RFC 8548 : Cryptographic Protection of TCP Streams (tcpcrypt)

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 23 mai 2019

Date de publication du RFC : Mai 2019

https://www.bortzmeyer.org/8548.html

Aujourd'hui, il n'est plus acceptable d'avoir des communications non chiffrées. États, entreprises et délinquants surveillent massivement les réseaux publics et toute communication effectuée en clair peut être espionnée, voire modifiée. Il est donc nécessaire de continuer les efforts pour chiffrer ce qui ne l'est pas encore. Ce RFC décrit un mécanisme expérimental pour TCP, nommé tcpcrypt, permettant de chiffrer les communications sans participation de l'application située au-dessus, sans authentification obligatoire. (Mais le projet semble mal en point donc je ne suis pas optimiste quant à son déploiement.)

La section 2 du RFC est le cahier des charges de « tcpcrypt », ce nouveau mécanisme de protection de TCP :

- Pouvoir être mis en œuvre conjointement à TCP. Cela implique de pouvoir tourner dans le noyau, où on ne peut pas charger des bibliothèques comme OpenSSL. Et certaines piles TCP tournent sur des machines contraintes (Internet des Objets), donc le protocole doit être léger.
- Ne pas trop augmenter la latence lors de la négociation cryptographique.
- Tolérer certains intermédiaires qui se permettent de modifier l'en-tête TCP.
- Ne pas faire de liaison avec l'adresse IP : une session tcpcrypt doit pouvoir reprendre si l'adresse IP a changé.

tcpcrypt est très différent des classiques TLS et SSH. Il est conçu pour ne **pas** impliquer l'application, qui peut ignorer qu'on chiffre dans les couches inférieures. tcpcrypt est prévu pour être une solution simple, ne nécessitant pas de modifier protocoles ou applications, changeant le moins de chose possible pour être déployable. Il est également intéressant de voir le « non-cahier des charges », ce qui n'est pas obligatoire dans tcpcrypt :

- Aucune authentification n'est faite par tcpcrypt,
- En cas de problème, on se replie sur du TCP non sécurisé.

Ces deux points rendent tcpcrypt vulnérable aux attaquants actifs.

Pendant la longue et douloureuse gestation de ce protocole, TLS avait été envisagé comme alternative. Après tout, pourquoi inventer un nouveau protocole de cryptographie, activité longue et délicate (les failles de sécurité sont vite arrivées) ? Il y avait deux propositions sur la table à l'IETF, le futur tcpcrypt, et une solution fondée sur TLS. C'est pour essayer de faire fonctionner les deux solutions que la négociation des paramètres avait été traitée à part (option ENO, RFC 8547 [1]). Mais, depuis, la proposition TLS a été "*de facto*" abandonnée, en partie parce que la communauté TLS était occupée par le travail sur la version 1.3.

tcpcrypt s'appuie sur l'option TCP ENO ("*Encryption Negotiation Option*") pour la négociation de l'utilisation du chiffrement. Ce RFC 8548 décrit comment chiffrer, une fois les paramètres négociés.

La section 3 décrit le protocole en détail. Je ne vais pas la reprendre ici (la cryptographie n'est pas mon point fort). On est dans le classique, de toute façon, avec cryptographie asymétrique pour se mettre d'accord sur une clé et cryptographie symétrique pour chiffrer ; tcpcrypt utilise trois types d'algorithmes cryptographiques :

— Un mécanisme de négociation de clé pour, à partir d'une clé publique temporaire, se mettre d'accord sur un secret partagé (qui servira, après traitement, à chiffrer la session),
— une fonction d'extraction pour tirer de ce secret partagé une clé,
— une fonction pseudo-aléatoire pour tirer de cette clé les clés de chiffrement symétrique.

La fonction d'extraction et la fonction pseudo-aléatoire sont celles de HKDF (RFC 5869), elle-même fondée sur HMAC (RFC 2104). Une fois qu'on a la clé, on chiffre avec un algorithme de chiffrement intègre.

Comme vous avez vu, les clés publiques utilisées dans le protocole tcpcrypt sont temporaires, jamais écrites sur disque et renouvellées fréquemment. Ce ne sont pas des clés permanentes, indiquant l'identité de la machine comme c'est le cas pour SSH. tcpcrypt n'authentifie pas la machine en face (la section 8 détaille ce point).

La négociation du protocole, pour que les deux parties qui font du TCP ensemble se mettent d'accord pour chiffrer, est faite avec l'option TCP ENO ("*Encryption Negotiation Option*"), décrite dans le RFC 8547. La négociation complète peut nécessiter un aller-retour supplémentaire par rapport à du TCP habituel, ce qui augmente la latence d'établissement de connexion. Un mécanisme de reprise des sessions permet de se passer de négociation, si les deux machines ont déjà communiqué, et gardé l'information nécessaire.

Une fois la négociation terminée, et le chiffrement en route, tcpcrypt génère (de manière imprévisible, par exemple en condensant les paramètres de la session avec un secret) un "*session ID*", qui identifie de manière unique cette session tcpcrypt particulière. Ce "*session ID*" est mis à la disposition de l'application via une API, qui reste à définir et l'application peut, si elle le souhaite, ajouter son mécanisme d'authentification et lier une authentification réussie au "*session ID*". Dans ce cas, et dans ce cas seulement, tcpcrypt est protégé contre l'Homme du Milieu.

Notez que seule la charge utile TCP est chiffrée, et donc protégée. L'en-tête TCP ne l'est pas (pour pouvoir passer à travers des boitiers intermédiaires qui tripotent cet en-tête, et tcpcrypt ne protège donc pas contre certaines attaques comme les faux paquets RST (terminaison de connexion, les détails figurent

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc8547.txt`

en section 8). Toutefois, certains champs de l'en-tête (mais pas RST) sont inclus dans la partie chiffrée (cf. section 4.2). C'est par exemple le cas de FIN, pour éviter qu'une troncation des données passe inaperçue.

L'option TCP ENO (RFC 8547) crée le concept de TEP ("*TCP Encryption Protocol*"). Un TEP est un mécanisme cryptographique particulier choisi par les deux machines qui communiquent. Chaque utilisation de l'option ENO doit spécifier son ou ses TEP. Pour tcpcrypt, c'est fait dans la section 7 de notre RFC, et ces TEP sont placés dans un registre IANA <https://www.iana.org/assignments/tcp-parameters/tcp-parameters.xml#tcp-encryption>. On y trouve, par exemple, TCPCRYPT_ECDHE_Curve25519 (le seul qui soit obligatoire pour toutes les mises en œuvre de tcpcrypt, cf. RFC 7696) qui veut dire « création des clés avec du Diffie-Hellman sur courbes elliptiques avec la courbe Curve25519 ». Pour le chiffrement lui-même, on a vu qu'il ne fallait utiliser que du chiffrement intègre, et le seul algorithme obligatoire pour tcpcrypt est AEAD_AES_128_GCM (« AES en mode GCM »). Les autres sont également dans un registre IANA <https://www.iana.org/assignments/tcp-parameters/tcp-parameters.xml#tcpcrypt-aead-algorithms>.

Le but de tcpcrypt est la sécurité donc la section 8, consacrée à l'analyse de la sécurité du protocole, est très détaillée. D'abord, tcpcrypt hérite des propriétés de sécurité de l'option ENO (RFC 8547). Ainsi, il ne protège **pas** contre un attaquant actif, qui peut s'insérer dans le réseau, intercepter les paquets, les modifier, etc. Un tel attaquant peut retirer l'option ENO des paquets et il n'y a alors plus grand'chose à faire (à part peut-être épingler la connaissance du fait qu'une machine donnée parlait tcpcrypt la dernière fois qu'on a échangé, et qu'il est bizarre qu'elle ne le fasse plus ?) Si l'application a son propre mécanisme d'autentification, situé au-dessus de tcpcrypt, et qui lie l'authentification au "*session ID*", alors, on est protégé contre les attaques actives. Sinon, seul l'attaquant passif (qui ne fait qu'observer) est bloqué par tcpcrypt. Une analyse plus détaillée figure dans l'article fondateur du projet tcpcrypt, « "*The case for ubiquitous transport-level encryption*" <https://www.usenix.org/legacy/events/sec10/tech/full_papers/Bittau.pdf> », par Bittau, A., Hamburg, M., Handley, M., Mazieres, D., et D. Boneh.. tcpcrypt fait donc du chiffrement opportuniste (RFC 7435).

tcpcrypt ne protège pas la plus grande partie des en-têtes TCP. Donc une attaque active comme l'injection de faux RST (RFC 793, et aussi RFC 5961) reste possible.

Comme la plupart des techniques cryptographiques, tcpcrypt dépend fortement de la qualité du générateur de nombres pseudo-aléatoires utilisé. C'est d'autant plus crucial qu'un des cas d'usage prévus pour tcpcrypt est les objets contraints, disposant de ressources matérielles insuffisantes. Bref, il faut relire le RFC 4086 quand on met en œuvre tcpcrypt. Et ne pas envoyer l'option ENO avant d'être sûr que le générateur a acquis assez d'entropie.

On a dit que tcpcrypt ne protégeait pas les « métadonnées » de la connexion TCP. Ainsi, les "*keepalives*" (RFC 1122) ne sont pas cryptographiquement vérifiables. Une solution alternative est le mécanisme de renouvellement des clés de tcpcrypt, décrit dans la section 3.9 de notre RFC.

Ce RFC 8548 est marqué comme « Expérimental ». On n'a en effet que peu de recul sur l'utilisation massive de tcpcrypt. La section 9 liste les points qui vont devoir être surveillés pendant cette phase expérimentale : que deux machines puissent toujours se connecter, même en présence de boitiers intermédiaires bogués et agressifs (tcpcrypt va certainement gêner la DPI, c'est son but, et cela peut offenser certains boitiers noirs), et que l'implémentation dans le noyau ne soulève pas de problèmes insurmontables (comme le chiffrement change la taille des données, le mécanisme de gestion des tampons va devoir s'adapter et, dans le noyau, la gestion de la mémoire n'est pas de la tarte). C'est d'autant plus important qu'il semble qu'après l'intérêt initial, l'élan en faveur de ce nouveau protocole se soit sérieusement refroidi (pas de "*commit*" depuis des années dans le dépôt initial <https://github.com/sorbo/tcpcrypt>).

Et les mises en œuvre de tcpcrypt (et de l'option ENO, qui lui est nécessaire) ? Outre celle de référence <https://github.com/sorbo/tcpcrypt> citée plus haut, qui est en espace utilisateur, et qui met en œuvre ENO et tcpcrypt, il y a plusieurs projets, donc aucun ne semble prêt pour la production :

— Une mise en œuvre <https://github.com/squarooticus/tcpinc-linux> dans le noyau Linux.
— Une autre <https://github.com/robertkoehler/tcpinc-freebsd> pour FreeBSD.
— Et encore une autre <https://github.com/mryraghi/tcpcrypt> pour MacOS.
— Une extension Firefox (apparemment ancienne et pas maintenue) pour afficher le session ID <https://github.com/sqs/fftcpcrypt>.
— Un module Apache, également apparemment abandonné, pour accéder au session ID <https://github.com/sqs/mod_tcpcrypt> et le fournir aux programmes tournant sur le serveur (dans la variable d'environnement `TCP_CRYPT_SESSID`, donc du code PHP, par exemple, peut lire `$_SERVER['TCP_CRYPT_SESSID']`).

Il y avait un site « officiel » pour le projet, mais qui semble désormais cassé.