RFC 9987 : SSH Agent Protocol

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 28 mai 2026

Date de publication du RFC : Mai 2026

https://www.bortzmeyer.org/9987.html

Voici encore un RFC qui normalise quelque chose qui existait depuis longtemps : le protocole Agent de SSH.

SSH est normalisé dans le RFC 4251 [1] et il permet la connexion à distance (RFC 4253) avec authentification (RFC 4252 et RFC 4254) par exemple avec une clé publique. Il est probablement inutile de le présenter davantage aux lecteurices de ce blog. Une des fonctions géniales de SSH est la possibilité d'avoir un agent (rien à voir avec l'IA agentique qui est à la mode en ce moment) qui mémorise les clés privées et peut effectuer les opérations demandées. Ainsi mémorisées, les clés seront utilisables sans nouvelle intervention de l'utilisateur (taper une phrase de passe, etc) tout en restant bien sécurisées. Et l'agent peut intervenir sur des connexions distantes, ce qui évite de copier sa clé privée sur des serveurs à qui on ne fait pas forcément totalement confiance. L'agent ne tourne pas dans le client SSH mais dans un processus dédié, ce qui améliore sa sécurité. Si vous êtes connecté en ce moment, vous avez sans doute un agent SSH qui tourne (ici sur une Debian) :

```
% ps uxwww | grep ssh
bortzme+  459934  0.0  0.0  10700  4964 ?        Ss   09:44   0:00 /usr/bin/ssh-agent /home/bortzmeyer/.xsession
```

Comme indiqué au début, ce mécanisme d'agent est connu, mis en œuvre et utilisé depuis très longtemps, ce RFC est une documentation "*a posteriori*" (le très ancien document `draft-ietf-secsh-agent` décrivait un protocole différent).

Donc, comment fonctionne ce protocole Agent (section 2 du RFC) ? Il est client-serveur, le serveur étant l'agent et le client de l'agent n'étant pas forcément un client SSH (mais, bon, c'est le cas le plus

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc4251.txt`

courant). Le client envoie des requêtes à l'agent et reçoit des réponses (comme dans beaucoup de protocoles réseau…). L'agent est un serveur pur, il ne fait que répondre au client, sans prendre d'initiatives. Les requêtes typiques sont le chargement d'une clé, la suppression d'une clé, la signature en utilisant une des clés. Le serveur reste maitre d'accepter ou pas les requêtes et le client doit donc être prêt à voir une requête refusée, par exemple parce que l'agent n'accepte que les clés d'un certain type.

La section 3 du RFC détaille les messages échangés entre le client et l'agent (le serveur). Ils sont de type TLV et les types figurent dans un registre IANA <https://www.iana.org/assignments/ssh-parameters/ssh-parameters.xml#ssh-agent-protocol-message-type-numbers>. La longueur peut être nulle, par exemple il existe des messages de type `SSH_AGENT_FAILURE` (type numérique 5) qui n'ont pas de valeur. Le client demande l'ajout d'une clé avec des messages de type `SSH_AGENTC_ADD_IDENTITY` (type numérique 17). La valeur est composée du type de la clé, de la clé elle-même et du commentaire que vous avez indiqué lors de la création de la clé ; si vous utilisez, par exemple, une clé Ed25519, cf. RFC 8709, le type est `ssh-ed25519` (la liste est dans un registre IANA <https://www.iana.org/assignments/ssh-parameters/ssh-parameters.xml#ssh-parameters-19>). Avec OpenSSH, vous trouverez ce commentaire dans `/.ssh/id_ed25519.pub`.

De la même façon, on peut retirer une clé avec les messages de type `SSH_AGENTC_REMOVE_IDENTITY` (type 18) et `SSH_AGENTC_REMOVE_ALL_IDENTITIES` (type 19).

Une fois les clés dans l'agent, le client peut lui demander de signer avec le type de message `SSH_AGENTC_SIGN_REQUEST` (type 13), message qui comprendra les données à signer.

Pour se connecter à l'agent (section 4 du RFC), le client doit utiliser une méthode sûre. Èvidemment pas question d'ouvrir l'agent à tout l'Internet. Sur Unix, la méthode la plus courante est d'utiliser une prise locale. Souvent, elle est trouvée par une variable d'environnement définie lors de la connexion, en général `SSH_AUTH_SOCK`. C'est ce que fait OpenSSH mais ce n'est pas imposé par la norme, qui laisse le choix aux programmes.

Voici un exemple avec OpenSSH :

```
# On lance l'agent (on aurait normalement utilisé eval ou un
# équivalent, pour définir la variable d'environnement) :
% ssh-agent
SSH_AUTH_SOCK=/tmp/ssh-xrLh0tnpVwLF/agent.23934; export SSH_AUTH_SOCK;
SSH_AGENT_PID=23935; export SSH_AGENT_PID;
echo Agent pid 23935;

% ls -l /tmp/ssh-ZzouiZGBumyI/agent.23934
srw------- 1 stephane stephane 0 May  4 18:38 /tmp/ssh-ZzouiZGBumyI/agent.23934

% SSH_AUTH_SOCK=/tmp/ssh-xrLh0tnpVwLF/agent.23934; export SSH_AUTH_SOCK

% ssh -vvv SERVEUR-DISTANT
...
debug1: Next authentication method: publickey
debug3: ssh_get_authentication_socket_path: path '/tmp/ssh-xrLh0tnpVwLF/agent.23934'
debug1: get_agent_identities: bound agent to hostkey
debug1: get_agent_identities: ssh_fetch_identitylist: agent contains no identities

[Et si on ajoute une clé dans l'agent ?]
% ssh-add ~/.ssh/id_ed25519
Enter passphrase for /home/stephane/.ssh/id_ed25519:
Identity added: /home/stephane/.ssh/id_ed25519 (stephane@foobar)

% ssh -vvv SERVEUR-DISTANT
...
debug1: Next authentication method: publickey
debug3: ssh_get_authentication_socket_path: path '/tmp/ssh-xrLh0tnpVwLF/agent.23934'
debug1: get_agent_identities: bound agent to hostkey
debug1: get_agent_identities: agent returned 1 keys
```

Autre possibilité très intéressante de l'agent (section 5), on peut faire suivre les communications sur un canal SSH (un peu comme avec X11). Cela permet, lorsque la machine A se connecte à la machine B puis à la C, d'utiliser les clés de la machine A pour s'authentifier sur la machine C. Cela utilise le mécanisme d'extension à SSH qui avait été normalisé dans le RFC 8308 pour signaler qu'on gère cette possibilité (mais comme le protocole Agent existait avant ce RFC, certains programmes n'annoncent pas cette gestion). La section 9 du RFC rappelle toutefois que cette fonction, si pratique, crée de nouveaux risques puisque elle introduit une relation de confiance transitive. Le RFC exige donc qu'elle ne soit pas activée par défaut.

Le protocole a entrainé la création de cinq nouveaux registres IANA (section 7), dont celui des types de messages <https://www.iana.org/assignments/ssh-parameters/ssh-parameters.xml#ssh-agent-protocol-message-type-numbers> (pour en ajouter un, politique « Examen par un expert », cf. RFC 8126.)

Un petit mot sur la sécurité (section 8 du RFC) puisqu'après tout, SSH est là pour améliorer notre sécurité. L'agent est chargé de garder des clés privées, il est donc très sensible et doit être de confiance. Mais le RFC rappelle aussi que l'accès à l'agent est évidemment très critique et doit être sécurisé (regardez les permissions de la prise dans l'exemple Unix plus haut), le protocole ne prévoyant aucune authentification.

Si on a accès à l'agent, et qu'il a chargé des clés, on peut signer ce qu'on veut et donc s'authentifier auprès de serveurs distants. Par contre, on ne peut pas récupérer de clés privées via le protocole, qui n'a pas d'opération pour cela. Mais comme l'agent garde les clés privées en mémoire, il faut faire attention à ce que personne ne puisse lire cette mémoire. (La page de manuel de OpenSSH est très nette à ce sujet et conseille d'utiliser plutôt la fonction "*ProxyJump*", via le `-J`.)

Ah, et puisque l'agent, lorsqu'il charge une clé, demande la phrase de passe de la clé, il faut aussi qu'il prenne des précautions pour limiter le risque d'une attaque par force brute (quand un attaquant essaie plein de phrases possibles). Par exemple, il peut introduire un délai après une phrase incorrecte.

Le protocole Agent est très ancien et est donc déjà mis en œuvre dans de nombreux programmes, par exemple OpenSSH (depuis 2000 !), PuTTY, Dropbear, Paramiko <https://www.paramiko.org/>, la bibliothèque standard de Go, etc.

Si vous voulez afficher les messages échangés entre le client SSH et l'agent, je ne connais pas l'équivalent de tcpdump ou Wireshark pour cela. Avec OpenSSH, `ssh-agent -d` affiche les connexions mais pas les messages. Sinon, on peut lire les messages échangés avec socat (ici, un exemple pour OpenSSH sur Debian) :

```
[Dans une fenêtre]
% ssh-agent -D

[Dans une autre]
[Copier-coller la première ligne, celle qui définit SSH_AUTH_SOCK]
% mv $SSH_AUTH_SOCK /tmp/real-agent.sock
% socat -x UNIX-LISTEN:$SSH_AUTH_SOCK,fork UNIX-CONNECT:/tmp/real-agent.sock

[Dans une troisième]
[Copier-coller la première ligne, celle qui définit SSH_AUTH_SOCK]
% ssh un-serveur
```

Mais les messages seront bruts, sans formatage. À vous de les décoder. Par exemple, ici,suite à un `ssh-add`, on voit :

```
> 2026/05/11 17:47:13.000145101  length=142 from=1152 to=1293
 00 00 00 8a 11 00 00 00 ...
< 2026/05/11 17:47:13.000146117  length=5 from=10 to=14
 00 00 00 01 06
```

(Pour décoder, référez-vous au RFC, section 3, et au registre IANA `<https://www.iana.org/assignments/ssh-parameters/ssh-parameters.xml#ssh-agent-protocol-message-type-numbers>`) Le premier message (après le ¿) a une longueur de 138 octets (les quatre premiers octets, 0000008A, nous le disent, socat l'affiche mais lui ajoute les quatre octets de la longueur). Le type du message (indiqué par l'octet suivant) est 17, `SSH_AGENTC_ADD_IDENTITY`. L'agent répond (après la ¡) par un message d'un seul octet, de type 6 (`SSH_AGENT_SUCCESS`) et de contenu nul. Si je me connecte en SSH à un serveur, en utilisant la clé qui vient d'être chargée, j'ai :

```
> 2026/05/11 17:59:54.000526321  length=5 from=665 to=669
 00 00 00 01 0b
< 2026/05/11 17:59:54.000526468  length=535 from=5 to=539
 00 00 02 13 0c 00 00 00 ...
> 2026/05/11 17:59:54.000609874  length=1017 from=670 to=1686
 00 00 03 f5 0d 00 00 01 ...
< 2026/05/11 17:59:54.000615719  length=285 from=540 to=824
 00 00 01 19 0e 00 00 01 14 ...
```

Le premier message, très court, est de type 11, `SSH_AGENTC_REQUEST_IDENTITIES`, il obtient une réponse 12 (`SSH_AGENT_IDENTITIES_ANSWER`), puis le client SSH demande une signature avec la clé privée que stocke l'agent (type 13, `SSH_AGENTC_SIGN_REQUEST`) et a une réponse (type 14, `SSH_-AGENT_SIGN_RESPONSE`).

Enfin, le fichier `./PROTOCOL.agent` dans le source de OpenSSH documente les **extensions** d'OpenSSH pour ce protocole agent-client.