RFC 9850 : The SSLKEYLOGFILE Format for TLS

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 16 juillet 2026

Date de publication du RFC : Juillet 2026

https://www.bortzmeyer.org/9850.html

Déboguer une application réseau qui utilise le chiffrement est difficile. Le but du chiffrement est justement d'empêcher un tiers (par exemple l'analyseur de paquets) de regarder ce qui se passe. Une approche possible est de demander gentiment à l'application d'exporter ses clés de chiffrement dans un fichier que l'analyseur pourra importer pour ensuite déchiffrer la communication. Ce RFC décrit le format le plus répandu pour exporter ses clés, connu sous le nom de SSLKEYLOGFILE et très commun aujourd'hui.

Le nom est trompeur car SSL est abandonné depuis longtemps (RFC 7568[1]), ce format est en fait pour TLS mais l'ancien nom est resté. Il vient d'une convention courante, définir une variable d'environnement nommée `SSLKEYLOGFILE`, qui va indiquer à l'application qu'on demande à recevoir les clés dans un fichier dont le nom est la valeur de la variable `SSLKEYLOGFILE`. Mais ce RFC n'indique pas comment activer cette fonction d'exportation de clés, il documente juste le format résultant.

Un petit rappel de cryptographie telle qu'elle est faite dans TLS (RFC 8446) : l'authentification est faite avec de la cryptographie asymétrique puis un échange de clés a lieu, permettant aux deux parties de se mettre d'accord sur des clés utilisés en cryptographie symétrique. Ce sont ces clés, qui sont évidemment secrètes (seules les deux parties qui communiquent les connaissent) qui nous intéressent ici. Sans elles, l'analyseur de paquets ne peut que baisser les bras et dire « ici, il y a des données chiffrées mais je ne peux pas te les montrer ». Ici, par exemple, Wireshark peut juste dire que c'est du TLS et que c'est chiffré :

Donc, le format du fichier des clés est simple (section 2 du RFC) : un fichier texte en UTF-8 (la partie signifiante est forcément en ASCII mais il peut y avoir des commentaires qui n'ont pas cette restriction). La fin de ligne n'est pas normalisée (on aurait pu exiger le respect du RFC 5198 mais, bon, ces fichiers ne

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc7568.txt`

sont typiquement pas échangés entre machines différentes). Les commentaires commencent par le croisillon. Chaque secret enregistré tient sur une ligne, le nom du secret, la valeur d'un champ TLS aléatoire qui permet d'identifier la connexion (il peut y avoir des secrets de plusieurs connexions distinctes dans le fichier) et enfin la valeur du secret, en hexadécimal. Ces trois champs sont séparés par des espaces. Par contre, aucune indication sur les paramètres cryptographiques de la connexion, comme l'algorithme de cryptographie utilisé donc, si on n'a pas le début de la session TLS, le fichier peut être inutilisable. Le RFC suggère d'être indulgent dans l'analyse du fichier et d'ignorer les lignes incorrectes, ce qui permet d'extraire des secrets d'un fichier abimé.

Le RFC liste ensuite les noms possibles pour les secrets. Ils dépendent de la version de TLS. Essayons avec curl et une connexion TLS 1.3 :

```
% export SSLKEYLOGFILE=./keys.sslkeylogfile

% curl https://www.example.com

% cat keys.sslkeylogfile
SERVER_HANDSHAKE_TRAFFIC_SECRET 09d... 2d7d2e4090c6...
EXPORTER_SECRET 09d... 21cbb457ee72aa...
SERVER_TRAFFIC_SECRET_0 09d... 7e0027e87...
CLIENT_HANDSHAKE_TRAFFIC_SECRET 09d... 7ef813bf0...
CLIENT_TRAFFIC_SECRET_0 09d... f4a7ab14e...
```

Lisez le RFC et la norme TLS pour avoir des explications sur chaque secret. Un registre IANA <https://www.iana.org/assignments/tls-parameters/tls-parameters.xml#tls-sslkeylogfile- des valeurs possibles a été créé et de nouveaux noms pourront y être ajoutés en suivant la procédure « Spécification nécessaire » du RFC 8126.

De toute façon, ce n'est pas vous qui allez le lire mais un logiciel comme Wireshark. En parlant de Wireshark, si on le configure pour lire ce fichier de secrets (dans le fichier `/.config/wireshark/preferences`, mettre `tls.keylog_file: /some/where/tmp/keys.sslkeylogfile`), Wireshark saura déchiffrer la session TLS vue plus haut (même pcap). Vous voyez les requêtes HTTP complètes, qui étaient masquées par le chiffrement :

L'ordre des secrets dans le fichier n'est pas pertinent. Ah et, sinon, les noms de secret se terminant par un tiret bas et un chiffre sont là pour le cas où TLS génère plusieurs clés pendant une session (RFC 8446, section 7.2). « nomDuSecret_0 » est la première.

On peut aussi obtenir ainsi les clés ECH (RFC 9849), celles du "*outer ClientHello*".

Évidemment, l'usage de cette possibilité d'écrire les clés secrètes annule tout l'intérêt de TLS. La section 3 du RFC insiste sur ce point. Cette possibilité est très pratique pour le déboguage mais ne doit surtout pas être utilisée avec des vraies sessions TLS sensibles. La lecture du fichier permet de tout déchiffrer et même la confidentialité persistante est compromise si on a ce fichier. Il faut faire attention à ce que les fichiers SSLKEYLOGFILE ne circulent pas et ne soient pas accessibles par tout le monde (bien des programmes les créent avec les autorisations d'accès par défaut, qui peuvent être assez laxistes). Et, de manière tout aussi évidente, un programme ne doit pas écrire ces secrets s'il n'est pas certain que la demande vient du vrai utilisateur. (L'utilisation d'une variable d'environnement dans l'exemple avec curl satisfait cette condition.) Le RFC suggère aussi une option de compilation permettant de compiler l'application sans gestion de SSLKEYLOGFILE.

Une note amusante : si la session TLS est de longue durée, un attaquant qui mettrait la main sur le fichier SSLKEYLOGFILE pourrait non seulement lire les données chiffrées mais également interférer avec la session tant qu'elle est en cours.

Le format ainsi normalisé peut marcher avec diverses versions de TLS, y compris dépassées (RFC 8996). Il fonctionne avec d'autres protocoles, s'ils utilisent le mécanisme de génération de clés de TLS, ce qui est le cas de DTLS (RFC 9147) ou de QUIC (RFC 9000).

Le type de média `application/sslkeylogfile` a été ajouté au registre IANA <`https://www.iana.org/assignments/media-types/media-types.xml#application`>.

Ce format SSLKEYLOGFILE est largement adopté par de nombreuses applications depuis des années, par exemple tous les navigateurs Web. (J'ai bien dit des applications ; contrairement à ce qu'on lit parfois, ce ne sont pas les bibliothèques TLS comme OpenSSL qui le mettent en œuvre. Certains programmes utilisant OpenSSL ont cette fonction d'exportation de clés et d'autres pas.) Un exemple avec mon client Gemini Agunua <`https://framagit.org/bortzmeyer/agunua/`> :

```
% export SSLKEYLOGFILE=./keys.sslkeylogfile
% agunua gemini.bortzmeyer.org
% cat keys.sslkeylogfile
SERVER_HANDSHAKE_TRAFFIC_SECRET 724f8f07224... 8bcf43772bc21b45ad...
...
```

Agunua est écrit en Python et activer cette fonction est simple, on s'appuie sur le fait qu'OpenSSL fournit une fonction pour demander à recevoir notification de la génération des clés :

```
def write_keys(conn, keys):
    keylogfile.write(keys.decode() + "\n")

if "SSLKEYLOGFILE" in os.environ:
    keylogfile = open(os.environ["SSLKEYLOGFILE"], "a")
    context.set_keylog_callback(write_keys)
```

(Pour curl, cité plus haut, regarde le fichier `lib/vtls/keylog.c`.)

La question plus générale du déboguage des applications dans un monde où tout est chiffré avait fait l'objet de mon exposé à Capitole du Libre 2022 <`https://www.bortzmeyer.org/capitole-du-libre-2022.html`> dont voici mes supports (en ligne sur `https://www.bortzmeyer.org/files/capitole-libre-2022-tout.pdf`).