’IE 8 ci-dessus.

Sans grande surprise le même phénomène, un peu atténué, s’est produit lorsque les premiers clients TLS 1.1/1.2 se sont adressés à des serveurs qui ne connaissaient que TLS 1.0 et SSLv3. Certains sites ont alors conseillé à leurs utilisateurs de désactiver TLS 1.2 et TLS 1.1 dans les « Options Internet » de Windows, les privant donc de ces versions sur tous les autres sites !



Si les deux cases « Utiliser TLS 1.2 » et « Utiliser TLS 1.1 » sont décochées Internet Explorer devient incapable de négocier les versions de TLS les plus récentes. Du coup il n’est pas tout à fait possible de faire le lien entre un *User Agent* et le support de TLS 1.2 vu que celui-ci peut être désactivé par ailleurs, ce qui implique de se pencher directement sur les logs de négociations TLS pour, par exemple, déterminer la part exacte des clients qui nécessitent TLS 1.0 sur un site donné.

La bonne nouvelle est que ce problème d’intolérance ne se produira pas avec TLS 1.3. En effet, lors de la conception de ce dernier, il a été évalué qu’environ 2% des serveurs web sécurisés seraient intolérants aux clients qui chercheraient à négocier TLS 1.3 d’entrée de jeu. De ce fait, les clients TLS 1.3 se présentent en définitive comme de simples clients TLS 1.2 mais arborent en plus une nouvelle extension qui permet une énumération claire des versions du protocole que le client souhaite utiliser. Les extensions inconnues étant proprement ignorées par les serveurs web, il ne devrait plus y avoir de problème d’intolérance.



Firefox 51.0.1 indique clairement qu’il souhaite utiliser TLS 1.3 (sur la base du *draft* 18).

**Des temps anciens à l’ère post NIST**

Ces dernières années la majorité des failles de sécurité ou faiblesses rendues publiques comme *BEAST, POODLE, DROWN, SWEET32…* concernent les versions les plus anciennes du protocole SSL/TLS. En compilant sur plus de 20 ans les différentes évolutions de ce protocole, on comprend mieux pourquoi TLS 1.3 devenait nécessaire et représente une étape significative vers un Internet plus sûr.

Un des objectifs de cette partie est de montrer que ce qui définit SSL/TLS n’est pas une spécification monolithique unique mais plutôt un patchwork de RFC apparues progressivement et dans lesquelles les différentes mises en œuvres logicielles vont piocher plus ou moins d’éléments ; les explications détaillées concernant certains concepts (auxquels sont souvent associé des acronymes : CBC, GCM, ECDHE…) ne seront fournies que dans les articles relatifs aux certificats et aux suites cryptographiques.

**SSLv2 (1995)**

SSL version 2 constitue la version initiale proposée par Netscape, la version 1 n’ayant jamais servi. SSLv2 présente des lacunes importantes et ne devrait plus être utilisé depuis très longtemps, très très longtemps même, l’IETF s’est clairement prononcé sur ce sujet en 2011 dans la [RFC 6176](https://tools.ietf.org/html/rfc6176). Toutefois, au début 2016, l’attaque [*DROWN*](https://drownattack.com/) a mis en lumière sa présence sur de nombreux serveurs HTTPS et SMTP alors que les clients web pouvant se connecter de la sorte ont disparu depuis belle lurette, et plus aucune autorité de certification ne peut émettre de « certificat SSL » avec des caractéristiques adaptées aux clients web qui ne connaîtraient que SSLv2. Ceci est malheureusement symptomatique de négligences de sécurité élémentaires : défaut de mise à jour et reconduction de paramètres par méconnaissance.

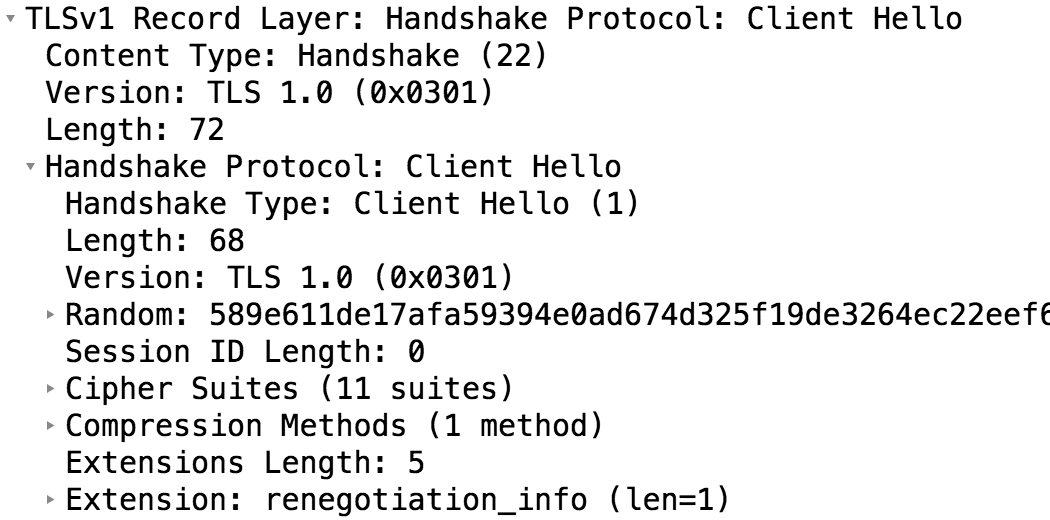
**SSLv3 (1996,** [**RFC 6101**](https://tools.ietf.org/html/rfc6101)**)**

Conscient des faiblesses de SSLv2, Netscape propose SSLv3 dès l’année suivante. Cette version est restée active et très largement utilisée pendant près de 18 ans (le long règne d’IE 6 et la prédominance de Windows XP y sont pour quelque chose). En septembre 2014, l’attaque [*POODLE*](https://www.openssl.org/~bodo/ssl-poodle.pdf) scelle le sort de SSLv3 car la seule contre-mesure possible est sa désactivation au profit de TLS. Mais cette version était déjà décriée depuis des années comme en témoigne les tutoriels de [désactivation de SSLv3](http://disablessl3.com/), la [RFC 7568](https://tools.ietf.org/html/rfc7568) de 2015 en interdit l’usage et préconise TLS 1.2 à la place. Il est à noter qu’en raison de *POODLE*, la majorité des navigateurs ont définitivement désactivé SSLv3 vers la fin 2014.

À moins de vouloir à tout prix être compatible avec Internet Explorer sur Windows XP Service Pack 2 (le SP3 apporte TLS 1.0 qu’il faut toutefois activer manuellement dans « Options Internet ») il n’y a strictement aucune raison d’activer SSLv3 côté serveur.

**TLS 1.0 (1999,** [**RFC 2246**](https://tools.ietf.org/html/rfc2246)**)**

Pour concilier les divergences entre Microsoft et Netscape concernant l’évolution de SSL, le protocole est confié à l’IETF —*Internet Engineering Task Force*— et renommé TLS —*Transport Layer Security*— au passage. À cette époque, TLS 1.0 ne constituait qu’une évolution assez légère de SSLv3 (en interne l’identification de version est d’ailleurs 3.1). TLS révise cependant une partie de l’établissement de la liaison (le *handshake*) et apporte de la flexibilité avec l’introduction d’extensions optionnelles qui permettent de faire évoluer le protocole sans revoir ses bases et constamment réviser son numéro de version.



Les premières années l’utilisation des extensions est restée très limitée, le *Client Hello* produit par IE sur Windows XP ne dépassera jamais cette forme rudimentaire.

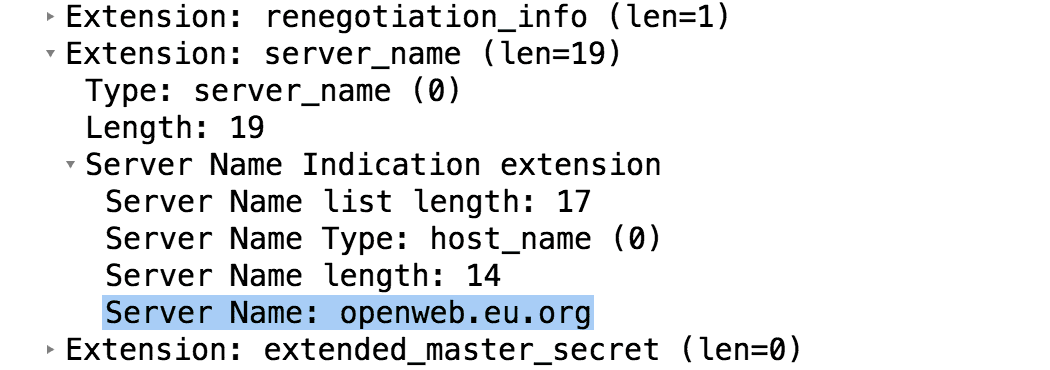
**AES —*Advanced Encryption Standard*— (2002,** [**RFC 3268**](https://tools.ietf.org/html/rfc3268)**)**

Le chiffrement symétrique AES a été intégré dans TLS quelques années après la parution de TLS 1.0, il est aujourd’hui très largement répandu et restera vraisemblablement indispensable encore de nombreuses années.

Parmi les améliorations apportées à Secure Channel dans Windows Vista par rapport à Windows XP figurent la mise en œuvre de suites cryptographiques qui proposent le chiffrement AES et dans un autre domaine le SNI…

**SNI —*Server Name Indication*— (2003,** [**RFC 3546**](https://tools.ietf.org/html/rfc3546)**)**

SNI est une extension TLS qui revêt désormais une importance toute particulière ! En effet, précédemment SSL partait du principe qu’un seul domaine était hébergé par adresse IP et à aucun moment le protocole ne faisait mention du domaine visé… Un peu à l’image de HTTP/1.1 qui a permis l’hébergement virtuel avec l’en-tête « *host* », TLS donne la possibilité d’indiquer le nom du serveur via l’extension SNI, ce qui permet donc d’héberger plusieurs sites sécurisés sur une même adresse IP sans avoir à mutualiser le certificat et la clé privée.



Les clients web incapables de gérer SNI sont en voie de disparition (voir [Can I use : SNI](http://caniuse.com/#feat=sni)) et les hébergeurs ont donc désormais massivement recours à SNI pour offrir HTTPS à tout le monde.

Aujourd’hui TLS 1.0 montre de sérieux signes de fatigue, car les suites cryptographiques qui y sont communément associées reposent sur des méthodes de chiffrement dont la cryptanalyse révèle désormais les faiblesses, on notera :

* l’interdiction du chiffrement RC4 dans la [RFC 7465](https://tools.ietf.org/html/rfc7465) voir également [*RC4 NOMORE*](http://www.rc4nomore.com/) ;
* un problème de chaînage et d’initialisation des blocs CBC [*BEAST Attack*](http://commandlinefanatic.com/cgi-bin/showarticle.cgi?article=art027) ;
* ou encore les blocs de 64 bits qui fragilisent le Triple DES [*SWEET32*](https://sweet32.info/).

De ce fait, configurer TLS 1.0 correctement est un exercice de plus en plus périlleux, au point que le PCI SSC —*Payment Card Industry Security Standards Council*— (Visa, Mastercard et les autres réseaux de cartes) cherche à bannir cette version lors des paiements CB en ligne et par là même des sites d’e-commerce (PCI DSS 3.2 fixe une date butoir à la [mi-2018](https://blog.pcisecuritystandards.org/migrating-from-ssl-and-early-tls), le PCI SSC a récemment publié [*Best Practices for Securing E-commerce*](https://www.pcisecuritystandards.org/pdfs/best_practices_securing_ecommerce.pdf) qui constitue une lecture intéressante). Pour satisfaire aux exigences de PCI DSS, certains hébergeurs proposent en option des configurations uniquement basées sur TLS 1.2.

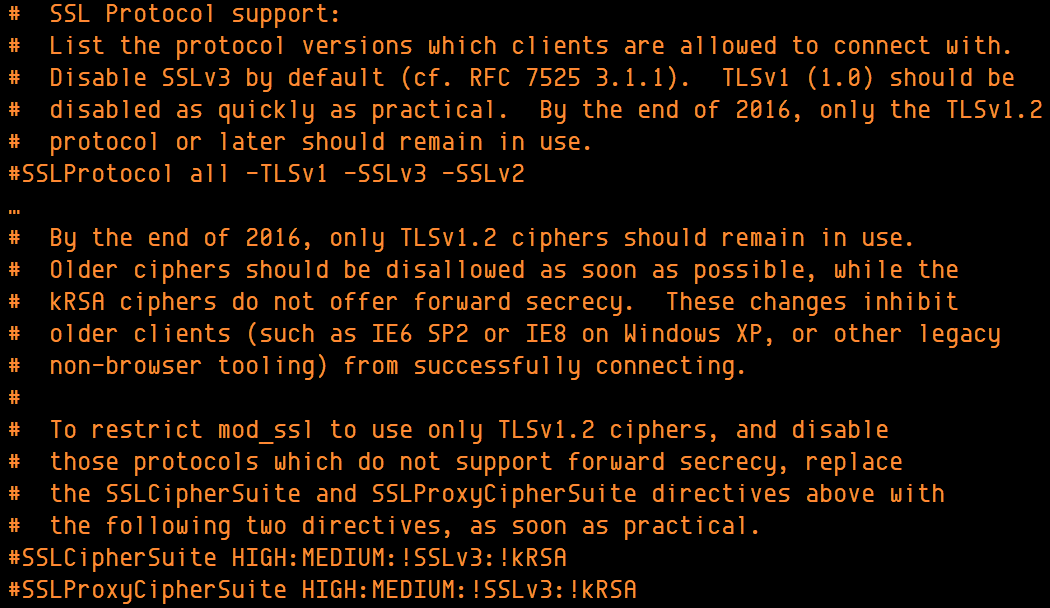
Désactiver TLS 1.0 revient à tourner le dos à environ 5 à 10 % des navigateurs utilisés et plus particulièrement IE 8 et antérieurs sur Windows XP et IE 9 et antérieurs sur Windows Vista, les versions de Safari antérieures à la 7 sur OS X ainsi que le navigateur intégré (pas Chrome) des appareils Android antérieurs à 5.0.

Firefox est devenu l’ultime planche de salut sur de vieux systèmes comme XP et Vista où contrairement à Chrome ([notification de Chrome 49](https://chrome.googleblog.com/2015/11/updates-to-chrome-platform-support.html)) il reçoit encore des mises à jour mais uniquement pour [quelques mois](https://blog.mozilla.org/futurereleases/2016/12/23/firefox-support-for-xp-and-vista/).

**TLS 1.1 (2006,** [**RFC 4346**](https://tools.ietf.org/html/rfc4346)**)**

TLS 1.1 est une version de transition qui consolide certaines RFC intermédiaires et inclut une modification de l’initialisation des blocs CBC (suite à un signalement fait en 2001) qui prévient l’attaque *BEAST* qui n’apparaitra toutefois qu’en 2011. Cette attaque a révélé à quel point la cryptographie sur le web était en retard, en effet en 2011 aucun navigateur ne peut traiter TLS 1.1 ou 1.2 et il en va pratiquement de même pour les serveurs (OpenSSL évolue alors malheureusement très lentement).

Le rattrapage brutal effectué depuis lors explique pourquoi TLS 1.1 est une version que l’on peut pratiquement oublier, en effet la gestion de TLS 1.1 est très souvent apparue simultanément avec celle de TLS 1.2. La question qui se pose au niveau de la configuration d’un serveur est donc de savoir jusqu’à quand il doit encore accepter TLS 1.0 avant de passer exclusivement à TLS 1.2 et plus récent.



Les commentaires du fichier de configuration httpd-tls.conf d’Apache 2.4.x suggèrent de désactiver TLS 1.0 dès que possible et de ne conserver que TLS 1.2 d’actif à compter de la fin 2016.

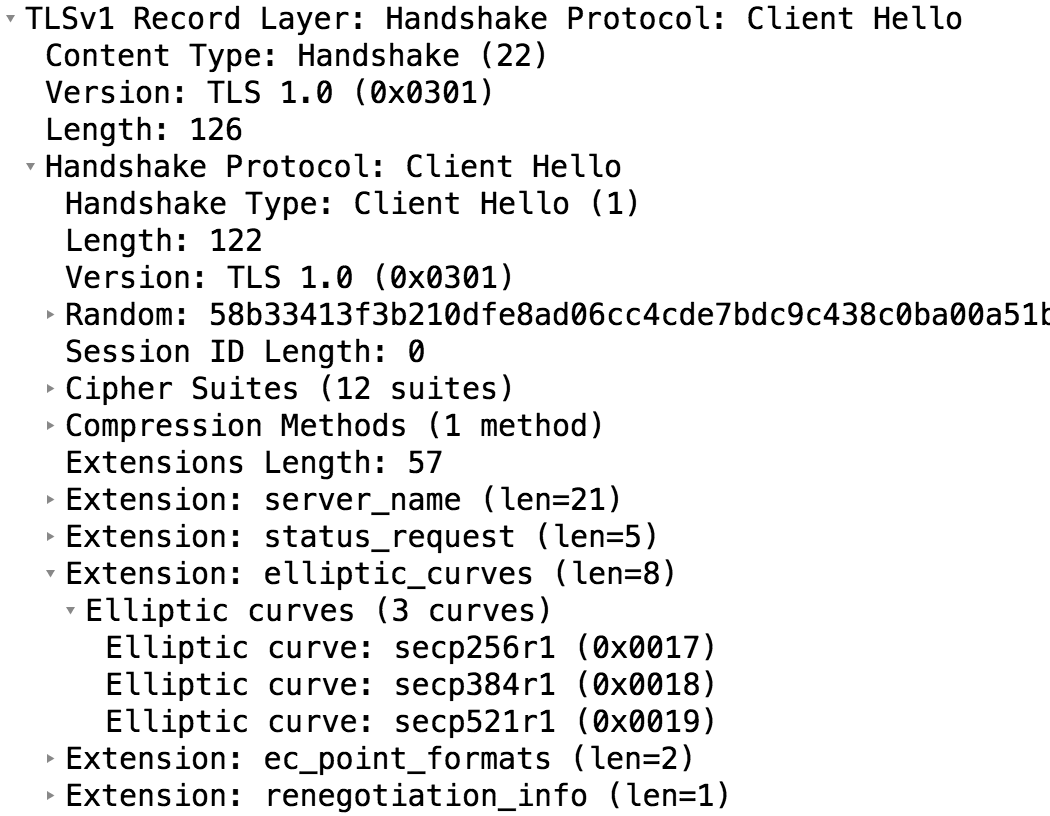
**ECC —*Elliptic Curve Cryptography*— (2006,** [**RFC 4492**](https://tools.ietf.org/html/rfc4492)**)**

La cryptographie à base de courbes elliptiques est une rupture technologique (enfin mathématique) par rapport aux algorithmes précédents qui reposent souvent sur de grands nombres entiers et la difficulté de les factoriser (type RSA). Utiliser les équivalents ECxxx présente quelques avantages :

* la description d’un point sur une courbe est un objet bien plus petit ;
* les calculs sont globalement plus rapides ;
* les plus petites clés EC offrent déjà un niveau de sécurité un cran au-dessus de ce qui se fait communément (équivalent à 3072 bits en asymétrique).

Utiliser en priorité l’algorithme d’échange de clé ECDHE est désormais unanimement conseillé.

En revanche, l’utilisation de clés et de signatures ECDSA est encore rare dans les certificats par rapport à l’omniprésent RSA mais celle-ci devrait connaitre un regain d’intérêt dans les prochaines années avec la disparition des clients web incompatibles (principalement IE et Chrome sur Windows XP).



L’intérêt de l’ECC est tel qu’il a souvent été intégré dans des composants par ailleurs limités à TLS 1.0 comme ici IE sur Windows Vista mais également les anciennes versions d’Android et les systèmes d’Apple.

**TLS 1.2 (2008,** [**RFC 5246**](https://tools.ietf.org/html/rfc5246)**)**

TLS 1.2 apporte une évolution importante : AEAD —*Authenticated Encryption with Associated Data*— le chiffrement authentifié (défini en propre dans la [RFC 5116](https://tools.ietf.org/html/rfc5116)) sous la forme des modes GCM et CCM. Malheureusement TLS 1.2 autorise l’usage de vieilles suites cryptographiques ce qui permet de réaliser des mélanges de modernité et d’archaïsme potentiellement dangereux.

Le simple fait activer TLS 1.2 côté serveur ne garantit pas d’avoir une configuration de qualité, c’est nécessaire mais pas suffisant. Le choix et l’ordre des suites cryptographiques utilisables ainsi que, dans une moindre mesure, le certificat employé vont également jouer un rôle déterminant.

**AES-GCM (2008,** [**RFC 5288**](https://tools.ietf.org/html/rfc5288)**)**

GCM —*Galois Counter Mode*— est le premier mode de chiffrement authentifié défini et le plus utilisé de nos jours…

**HMAC SHA2, ECC et AES-GCM (2008,** [**RFC 5289**](https://tools.ietf.org/html/rfc5289)**)**

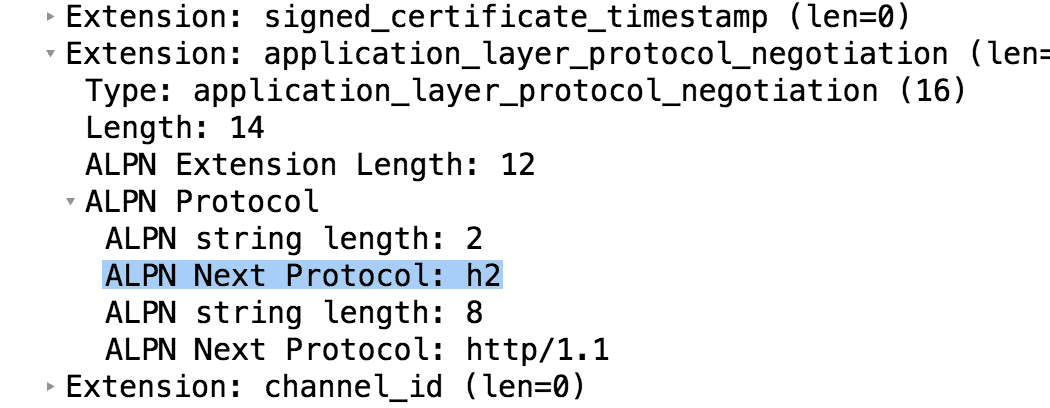
…mais l’échange de clé ECDHE étant à privilégier, les suites cryptographiques considérées comme étant les plus sûres proviennent de cette seconde RFC, qui définit également les HMAC en SHA-2 pour les modes CBC.

**AES-CCM (2012,** [**RFC 6655**](https://tools.ietf.org/html/rfc6655)**)**

L’autre mode de chiffrement authentifié n’a été ajouté que quelques années après sous la forme de CCM —*Counter with CBC-MAC*— ce qui a comme effet qu’aucun navigateur web ne l’utilise actuellement, il est disponible dans OpenSSL depuis la version 1.1.0.

**ALPN —*Application-Layer Protocol Negotiation*— (2014,** [**RFC 7301**](https://tools.ietf.org/html/rfc7301)**)**

Cette extension permet d’indiquer dès la négociation TLS que le client web est disposé à faire transiter autre chose que HTTP, c’est-à-dire HTTP/2 (h2) ou éventuellement son ancêtre SPDY (spdy/3), si le serveur en est capable il va donc basculer vers le nouveau protocole. Il est à noter que pour assurer un niveau de sécurité minimal HTTP/2 prohibe une grande quantité de suites cryptographiques dont la liste d’exclusion figure dans l’annexe A : TLS 1.2 *Cipher Suite Black List* de la [RFC 7540](https://tools.ietf.org/html/rfc7640).



***Encrypt then MAC* (2014,** [**RFC 7366**](https://tools.ietf.org/html/rfc7366)**)**

Les opérations disjointes de chiffrement et de contrôle d’intégrité opérées lors de l’utilisation du mode CBC se font dans un ordre qui n’est plus considéré comme le plus sûr. Cette RFC introduit une subtile modification pour remettre ces opérations dans le bon ordre à l’aide d’une énième extension… mais ceci arrive quelque peu tardivement car les clients et serveurs susceptibles de la mettre en œuvre disposent déjà du chiffrement authentifié sous la forme du mode GCM qui n’est pas confronté à ce problème. En définitive il semble qu’aucun client web n’intègre cette modification et TLS 1.3 réserve un sort tout particulier au mode CBC…

***TLS Fallback SCSV* (2015,** [**RFC 7507**](https://tools.ietf.org/html/rfc7507)**)**

Comme les versions les plus anciennes de TLS/SSL sont également les moins sûres, pour perpétrer une attaque un assaillant pouvant s’interposer entre le client et le serveur (on parle de MITM —*Man In The Middle*— en anglais) pourrait procéder à une attaque par repli qui consiste à proprement éliminer les demandes de négociations pour TLS 1.2 et TLS 1.0 et ne laisser passer que celle pour SSLv3.

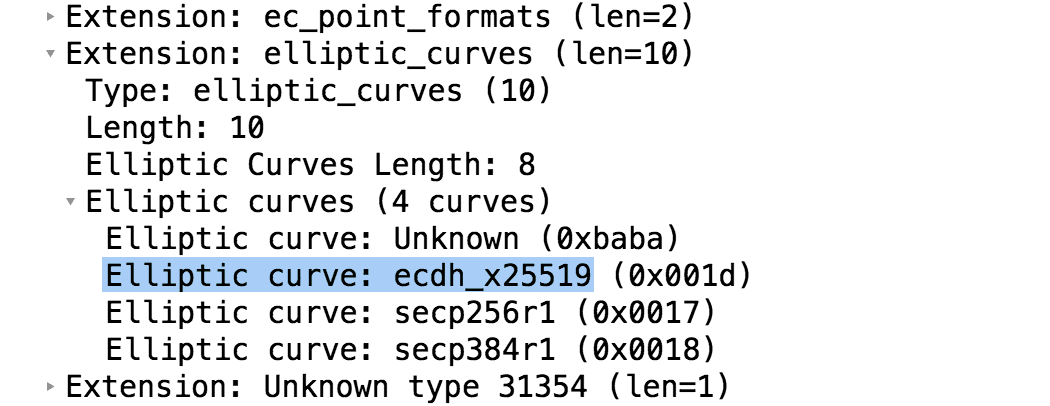
Le serveur ne voyant arriver que cette dernière il négocierait donc cette version sans avoir la moindre notion qu’il s’est passé quelque chose sur le réseau. Mais si le client insère la SCSV —*Signaling Cipher Suite Value*— à la fin de la liste des suites cryptographiques qu’il souhaite utiliser lors de ses tentatives en mode dégradé, le serveur aura la possibilité que se rendre compte qu’il n’a pas affaire à un vieux client mais au contraire à un client récent dont les précédentes demandes ne lui sont pas parvenues, il pourra donc l’alerter en retour.

***Extended Master Secret* (2015,** [**RFC 7627**](https://tools.ietf.org/html/rfc7627)**)**

Pour lutter contre une forme d’attaque par interposition dite *Triple Handshake* cette RFC propose d’améliorer le calcul du secret partagé en y incorporant une empreinte portant sur l’ensemble des données échangées lors de l’établissement de la liaison.

**Curve25519 et Curve448 (2016,** [**RFC 7748**](https://tools.ietf.org/html/rfc7748)**)**

Une pierre d’achoppement qui a retardé l’adoption de la cryptographie à base de courbes elliptiques est le choix des courbes en question, en effet certains redoutaient que les courbes proposées par le NIST —*National Institute of Standards and Technology*— comme P-256, P-384 et P-521 ne présentent des propriétés douteuses mais de prime abord bien cachées. Les nouvelles courbes 25519 et 448 issues des travaux de [Daniel J. Bernstein](https://fr.wikipedia.org/wiki/Daniel_J._Bernstein) devraient soulever bien moins d’objections.



Chrome et Firefox proposent déjà d’utiliser la fonction x25519 pour les échanges de clé ECDHE. Au passage : tout ce qui apparaît ici comme « *Unknown* » ne reflète pas une lacune de Wireshark, mais correspond à des valeurs fictives de type GREASE —*Generate Random Extensions And Sustain Extensibility*— dont le but est de stresser en permanence les implémentations TLS côté serveur pour voir si elles résistent a des données inattendues, ce système n’est qu’un brouillon à l’heure actuelle, voir [*draft-ietf-tls-grease*](https://tools.ietf.org/html/draft-ietf-tls-grease-00).

**ChaCha20-Poly1305 (2016,** [**RFC 7905**](https://tools.ietf.org/html/rfc7905)**)**

ChaCha20 est un nouvel algorithme de chiffrement par flux assez simple et rapide dérivé des travaux de Daniel J. Bernstein. Comme il est particulièrement adapté aux processeurs ARM (ceux qu’on trouve communément dans les téléphones et autres tablettes), il s’est très vite retrouvé dans Chrome et sur les serveurs de Google. Firefox n’est pas en reste, mais il faut OpenSSL 1.1.0 ou une version récente de LibreSSL pour en disposer côté serveur.

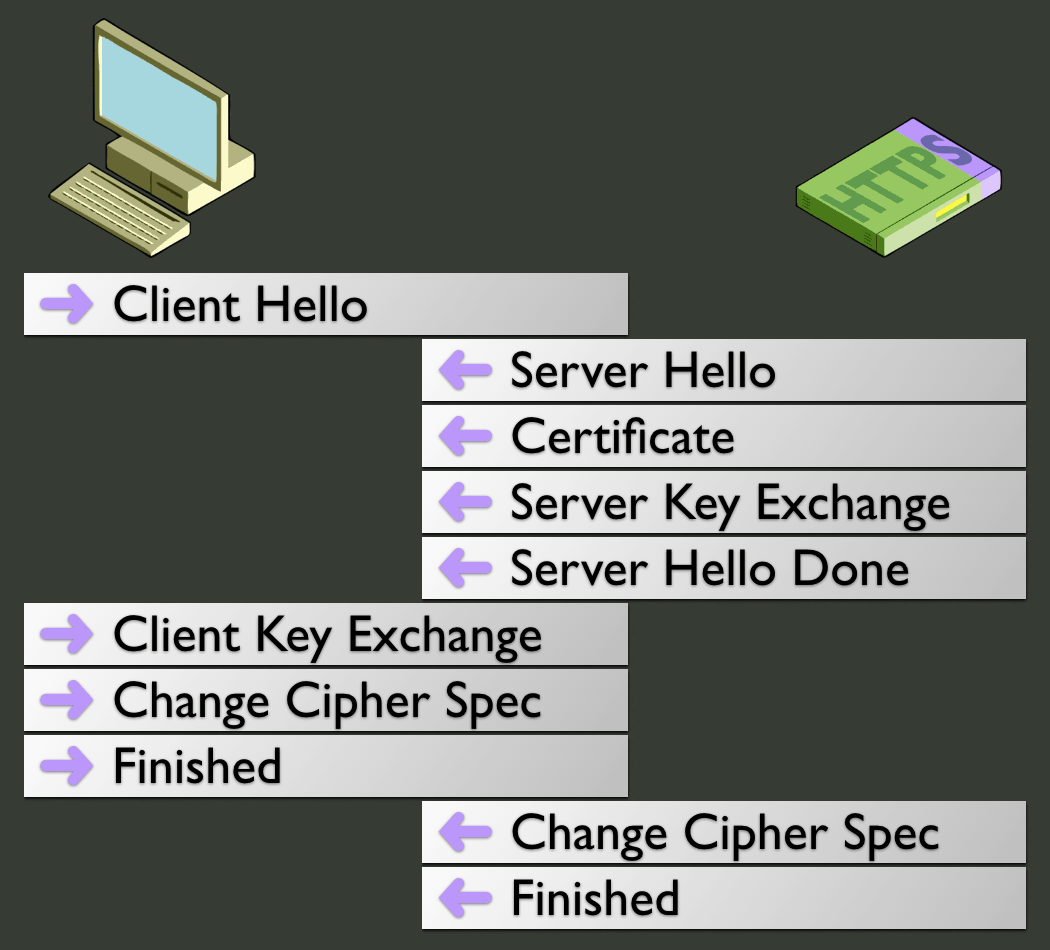
**EdDSA —*Edwards-curve Digital Signature Algorithm*— (2017,** [**RFC 8032**](https://tools.ietf.org/html/rfc8032)**)**

EdDSA est un algorithme de signature que l’on doit à… Daniel J. Bernstein et son équipe, il est relativement similaire à ECDSA mais est plus rapide et fait exclusivement usage des courbes Ed25519 et Ed448.

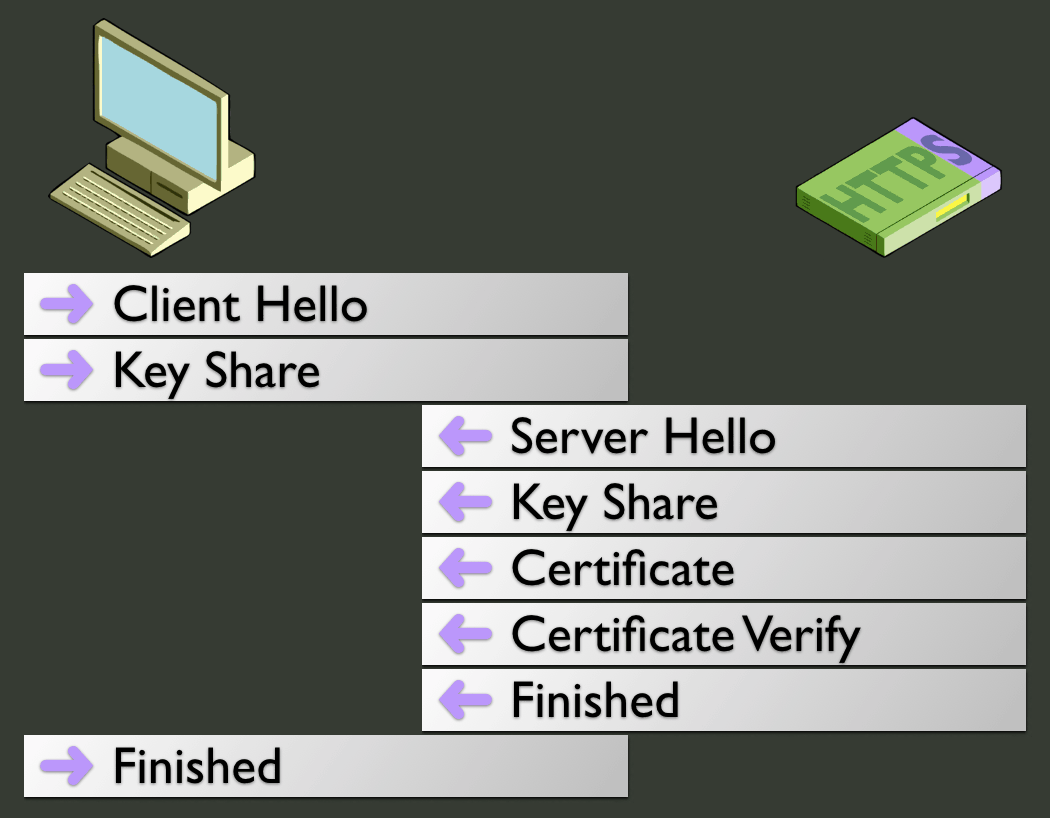
**TLS 1.3 (2017, brouillon final en février)**

TLS 1.3 est une évolution très importante, elle apporte :

* une vitesse accrue (en accélérant la négociation en réduisant les allers-retours entre le client et le serveur) ;
* une sécurité renforcée (en désactivant ou améliorant tout ce qui pouvait être abusé dans TLS 1.2 et en incluant de nouveaux algorithmes).



Le processus complet d’établissement de la liaison sécurisée ressemble le plus souvent à ces deux vagues d’aller-retour avec TLS 1.2.



Avec TLS 1.3 l’ordre change, des anticipations ont lieu pour réduire les échanges au minimum, comme le client à encore la main après le « *Finished* » il peut transmettre une requête HTTP GET dans la foulée, réduisant en pratique l’échange complet à un seul aller-retour (1 RTT), ce qui devrait produire un gain conséquent lorsque le client et le serveur sont très éloignés ou lorsque le client passe par un réseau de communication mobile, la latence étant dans ces deux cas importante.

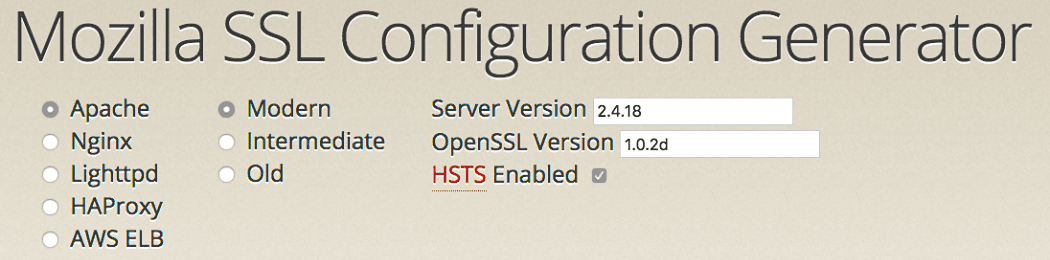
Il est à noter que le mécanisme de reprise de session a également été complètement remanié pour être plus rapide. L’implication de Google dans le développement et la promotion de HTTP/2 et TLS 1.3 et la large diffusion de HTTPS est considérable. Il n’est donc pas impossible qu’à l’avenir ils cherchent à accélérer l’adoption de TLS 1.3 en lui donnant un petit coup de pouce au niveau du classement des résultats de recherche, pour l’instant il n’y a rien de tel mais « plus sûr et plus rapide » est un bon facteur de différentiation qui peut être associé à TLS 1.3.

TLS 1.3 a la particularité d’être la première version à largement s’affranchir des recommandations du NIST et de ce fait d’une éventuelle influence de la NSA (dont certaines pratiques visant à [affaiblir les systèmes cryptographiques](https://www.scientificamerican.com/article/nsa-nist-encryption-scandal/) ont été mises en lumière). Cela se traduit par l’adoption d’algorithmes dérivés des travaux de Daniel J. Bernstein (x25519, EdDSA, ChaCha20…), de ce fait on pourra à l’avenir basculer du tout NIST vers du tout DJB. Il est toujours possible d’utiliser les courbes NIST, ECDSA et AES avec TLS 1.3, mais il existe désormais une alternative pour chacun.

Par ailleurs un sacré coup de balai a été donné dans les vieilles suites cryptographiques, le mécanisme d’échange de clé le plus simple (celui chiffrant le secret partagé avec la clé RSA publique du serveur et qui n’offrait donc pas de confidentialité persistante) n’est plus utilisable, le mode CBC disparait de même que les vieux algorithmes de chiffrement comme Triple DES. L’élimination de tout ce qui n’a pas fait preuve de robustesse contribue à rendre TLS 1.3 sûr par défaut (contrairement à TLS 1.2 qui restait tourné vers le passé) et simplifie grandement sa configuration. Le seul point sensible, a priori, est la possibilité de faire du 0 RTT mais qui ne devrait pas être activé par défaut.

**Mettre fin aux erreurs du passés**

Le fait que l’on trouve encore autant de serveurs configurés avec une rétrocompatibilité bien trop étendue ou des suites cryptographiques dépassées a probablement plusieurs origines, configurer SSL/TLS n’a pas toujours été simple et clairement documenté (encore moins en français) et, une fois que cela fonctionne (les clients arrivent à se connecter), il peut être tentant de ne plus y toucher pendant quelques années… c’était d’autant plus facile lorsqu’il existait des certificats valables quatre ou même cinq ans. Les outils comme [*SSL Server Test*](https://www.ssllabs.com/ssltest/) de *SSL Labs* ou le redoutable [CryptCheck](https://tls.imirhil.fr/) d’Aeris qui attribuent des notes et proposent des améliorations ou [le générateur de configuration de Mozilla](https://mozilla.github.io/server-side-tls/ssl-config-generator/) qui s’efforce de simplifier la configuration de différents serveurs sont encore trop méconnus.





Le système de notation de *SSL Labs* évolue régulièrement pour s’adapter aux nouvelles failles ou bonnes pratiques. Le A obtenu aujourd’hui pourrait fort bien ne plus être qu’un B dans quelques mois.

En se fixant pour règle de n’utiliser que deux ou trois configurations TLS types (les configurations « moderne » et « intermédiaire » proposées par Mozilla sont un bon début), il est bien plus simple de s’assurer régulièrement que tous les sites que l’on contrôle restent dans les clous.

La sécurité informatique est un processus qui implique de remettre l’ouvrage sur le métier fréquemment et d’assurer un minimum de veille, sur ce sujet je ne peux que conseiller de s’abonner à la lettre mensuelle « [*Bulletproof TLS Newsletter*](https://www.feistyduck.com/bulletproof-tls-newsletter/) » compilée par l’éditeur du livre qui fait office de référence dans le domaine : « [*Bulletproof SSL and TLS*](https://www.feistyduck.com/books/bulletproof-ssl-and-tls/) » d’Ivan Ristić.

Seuls des électrochocs comme *BEAST, POODLE* ou [*Heartbleed*](https://fr.wikipedia.org/wiki/Heartbleed) et certaines des révélations d’Edward Snowden ont eu pour effet d’accélérer l’adoption de technologies plus récentes. Le fait que la crypto sur le web n’évolue qu’en réaction à des incidents majeurs est plutôt inquiétant, on est loin d’une situation d’amélioration continue.

Le lancement de TLS 1.3 se fait dans un contexte totalement différent de celui de TLS 1.2, car TLS 1.3 a vocation à accompagner le passage au tout HTTPS lié aux craintes de piratages et de collectes massives d’informations sur le réseau. Cette nouvelle version est de plus soutenue par des acteurs de tout premier plan (des éditeurs de navigateurs et des opérateurs de CDN —*Content Delivery Network*—) qui ont produit le code nécessaire pour tester les versions préliminaires, ce qui promet une adoption très rapide par quelques acteurs qui génèrent un trafic conséquent.

Il faudra cependant de longues années avant que TLS 1.3 ne devienne la version la plus répandue sur l’ensemble des serveurs web (il faudra entre autres attendre une large diffusion d’OpenSSL 1.1.1 dont la disponibilité n’est pas prévue avant avril 2017). Il est cependant possible de s’inspirer des recommandations et des changements qui accompagnent TLS 1.3 pour mieux configurer TLS 1.2, se qui se traduit souvent par la désactivation de suites cryptographiques qui datent de SSLv3 et TLS 1.0 et un peu d’audace pour adopter plus de cryptographie basée sur les courbes elliptiques, y compris dans les certificats.

**Notes**

[[1](https://openweb.eu.org/articles/https-de-ssl-a-tls-1-3#nh1)] Écrire le code informatique qui gère la cryptographie est une affaire de spécialistes, de ce fait les logiciels qui en ont l’usage (y compris les clients et les serveurs web) utilisent des bibliothèques spécialisées comme : OpenSSL (souvent utilisé par Apache via mod\_ssl), LibreSSL (*fork* d’OpenSSL par l’équipe d’OpenBSD, facilement utilisable avec Nginx), BoringSSL (fork d’OpenSSL par Google se trouve dans Chrome et sur les serveurs de Google), *Network Security Services* / NSS (de Netscape puis Mozilla se trouve dans Firefox, peut être utilisé par Apache via mod\_nss), *Secure Channel* / *SChannel* (de Microsoft, se trouve dans Windows et est utilisé par IE/Edge et IIS) ou encore *Secure Transport* (d’Apple, se trouve dans macOS et iOS et est utilisé par Safari). Ceci entraîne parfois des imprécisions, c’est notamment le cas avec Internet Explorer pour lequel il faudrait distinguer IE 8 sur XP d’IE 8 sur Vista car les versions de *Secure Channel* qui équipent ces deux systèmes n’ont pas exactement les mêmes capacités.