RFC 9849 : TLS Encrypted Client Hello

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 4 mars 2026

Date de publication du RFC : Mars 2026

https://www.bortzmeyer.org/9849.html

La cryptographie est indispensable à la sécurité dans l'Internet d'aujourd'hui. Mais sa mise en œuvre peut être délicate, par exemple lorsqu'il y a un problème d'œuf et de poule : quand un client TLS se connecte, il envoie en clair au serveur le nom de domaine utilisé, le serveur ayant besoin de cette information pour choisir le bon certificat qui servira pour choisir les paramètres de chiffrement qui protégeront le reste de la session. Cet envoi en clair pose un problème de vie privée, et est parfois utilisé pour la censure, par exemple en Russie. Il faut donc chiffrer ce nom annoncé, ce SNI. Mais avec quelle clé, puisqu'on a besoin du nom pour avoir une clé ? Ce RFC fournit un mécanisme pour cela, ECH ("*Encrypted Client Hello*"), qu'on pourrait traduire par « salutation chiffrée ».

Pour voir une communication TLS classique, avant ECH, vous pouvez regarder un pcap, . Le quatrième paquet contient le Client Hello, avec (vu par Wireshark) :

```
Extension: server_name (len=20) name=www.langtag.net
```

Le nom demandé est donc en clair. Pas bien. C'est ainsi qu'en Russie, le système TSPU ("*[Caractère Unicode non montré* [1] *][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré], [Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré] [Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré] [Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère*

1. Car trop difficile à faire afficher par LaTeX

Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré] [Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré][Caractère Unicode non montré]", mesures techniques de protection) détecte les noms censurés dans le SNI et envoie un faux paquet TCP RST ("*ReSeT*"). C'est mis en œuvre, par exemple, dans les boitiers SORM <https://books.openedition.org/pressesmines/9073>.

Un petit avertissement tout de suite : on lit assez souvent que la version 1.3 de TLS (normalisée dans le RFC 8446 [2]) chiffre ce SNI. C'est faux et, si vos lisez cela dans un article, vous saurez que l'auteur ne vérifie pas ce qu'il raconte. TLS 1.3 chiffre davantage de choses que les versions précédentes, mais pas le SNI.

C'est que chiffrer le SNI, ou, encore mieux, tout le "*Client Hello*", est compliqué puisque le client doit obtenir une clé publique du serveur, avant d'avoir pu lui indiquer le nom de domaine souhaité. Il faut donc au client une autre clé publique que celle du certificat. Notez qu'il n'y a pas que le SNI qui est sensible, l'ALPN (RFC 7301) l'est aussi, et c'est pour cela que le choix de notre RFC est de chiffrer tout le message "*Client Hello*". Toujours côté analyse de la menace, notez qu'ECH n'est utile que si le même serveur TLS gère plusieurs domaines. S'il n'en a qu'un, un obsevateur saura lequel, juste par l'adresse IP du serveur. ECH vise à empêcher cet observateur de savoir quel domaine était choisi, au sein d'un "*anonymity set*", l'ensemble des domains hébergés sur ce serveur.

Le RFC est long mais le principe d'ECH est simple. (Au fait, le cahier des charges d'ECH est dans le RFC 8744.) La section 3 le résume. ECH a deux modes, "*shared*" et "*split*" mais qui fonctionnent de manière très proche. Pour les configurations simples (le serveur est composé d'une seule machine), "*shared*" suffit. Pour les cas où le « serveur » est en fait composé deux machines, une frontale (qui relaie le TLS sans l'interpréter) et une dorsale, le "*split*" est utilisé. Oubliez cela tout de suite, si c'est important pour vous, voyez la section 10 du RFC pour les détails (et rappelez-vous d'un point de terminologie TLS : « terminer » une session TLS, ce n'est pas y mettre fin, c'est juste être l'extrêmité de la session).

On l'a dit, toute la difficulté d'ECH est que le serveur doit publier une clé qui permettra le chiffrement du "*ClientHello*". Notre RFC n'impose pas une manière unique de la publier. Elle peut être manuellement et statiquement configurée chez le client, par exemple. Mais la méthode qui sera sans doute la plus courante est une publication dans le DNS, via les enregistrements SVCB (RFC 9848). (En fait, le serveur va publier une clé et des métadonnées associées.) Une fois muni de cette clé, le client chiffre son "*Client Hello*" et met le message chiffré (`ClientHelloInner`, dit le RFC) dans une extension du "*Client Hello*" normal (appelé `ClientHelloOuter` dans le RFC), extension nommée `encrypted_client_hello` (enregistrée avec le numéro 65037 <https://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xml#tls-extensiontype-values-1>). Le serveur va déchiffrer l'extension, récupérant le bon nom de domaine, et continuera ensuite comme d'habitude.

Notez que le client doit bien mettre un nom de domaine dans le SNI du `ClientHelloOuter` (envoyé en clair) La section 6.1 du RFC détaille ce qu'il faut placer dans cette « salutation chiffrée ».

Et voici le pcap produit : . Le client annonce un nom `cover.defo.ie` dans le SNI mais le vrai nom (`defo.ie`) est dans l'extension ECH (et vous ne le voyez pas, bien sûr, il est chiffré) :

2. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc8446.txt`

```
Extension: encrypted_client_hello (len=473)
    Type: encrypted_client_hello (65037)
    Length: 473
    Client Hello type: Outer Client Hello (0)
    Cipher Suite: HKDF-SHA256/AES-128-GCM
    Config Id: 185
    Enc length: 32
    Enc: 6acfeceeea540b510bad0b28f5a9913af4e52cbdc1496750d91658d1e2200a3e
    Payload length: 431
    Payload [truncated]: b8f50aa3492607d8c05c1150b4f12496cf3910468ed82e775c0c0d64...
```

Et un exemple de requête DNS pour récupérer la clé ECH :

```
% dig  tls-ech.dev HTTPS

; <<>> DiG 9.18.18-0ubuntu0.22.04.2-Ubuntu <<>> tls-ech.dev HTTPS
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 33050
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 1472
;; QUESTION SECTION:
;tls-ech.dev.  IN HTTPS

;; ANSWER SECTION:
tls-ech.dev.  60 IN HTTPS 1 . ech=AEn+DQBFKwAgACABWIHUGj4u+PIggYXcR5JF0gYk3dCRioBW8uJq9H4mKAAIAAEAAQABAANAEnB1Ym

;; Query time: 132 msec
;; SERVER: 192.168.0.254#53(192.168.0.254) (UDP)
;; WHEN: Mon Apr 15 18:46:04 CEST 2024
;; MSG SIZE  rcvd: 134
```

(Pour Cloudflare, regardez `cloudflare-ech.com`, le nom choisi par cette entreprise pour ECH.)

À ce stade, vous savez l'essentiel. Lisez le RFC si vous voulez les détails de syntaxe (sections 5 à 7). Voyons plutôt le problème de déploiement pratique dans l'Internet (section 8 du RFC). ECH nécessite de changer clients et serveurs et nous pouvons être sûrs qu'il y aura une longue période de coexistence de logiciels ECH et non-ECH. On verra sans doute aussi des cafouillages comme un serveur qui publie une clé pour ECH mais ne gère pas ECH (le RFC prévoit un mécanisme de détection de ce problème et de ré-essai - section 6.1.6 - dans ce cas, si bien que le serveur qui ferait cette erreur imposera un délai de connexion plus long à ses clients).

Pour éviter que l'utilisation de ECH se remarque trop, le RFC spécifie également (section 2) comment graisser (RFC 9170) les communications TLS afin de rendre plus difficile la détection des vrais usages.

Ah, et le RFC précise, avec un humour discret, qu'ECH va casser « certains usages », euphémisme pour désigner la surveillance mais aussi la censure, que le SNI en clair permettait (pour la censure, on fait du DPI et, si on trouve un nom censuré dans un SNI, on envoie un message TCP RST - "*ReSeT*" - mensonger au client). Cela a déjà fait l'objet de critiques `<https://labs.ripe.net/author/kathleen_moriarty/security-control-changes-due-to-tls-encrypted-clienthello/>`, regrettant qu'on ne puisse plus espionner autant.

Cette technique de censure (DPI puis envoi d'un RST TCP si on trouve un nom censuré) est rare en Europe mais très répandue en Russie <https://books.openedition.org/pressesmines/9073>, notamment depuis le début de la guerre. ECH permet de l'empêcher.

Le but d'ECH est d'améliorer la sécurité de TLS, il n'est donc pas étonnant que la section d'analyse de sécurité, la section 10, soit très détaillée. Elle commence par exliquer le modèle de menace (un attaquant situé sur le trajet, qui peut être actif (capable d'injecter des paquets). ECH empêchera cet attaquant de savoir quel service était demandé, parmi tous ceux hébergés à cette adresse IP (et cela a été formellement prouvé dans « *"A Symbolic Analysis of Privacy for TLS 1.3 with Encrypted Client Hello"* <https://research.cloudflare.com/publications/Bhargavan2022/> »). Mais il reste des pièges.

ECH peut récupérer la clé du serveur dans le DNS (c'est en tout cas la méthode par défaut) mais n'impose pas DNSSEC donc un attaquant qui peut injecter de faux enregistrements DNS peut casser ECH. La solution est évidemment d'utiliser DNSSEC. (Le RFC note aussi qe, si l'attaquant est sur le réseau local de la victime, utiliser DoT ou DoH pour la résolution de noms va aider.)

Certaines extensions de TLS peuvent défaire ECH, par exemple la `cached_info` du RFC 7924. Elles ne doivent pas être utilisées dans le `ClientHelloOuter`. ALPN eut aussi être révélateur si un client nvoie des ALPN différents à différents serveurs. à encore, ne pas le mettre dans le `ClientHelloOuter`.

La vérification du certificat contient également des risques, si on utilise OCSP, dont la requête peut trahir le nom du serveur contacté.

Le RFC note à juste titre qu'un observateur indélicat a d'autres moyens à sa disposition pour découvrir le domaine demandé. Le plus évident est le DNS (cf. RFC 7626), et il faut donc utiliser les mécanismes de protection de la vie privée (RFC 7858, RFC 8484, mais aussi RFC 9156, que le RFC sur ECH a malencontreusement oublié).

Et il y a bien d'autres faiblesses possibles si on utilise mal ECH, lisez la section 10 si vous voulez en savoir plus, vous apprendrez plein de choses.

ECH a été défini il y a déjà plusieurs années. (Le projet ECH à l'IETF a duré bien plus longtemps que prévu. Ce RFC a commencé sa vie en 2018 <https://datatracker.ietf.org/doc/draft-ietf-tls-esni/>.) Il existe de nombreuses mises en œuvre. On le trouve par exemple dans Firefox (cf. leur documentation « Comprendre le client chiffré Hello (ECH) <https://support.mozilla.org/fr/kb/comprendre-client-hello-chiffre-ech> » ou bien le wiki pour des détails pointus <https://wiki.mozilla.org/Security/Encrypted_Client_Hello>). Les bibliothèques TLS en logiciel libre, comme BoringSSL <https://boringssl.googlesource.com/boringssl/> ou OpenSSL ont souvent ECH (mais pas forcément dans une version officielle et publiée). Si vous voulez tester si votre client TLS gère ECH, il existe plusieurs sites Web pour cela : , ou . Côté serveur, c'est en cours, par exemple pour Apache, il y a eu du code <https://github.com/apache/httpd/pull/551>, publié à partir de la version 2.5.1 (paramètre `SSLECHKeyDir`). Pour tester vos serveurs, vous pouvez utiliser curl si vous avez un curl récent, compilé avec les bonnes options. Car, sinon (sur un Arch à jour) :

```
% curl --ech true tls-ech.dev
curl: option --ech: the installed libcurl version does not support this
```

Pour davantage de tests, il y a echspec <https://github.com/thekuwayama/echspec> :

```
% echspec www.bortzmeyer.org
TLS Encrypted Client Hello Server
...  HTTPS resource record for www.bortzmeyer.org does NOT have ech SvcParams.
1 failure
```

(Pas d'ECH encore sur ce blog mais ça n'aurait guère d'intérêt puisque peu de sites sont sur ce serveur.)

```
%  echspec tls-ech.dev
TLS Encrypted Client Hello Server
...
Failures:

 1) MUST abort with an "illegal_parameter" alert, if ECHClientHello.type is not a valid ECHClientHelloType in Cl
  https://datatracker.ietf.org/doc/html/draft-ietf-tls-esni-22#section-7-5
  did not send expected alert: illegal_parameter
1 failure
```

À part un point de détail subtil, tout va bien. Par contre, ne teste pas l'ECH. Dommage.

Quelques lectures intéressantes :

— « *"Say (an encrypted) hello to a more private internet"* <https://blog.mozilla.org/en/products/firefox/encrypted-hello/> », une courte explication par Mozilla.
— L'annonce de Cloudflare <https://blog.cloudflare.com/announcing-encrypted-client-hello/>.
— D'un intérêt historique seulement, la première annonce de Cloudflare <https://blog.cloudflare.com/encrypted-sni/>, quand ECH s'appelait encore ESNI (car il ne chiffrait que le SNI, pas tout le ClientHello, laissant par exemple l'ALPN sans protection).