Groupe de travail Réseau
Request for Comments : 4253
Catégorie : Sur la voie de la normalisation
Traduction Claude Brière de L'Isle

T. Ylonen, SSH Communications Security Corp
C. Lonvick, éd., Cisco Systems, Inc.

01/01/06

Protocole de couche Transport Secure Shell (SSH)

Statut du présent mémoire
Le présent document spécifie un protocole Internet en cours de normalisation pour la communauté de l'Internet. Il appelle à la discussion et à des suggestions pour son amélioration. Prière de se référer à l'édition actuelle des "Normes officielles des protocoles de l'Internet" (STD 1) pour connaître l'état de normalisation et le statut de ce protocole. La distribution du présent mémoire n'est soumise à aucune restriction.

Résumé
Secure Shell (SSH) est un protocole pour la connexion sécurisée à distance et autres services réseau sécurisés sur un réseau non sûr.

Le présent document décrit le protocole SSH de couche transport, qui fonctionne normalement par dessus TCP/IP. Le protocole peut être utilisé comme base d'un certain nombre de services réseau sécurisés. Il fournit un chiffrement fort, l'authentification du serveur, et la protection d'intégrité. Il peut aussi fournir la compression.

La méthode d'échange de clés, l'algorithme de clé publique, l'algorithme de chiffrement symétrique, l'algorithme d'authentification de message, et l'algorithme de hachage sont tous négociés.

Le présent document décrit aussi la méthode Diffie-Hellman d'échange de clés et l'ensemble minimal d'algorithmes qui sont nécessaires pour mettre en œuvre le protocole SSH de couche transport.

Table des matières

1. Introduction

La couche de transport SSH est un protocole sûr de transport de bas niveau. Il fournit un chiffrement fort, une authentification cryptographique d'hôte, et la protection de l'intégrité.

Dans ce niveau de protocole, l'authentification est fondée sur l'hôte ; ce protocole n'effectue pas d'authentification de l'utilisateur. Un protocole de niveau supérieur peut être conçu par dessus ce protocole pour l'authentification d'utilisateur.

Le protocole a été conçu pour être simple et flexible afin de permettre la négociation des paramètres, et pour minimiser le nombre d'allers-retours. La méthode d'échange de clés, l'algorithme de clé publique, l'algorithme de chiffrement symétrique, l'algorithme d'authentification de message, et l'algorithme de hachage sont tous négociés. Il est prévu que dans la plupart des environnements, seuls deux allers-retours seront nécessaires pour l'échange de clé complet, l'authentification de serveur, la demande de service, et la notification d'acceptation de la demande de service. Le pire cas est de trois allers-retours.

2. Contributeurs

Les contributeurs majeurs d'origine de cet ensemble de documents ont été : Tatu Ylonen, Tero Kivinen, Timo J. Rinne, Sami Lehtinen (tous de SSH Communications Security Corp), et Markku-Juhani O. Saarinen (Université de Jyvaskyla). Darren Moffat était l'éditeur d'origine de cet ensemble de documents et il a aussi fait des contributions très substantielles.

De nombreuses personnes ont contribué au développement de ce document ai fil des ans. Parmi ceux qui doivent être remerciés sont Mats Andersson, Ben Harris, Bill Sommerfeld, Brent McClure, Niels Moller, Damien Miller, Derek Fawcus, Frank Cusack, Heikki Nousiainen, Jakob Schlyter, Jeff Van Dyke, Jeffrey Altman, Jeffrey Hutzelman, Jon Bright, Joseph Galbraith, Ken Hornstein, Markus Friedl, Martin Forssen, Nicolas Williams, Niels Provos, Perry Metzger, Peter Gutmann, Simon Josefsson, Simon Tatham, Wei Dai, Denis Bider, der Mouse et Tadayoshi Kohno. Faire la liste de leurs noms ici ne signifie pas qu'ils souscrivent entièrement à ce document, mais qu'ils y ont contribué.

3. Conventions utilisées dans ce document

Tous les documents qui se rapportent aux protocoles SSH doivent utiliser les mots clés "DOIT", "NE DOIT PAS", "EXIGE", "DEVRA", "NE DEVRA PAS", "DEVRAIT", "NE DEVRAIT PAS", "RECOMMANDE", "PEUT", et "FACULTATIF" pour décrire les exigences. Ces mots clés sont à interpréter comme décrit dans la [RFC2119].

Les mots clés "UTILISATION PRIVÉE", "ALLOCATION HIÉRARCHIQUE ", "PREMIER ENTRÉ PREMIER SERVI", "RÉVISION D'EXPERT", "SPÉCIFICATION EXIGÉE", "APPROBATION IESG", "CONSENSUS IETF", et "ACTION DE NORMALISATION" qui apparaissent dans le présent document lorsqu'ils sont utilisés pour décrire une allocation d'espace de nom sont à interpréter comme décrit dans la [RFC2434].

Les champs de protocole et les valeurs possibles pour les remplir sont définies dans cet ensemble de documents. Les champs de protocole seront définis dans les définitions de message. Par exemple, SSH_MSG_CHANNEL_DATA est défini comme suit.

octet SSH_MSG_CHANNEL_DATA

uint32 canal de réception
chaîne données

Tout au long de ces documents, lorsque des champs sont mentionnés, ils vont apparaître avec des guillemets simples. Lorsque des valeurs pour remplir ces champs sont mentionnées, elles vont apparaître avec des guillemets doubles. En utilisant l'exemple ci-dessus, des valeurs possibles pour 'données' sont "foo" et "bar".

4. Établissement de connexion

SSH fonctionne sur tout transport transparent binaire à 8 bits francs. Le transport sous-jacent DEVRAIT protéger contre les erreurs de transmission, car de telles erreurs causent la terminaison de la connexion SSH.

Le client prend l'initiative de la connexion.

4.1 Utilisation sur TCP/IP

Dans une utilisation sur TCP/IP, le serveur écoute normalement les connexions sur l'accès 22. Ce numéro d'accès a été enregistré auprès de l'IANA, et a été officiellement alloué à SSH.

4.2 Échange de version de protocole

Lorsque la connexion a été établie, les deux côtés DOIVENT envoyer une chaîne d'identification. Cette chaîne d'identification DOIT être :

SSH-protoversion-softwareversion SP comments CR LF

Comme le protocole qu'on définit dans cet ensemble de documents est la version 2.0, la 'protoversion' DOIT être "2.0". La chaîne 'comments' est FACULTATIVE. Si la chaîne 'comments' est incluse, un caractère 'espace' (noté ci-dessus SP, ASCII 32) DOIT séparer les chaînes 'softwareversion' et 'comments'. L'identification DOIT être terminée par un seul caractère Retour chariot (CR) et un seul Saut à la ligne (LF) (respectivement ASCII 13 et 10). Les mises en œuvre qui souhaitent maintenir la compatibilité avec des versions plus anciennes, non documentées de ce protocole peuvent vouloir traiter la chaîne d'identification sans attendre la présence du caractère retour chariot pour des raisons décrites à la Section 5 de ce document. Le caractère nul NE DOIT PAS être envoyé. La longueur maximum de la chaîne est de 255 caractères, y compris le retour chariot et le saut à la ligne.

La partie de la chaîne d'identification qui précède le retour chariot et le saut à la ligne est utilisée dans l'échange de clé Diffie-Hellman (voir la Section 8).

Le serveur PEUT envoyer d'autres lignes de données avant d'envoyer la chaîne version. Chaque ligne DEVRAIT être terminée par un retour chariot et saut à la ligne. De telles lignes NE DOIVENT PAS commencer par "SSH-", et DEVRAIENT être codée en UTF-8 ISO-10646 [RFC3629] (le langage n'est pas spécifié). Les clients DOIVENT être capables de traiter de telles lignes. De telles lignes PEUVENT être ignorées en silence, ou PEUVENT être affichées à l'utilisateur client. Si elles sont affichées, le filtrage des caractères de contrôle, tel qu'exposé dans [RFC4251], DEVRAIT être utilisé. La principale utilisation de ce dispositif est de permettre aux enveloppes TCP d'afficher un message d'erreur avant de déconnecter.

Les chaînes 'protoversion' et 'softwareversion' DOIVENT toutes deux consister en caractères US-ASCII imprimables, à l'exception des caractères d'espace et du signe moins (-). La chaîne 'softwareversion' est principalement utilisée pour déclencher les extensions de compatibilité et pour indiquer les capacités d'une mise en œuvre. La chaîne 'comments' DEVRAIT contenir des informations supplémentaires qui pourraient être utiles pour résoudre des problèmes d'usager. À ce titre, un exemple de chaîne d'identification valide est :

SSH-2.0-billsSSH_3.6.3q3<CR><LF>

Cette chaîne d'identification ne contient pas la chaîne facultative 'comments' et se termine donc par un CR et un LF immédiatement après la chaîne 'softwareversion'.

L'échange de clés va commencer immédiatement après l'envoi de cet identifiant. Tous les paquets qui suivent la chaîne d'identification DEVRONT utiliser le protocole de paquet binaire, qui est décrit à la Section 6.

5. Compatibilité avec les anciennes versions SSH

Comme indiqué plus haut, la 'protoversion' spécifiée pour ce protocole est "2.0". Les versions antérieures de ce protocole n'ont pas été formellement documentées, mais il est bien connu qu'elles utilisent une 'protoversion' de "1.x" (par exemple, "1.5" ou "1.3"). Au moment de la rédaction du présent document, de nombreuses mises en œuvre de SSH utilisent la version de protocole 2.0, mais il est bien connu qu'il y a encore des appareils qui utilisent les versions précédentes. Durant la période de transition, il est important d'être capable de travailler d'une façon compatible avec les clients et serveurs SSH installés qui utilisent de plus anciennes versions du protocole. Les informations de cette section ne sont pertinentes que pour les mises en œuvre qui prennent en charge la compatibilité avec les versions SSH 1.x. Pour ceux que cela intéresse, la seule documentation connue du protocole 1.x est contenue dans les fichiers README qui sont livrés avec le code source [ssh-1.2.30].

5.1. Vieux client, nouveau serveur

Les mises en œuvre de serveur PEUVENT prendre en charge un fanion de compatibilité configurable qui active la compatibilité avec les anciennes versions. Lorsque ce fanion est mis *(à 1)*, le serveur DEVRAIT identifier sa 'protoversion' à "1.99". Les clients qui utilisent le protocole 2.0 DOIVENT être capables d'identifier cela comme identique à "2.0". Dans ce mode, le serveur NE DEVRAIT PAS envoyer le caractère retour chariot (ASCII 13) après la chaîne d'identification.

Dans le mode compatibilité, le serveur NE DEVRAIT PAS envoyer d'autres données après l'envoi de sa chaîne d'identification jusqu'à ce qu'il ait reçu une chaîne d'identification du client. Le serveur peut alors déterminer si le client utilise un vieux protocole, et peut revenir au vieux protocole si nécessaire. En mode compatibilité, le serveur NE DOIT PAS envoyer de données supplémentaires avant la chaîne d'identification.

Lorsque la compatibilité avec les vieux clients n'est pas nécessaire, le serveur PEUT envoyer ses données d'échange de clé initial immédiatement après la chaîne d'identification.

5.2 Nouveau client, vieux serveur

Comme le nouveau client PEUT immédiatement envoyer des données supplémentaires après sa chaîne d'identification (avant de recevoir la chaîne d'identification du serveur), le vieux protocole peut déjà être corrompu lorsque le client apprend que le serveur est d'un ancien protocole. Lorsque cela arrive, le client DEVRAIT clore la connexion au serveur, et se reconnecter en utilisant le vieux protocole.

5.3 Taille de paquet et redondance

Certains lecteurs vont s'inquiéter de l'augmentation de la taille des paquets due aux nouveaux en-têtes, bourrage et code d'authentification de message (MAC). La taille de paquet minimum est de l'ordre de 28 octets (selon les algorithmes négociés). L'augmentation est négligeable pour les grands paquets, mais très significative pour les paquets d'un octet (sessions de type telnet). Il y a cependant plusieurs facteurs qui font que dans presque tous les cas, ce problème n'en est pas un :

- La taille minimum d'un en-tête TCP/IP est de 32 octets. Et donc, l'augmentation est en fait de 33 à 51 octets (en gros).
- La taille minimum du champ données d'un paquet Ethernet est 46 octets [RFC0894]. Et donc, l'augmentation n'est pas de plus de 5 octets. Quand on prend les en-têtes Ethernet, l'augmentation est de moins de 10 pour cent.
- La fraction totale des données de type telnet dans l'Internet est négligeable, même avec une augmentation de la taille des paquets.

Le seul environnement où l'augmentation de la taille de paquet peut vraisemblablement avoir un effet significatif est PPP [RFC1661] sur des lignes modem lentes (PPP compresse les en-têtes TCP/IP, amplifiant l'augmentation de la taille de paquet). Cependant, avec les modems modernes, le temps nécessaire au transfert est de l'ordre de 2 millisecondes, ce qui est nettement supérieur à la vitesse de frappe d'une personne.

Il y a aussi des problèmes en ce qui concerne la taille maximum de paquet. Pour minimiser les délais de mise à jour d'écran, on ne doit pas vouloir de paquets trop grands dans les sessions interactives. La taille maximum de paquet est négociée séparément pour chaque canal.

6. Protocole de paquet binaire

Chaque paquet est dans le format suivant :

```
uint32      longueur_de_paquet
octet       longueur_de_bourrage
octet[n1]   charge_utile ; n1 = longueur_de_paquet - longueur_de_bourrage - 1
octet[n2]   bourrage_aléatoire ; n2 = longueur_de_bourrage
octet[m]    mac (code d'authentification de message - MAC) ; m = longueur_de_mac
```

longueur_de_paquet : longueur du paquet en octets, sans inclure le 'mac' ou le champ 'longueur_de_paquet' lui-même.

longueur_de_bourrage : longueur de 'bourrage aléatoire' (octets).

charge_utile : le contenu utile du paquet. Si la compression a été négociée, ce champ est compressé. Initialement, la compression DOIT être "aucune".

bourrage_aléatoire : bourrage de longueur arbitraire, tel que la longueur totale de (longueur_de_paquet || longueur_de_bourrage || charge_utile || bourrage aléatoire) soit un multiple de la taille de bloc de chiffrement ou de 8, selon ce qui est le plus grand. Il DOIT y avoir au moins quatre octets de bourrage. Le bourrage DEVRAIT consister en octets au hasard. La quantité maximum de bourrage est 255 octets.

mac : code d'authentification de message. Si l'authentification de message a été négociée, ce champ contient les octets du MAC. Initialement, l'algorithme MAC DOIT être "aucun".

Noter que la longueur de l'enchaînement de 'longueur_de_paquet', 'longueur_de_bourrage', 'charge_utile', et 'bourrage_aléatoire' DOIT être un multiple de la taille de bloc de chiffrement ou de 8, selon ce qui est le plus grand. Cette contrainte DOIT être mise en application, même si des chiffrements de flux sont utilisés. Noter que le champ 'longueur_de_paquet' est aussi chiffré, et son traitement demande des soins particuliers lors de l'envoi ou la réception des paquets. Noter aussi que l'insertion de quantités variables de 'bourrage aléatoire' peut aider à déjouer l'analyse de trafic.

La taille minimum d'un paquet est de 16 octets (ou la taille du bloc de chiffrement, selon ce qui est le plus grand) (plus 'mac'). Les mises en œuvre DEVRAIENT déchiffrer la longueur après la réception des huit premiers octets d'un paquet (ou la taille du bloc de chiffrement, selon ce qui est le plus grand).

6.1 Taille maximum de paquet

Toutes les mises en œuvre DOIVENT être capables de traiter les paquets avec une longueur de charge utile non compressée de 32 768 octets ou moins et une taille de paquet totale de 35 000 octets ou moins (incluant 'longueur_de_paquet', 'longueur_de_bourrage', 'charge_utile', 'bourrage-aléatoire', et 'mac'). Le maximum de 35 000 octets est une valeur choisie arbitrairement supérieure à la longueur non compressée notée ci-dessus. Les mises en œuvre DEVRAIENT accepter de plus longs paquets, lorsqu'ils peuvent être nécessaires. Par exemple, si une mise en œuvre veut envoyer un très grand nombre de certificats, les plus grands paquets PEUVENT être envoyés si la chaîne d'identification indique que l'autre partie est capable de les traiter. Cependant, les mises en œuvre DEVRAIENT vérifier que la longueur de paquet est raisonnable afin que la mise en œuvre évite des attaques de déni de service et/ou de débordement de mémoire tampon.

6.2 Compression

Si la compression a été négociée, le champ 'charge_utile' (et lui seul) sera compressé en utilisant l'algorithme négocié. Le champ 'longueur_de_paquet' et le 'mac' seront calculés à partir de la charge utile compressée. Le chiffrement sera fait après la compression.

La compression PEUT être à état plein, selon la méthode. La compression DOIT être indépendante pour chaque direction, et les mises en œuvre DOIVENT permettre un choix indépendant de l'algorithme pour chaque direction. En pratique cependant, il est RECOMMANDÉ que la méthode de compression soit la même dans les deux directions.

Les méthodes de compression suivantes sont actuellement définies :

aucune	EXIGÉ	aucune compression
zlib	FACULTATIF	compression ZLIB (LZ77)

La compression "zlib" est décrite dans la [RFC1950] et dans la [RFC1951]. Le contexte de compression est initialisé après chaque échange de clé, et est passé d'un paquet au suivant, avec seulement une purge partielle effectuée à la fin de chaque paquet. Une purge partielle signifie que le bloc actuellement compressé se termine et que toutes les données seront en sortie. Si le bloc en cours n'est pas un bloc mémorisé, un ou plusieurs blocs vides sont ajoutés après le bloc en cours pour s'assurer qu'il y a au moins 8 bits, en comptant depuis le début du code de fin de bloc du bloc en cours jusqu'à la fin de la charge utile du paquet.

Des méthodes supplémentaires peuvent être définies comme spécifié dans les [RFC4250] et [RFC4251].

6.3 Chiffrement

Un algorithme de chiffrement et une clé seront négociés durant l'échange de clé. Lorsque le chiffrement est activé, les champs longueur_de_paquet, longueur_de_bourrage, charge_utile, et bourrage de chaque paquet DOIVENT être chiffrés avec l'algorithme en question.

Les données chiffrées dans tous les paquets envoyés dans une direction DEVRAIENT être considérés comme un seul flux de données. Par exemple, les valeurs d'initialisation DEVRAIENT être passées de la fin d'un paquet au début du paquet suivant. Tous les chiffrements DEVRAIENT utiliser des clés d'une longueur effective de 128 bits ou plus.

Les chiffrements dans chaque direction DOIVENT fonctionner indépendamment l'un de l'autre. Les mises en œuvre DOIVENT permettre que l'algorithme pour chaque direction soit choisi de façon indépendante, si plusieurs algorithmes sont permis par la politique locale. En pratique cependant, il est RECOMMANDÉ que le même algorithme soit utilisé dans les deux directions.

Les chiffrements suivants sont actuellement définis :

3des-cbc	EXIGÉ	3DES à trois clés en mode CBC
blowfish-cbc	FACULTATIF	Blowfish en mode CBC
twofish256-cbc	FACULTATIF	Twofish en mode CBC, avec une clé de 256 bits
twofish-cbc	FACULTATIF	alias pour "twofish256-cbc" (ne figure que pour des raisons historiques)
twofish192-cbc	FACULTATIF	Twofish avec clé de 192 bits
twofish128-cbc	FACULTATIF	Twofish avec clé de 128 bits
aes256-cbc	FACULTATIF	AES en mode CBC, avec clé de 256 bits
aes192-cbc	FACULTATIF	AES avec clé de 192 bits
aes128-cbc	RECOMMANDÉ	AES avec clé de 128 bits
serpent256-cbc	FACULTATIF	Serpent en mode CBC, avec clé de 256 bits
serpent192-cbc	FACULTATIF	Serpent avec clé de 192 bits
serpent128-cbc	FACULTATIF	Serpent avec clé de 128 bits
arcfour	FACULTATIF	chiffrement de flux ARCFOUR avec clé de 128 bits
idea-cbc	FACULTATIF	IDEA en mode CBC
cast128-cbc	FACULTATIF	CAST-128 en mode CBC
aucun	FACULTATIF	aucun chiffrement ; NON RECOMMANDÉ

Le chiffrement "3des-cbc" est le triple-DES à trois clés (chiffrement-déchiffrement-chiffrement), où les huit premiers octets de la clé sont utilisés pour le premier chiffrement, les huit octets suivants pour le déchiffrement, et les huit octets suivants pour le chiffrement final. Cela exige 24 octets de données de clé (dont 168 bits sont en fait utilisés). Pour mettre en œuvre le mode CBC, le chaînage externe DOIT être utilisé (c'est-à-dire qu'il n'y a qu'une valeur d'initialisation). C'est un chiffrement de bloc avec des blocs de huit octets. Cet algorithme est défini dans [FIPS-46-3]. Noter que dans la mesure où cet algorithme a seulement une longueur effective de clé de 112 bits ([SCHNEIER]) il ne satisfait pas à la spécification que les algorithmes de chiffrement de SSH devraient utiliser des clés de 128 bits ou plus. Cependant, cet algorithme est toujours EXIGÉ pour des raisons historiques ; pour l'essentiel, toutes les mises en œuvre connues au moment de la rédaction de ce mémoire prenaient en charge cet algorithme, et il est couramment utilisé parce qu'il est l'algorithme interopérable fondamental. À l'avenir, il est prévu qu'un autre algorithme, plus fort, deviendra tellement prévalent et universel que l'utilisation du "3des-cbc" sera déconseillée par une autre ACTION DE NORMALISATION.

Le chiffrement "blowfish-cbc" est Blowfish en mode CBC, avec des clés de 128 bits [SCHNEIER]. C'est un chiffrement de bloc avec des blocs de huit octets.

Le chiffrement "twofish-cbc" ou "twofish256-cbc" est Twofish en mode CBC, avec des clés de 256 bits comme décrit dans [TWOFISH]. C'est un chiffrement de bloc avec des blocs de 16 octets.

Le chiffrement "twofish192-cbc" est le même que ci-dessus, mais avec une clé de 192 bits.

Le chiffrement "twofish128-cbc" est le même que ci-dessus, mais avec une clé de 128 bits.

Le chiffrement "aes256-cbc" est AES (Norme de chiffrement évolué) [FIPS-197], en mode CBC. Cette version utilise une clé de 256 bits.

Le chiffrement "aes192-cbc" est le même que ci-dessus, mais avec une clé de 192 bits.

Le chiffrement "aes128-cbc" est le même que ci-dessus, mais avec une clé de 128 bits.

Le chiffrement "serpent256-cbc" en mode CBC, avec une clé de 256 bits, comme décrit dans le document Serpent AES.

Le chiffrement"serpent192-cbc" est le même que ci-dessus, mais avec une clé de 192 bits.

Le chiffrement"serpent128-cbc" est le même que ci-dessus, mais avec une clé de 128 bits.

Le chiffrement "arcfour" est le chiffrement de flux Arcfour avec des clés de 128 bits. Le chiffrement Arcfour est estimé compatible avec le chiffrement RC4 [SCHNEIER]. Arcfour (et RC4) a des problèmes avec les clés faibles, et devrait être utilisé avec des précautions.

Le chiffrement "idea-cbc" est le chiffrement IDEA en mode CBC [SCHNEIER].

Le chiffrement "cast128-cbc" est le chiffrement CAST-128 en mode CBC avec une clé de 128 bits [RFC2144].

L'algorithme "aucun" spécifie qu'aucun chiffrement n'est effectué. Noter que cette méthode ne fournit aucune protection de la confidentialité, et qu'elle est NON RECOMMANDÉE. Certaines fonctionnalités (par exemple, d'authentification de mot de passe) peuvent être désactivées pour des raisons de sécurité si ce chiffrement est choisi.

Des méthodes supplémentaires peuvent être définies comme spécifié dans les [RFC4250] et [RFC4251].

6.4 Intégrité des données

L'intégrité des données est protégée en incluant avec chaque paquet un MAC qui est calculé à partir d'un secret partagé, du numéro de séquence du paquet, et du contenu du paquet.

L'algorithme d'authentification de message et la clé sont négociés durant l'échange de clé. Au départ, aucun MAC ne sera activé, et sa longueur DOIT être zéro. Après l'échange de clé, le 'mac' pour l'algorithme de MAC choisi sera calculé avant le chiffrement à partir de la concaténation des données du paquet :

mac = MAC(clé, numéro_de_séquence || paquet_non_chiffré)

où paquet_non_chiffré est le paquet entier sans 'mac' (les champs de longueur, 'charge_utile' et 'bourrage aléatoire'), et numéro_de_séquence est un numéro de séquence de paquet implicite représenté par uint32. Le numéro_de_séquence est initialisé à zéro pour le premier paquet, et est incrémenté après chaque paquet (sans considération de l'utilisation du chiffrement ou du MAC). Il n'est jamais remis à zéro, même si les clés/algorithmes sont renégociés plus tard. Il revient à zéro tous les 2^32 paquets. Le numéro_de_séquence de paquet lui-même n'est pas inclus dans le paquet envoyé sur le réseau.

Les algorithmes de MAC pour chaque direction DOIVENT fonctionner de façon indépendante, et les mises en œuvre DOIVENT permettre de choisir l'algorithme de façon indépendante dans les deux directions. En pratique cependant, il est RECOMMANDÉ que le même algorithme soit utilisé dans les deux directions.

La valeur de 'mac' résultant de l'algorithme de MAC DOIT être transmise sans chiffrement à la dernière partie du paquet. Le nombre d'octets de 'mac' dépend de l'algorithme choisi.

Les algorithmes de MAC actuellement définis sont :

hmac-sha1	EXIGÉ	HMAC-SHA1 (longueur du résumé = longueur de clé = 20)
hmac-sha1-96	RECOMMANDÉ	96 premiers bits de HMAC-SHA1 (longueur résumé = 12, longueur de clé = 20)
hmac-md5	FACULTATIF	HMAC-MD5 (longueur de résumé = longueur de clé = 16)
hmac-md5-96	FACULTATIF	96 premiers bits de HMAC-MD5 (longueur résumé = 12, longueur de clé = 16)
aucun	FACULTATIF	pas de MAC ; NON RECOMMANDÉ

Les algorithmes "hmac-*" sont décrits dans la [RFC2104]. Les MAC "*-n" utilisent seulement les n premiers bits de la valeur résultante.

SHA-1 est décrit dans [FIPS-180-2] et MD5 est décrit dans la [RFC1321].

Des méthodes supplémentaires peuvent être définies, comme spécifié dans les [RFC4250] et [RFC4251].

6.5 Méthodes d'échange de clé

La méthode d'échange de clé spécifie comment les clés de session à utilisation unique sont générées pour le chiffrement et pour l'authentification, et comment l'authentification du serveur est faite.

Deux méthodes d'échange de clé EXIGÉES ont été définies :

diffie-hellman-group1-sha1 EXIGÉ
diffie-hellman-group14-sha1 EXIGÉ

Ces méthodes sont décrites à la Section 8.

Des méthodes supplémentaires peuvent être définies comme spécifié dans la [RFC4250]. Le nom "diffie-hellman-group1-sha1" est utilisé pour une méthode d'échange de clé qui utilise un groupe Oakley, tel que défini dans la [RFC2409]. SSH entretient son propre espace d'identifiants de groupe qui est logiquement distinct de Oakley [RFC2412] et de IKE ; cependant, pour un groupe supplémentaire, le groupe de travail a adopté le numéro alloué par la [RFC3526], utilisant diffie-hellman-group14-sha1 comme nom du second groupe défini. Les mises en œuvre devraient traiter ces noms comme des identifiants opaques et ne devraient supposer aucune relation entre les groupes utilisés par SSH et les groupes définis pour IKE.

6.6 Algorithmes de clé publique

Le présent protocole a été conçu pour fonctionner avec la plupart des formats, codages, et algorithmes (signature et/ou chiffrement) de clé publique.

Il y a plusieurs aspects dans la définition d'un type de clé publique :

- Format de clé : comment la clé est codée et comment sont représentés les certificats. Les modules de clé dans le présent protocole PEUVENT contenir des certificats en plus des clés.
- Algorithmes de signature et/ou chiffrement. Certains types de clés peuvent ne pas prendre en charge à la fois la signature et le chiffrement. L'usage de la clé devrait être encadré par les déclarations de politique (par exemple, dans les certificats). Dans ce cas, les différents types de clé DEVRAIENT être définis pour les différentes solutions de remplacement de politiques.
- Codage des signatures et/ou données chiffrées. Cela inclut, sans s'y limiter, le bourrage, l'ordre des octets, et les formats de données.

Les formats de clé publique et/ou de certificat suivants sont définis actuellement :

ssh-dss	EXIGÉ	signature	clé DSS brute
ssh-rsa	RECOMMANDÉ	signature	clé RSA brute
pgp-sign-rsa	FACULTATIF	signature	certificats OpenPGP (clé RSA)
pgp-sign-dss	FACULTATIF	signature	certificats OpenPGP (clés DSS)

Des types de clé supplémentaires peuvent être définis, comme spécifié dans les [RFC4250] et [RFC4251].

Le type de clé DOIT toujours être explicitement connu (à partir de la négociation d'algorithme ou de quelque autre source). Il n'est pas normalement inclus dans le module de clé.

Les certificats et les clés publiques sont codés comme suit :

chaîne	certificat ou identifiant de format de clé publique
octet[n]	données de clé/certificat

La partie certificat peut être une chaîne de longueur zéro, mais une clé publique est exigée. C'est la clé publique qui sera utilisée pour l'authentification. La séquence de certificat contenue dans le module de certificat peut être réutilisée pour fournir l'autorisation.

Les formats de clé publique/certificat qui ne spécifient pas explicitement un identifiant de format de signature DOIVENT utiliser l'identifiant de format de clé publique/certificat comme identifiant de signature.

Les signatures sont codées comme suit :

chaîne	identifiant de format de signature (comme spécifié par le format de clé publique/certificat)
octet[n]	module de signature dans le codage spécifique du format.

Le format de clé "ssh-dss" a le codage spécifique suivant :

chaîne	"ssh-dss"
mpint	p
mpint	q
mpint	g
mpint	y

Ici, les paramètres 'p', 'q', 'g', et 'y' forment le module de clé de signature.

La signature et la vérification en utilisant ce format de clé sont faites conformément à la norme de signature numérique *(Digital Signature Standard)* [FIPS-186-2] en utilisant le hachage SHA-1 [FIPS-180-2].

La signature résultante est codée comme suit :

chaîne	"ssh-dss"
chaîne	dss_signature_blob

La valeur pour 'dss_signature_blob' est codée comme une chaîne contenant r, suivi de s (qui sont des entiers de160 bits, sans longueurs ou bourrage, non signés, et dans l'ordre des octets du réseau).

Le format de clé "ssh-rsa" a le codage spécifique suivant :

chaîne	"ssh-rsa"
mpint	e
mpint	n

Ici les paramètres 'e' et 'n' forment le module de clé de signature.

La signature et la vérification en utilisant ce format de clé sont faites conformément au schéma RSASSA-PKCS1-v1_5 de la [RFC3447] en utilisant le hachage SHA-1.

La signature résultante est codée comme suit :

chaîne	"ssh-rsa"
chaîne	rsa_signature_blob

La valeur pour 'rsa_signature_blob' est codée comme une chaîne contenant s (qui est un entier, sans longueurs ni bourrage, non signé, et dans l'ordre des octets du réseau).

La méthode "pgp-sign-rsa" indique que les certificats, la clé publique, et la signature sont en format binaire compatible OpenPGP de la ([RFC2440]). Cette méthode indique que la clé est une clé RSA.

Le "pgp-sign-dss" est comme ci-dessus, mais indique que la clé est une clé DSS.

7. Échange de clés

L'échange de clé (kex, *key exchange*) commence par l'envoi par chaque côté de la liste des noms des algorithmes pris en charge. Chaque côté a un algorithme préféré dans chaque catégorie, et on suppose que la plupart des mises en œuvre vont à tout moment, utiliser le même algorithme préféré. Chaque côté PEUT faire une conjecture sur l'algorithme qu'utilise l'autre côté, et PEUT envoyer un paquet d'échange de clé initial conforme à l'algorithme, s'il est approprié pour la méthode préférée.

La conjecture est considérée comme fausse si :
- o l'algorithme de kex et/ou l'algorithme de clé d'hôte est conjecturé à tort (le serveur et le client ont un algorithme préféré différent), ou
- o si on ne peut se mettre d'accord sur aucun des autres algorithmes (la procédure est définie au paragraphe 7.1).

Autrement, la conjecture est considérée comme juste, et le paquet envoyé à tout hasard DOIT être traité comme premier paquet de l'échange de clé.

Cependant, si la conjecture était fausse, et qu'un paquet a été envoyé à tout hasard par l'une ou l'autre des parties, ou les deux, de tels paquets DOIVENT être ignorés (même si l'erreur de conjecture n'aurait pas affecté le contenu du ou des paquets initiaux, et le côté approprié DOIT envoyer le paquet initial correct.

Une méthode d'échange de clé utilise l'authentification explicite de serveur si les messages d'échange de clé incluent une signature ou autre preuve de l'authenticité du serveur. Une méthode d'échange de clé utilise l'authentification implicite du serveur si, afin de prouver son authenticité, le serveur a aussi à prouver qu'il connaît le secret partagé, K, en envoyant un message et un MAC correspondant que le client peut vérifier.

La méthode d'échange de clé définie dans le présent document utilise l'authentification explicite du serveur. Cependant, les méthodes d'échange de clé avec authentification implicite du serveur PEUVENT être utilisées avec ce protocole. Après un échange de clé avec authentification implicite du serveur, le client DOIT attendre une réponse à son message de demande de service avant d'envoyer d'autres données.

7.1 Négociation d'algorithme

L'échange de clé commence par l'envoi par chaque côté du paquet suivant :

octet	SSH_MSG_KEXINIT
octet[16]	cookie (octets aléatoires)
liste-de-nom	algorithmes_d'échange_de_clé
liste-de-nom	algorithmes_de_clé_hôte_serveur
liste-de-nom	algorithmes_de_chiffrement_client_à_serveur
liste-de-nom	algorithmes_de_chiffrement_serveur_à_client
liste-de-nom	algorithmes_mac_client_à_serveur
liste-de-nom	mac_algorithmes_serveur_to_client
liste-de-nom	algorithmes_de_compression_client_à_serveur
liste-de-nom	algorithmes_de_compression_serveur_à_client
liste-de-nom	langages_client_à_serveur
liste-de-nom	langages_serveur_à_client
booléen	premier_paquet_kex_suit
uint32	0 (réservé pour extension future)

Chacune des listes de nom d'algorithme DOIT être une liste des noms d'algorithme séparés par des virgules (voir "Dénomination des algorithmes" dans la [RFC4251] et les informations additionnelles dans la [RFC4250]). Chaque algorithme pris en charge (admis) DOIT figurer sur la liste dans l'ordre des préférences, de la plus grande à la moindre.

Le premier algorithme de chaque liste de noms DOIT être l'algorithme préféré (supposé). Chaque liste de noms DOIT contenir au moins un nom d'algorithme.

cookie
Le 'cookie' DOIT être une valeur aléatoire générée par l'envoyeur. Son but est de rendre impossible à l'un et l'autre côté de pleinement déterminer les clés et l'identifiant de session.

algorithmes_d'échange_de_clé
Les algorithmes d'échange de clé ont été définis plus haut. Le premier algorithme DOIT être le préféré (et celui qui est conjecturé). Si les deux côtés font la même conjecture, cet algorithme DOIT être utilisé. Autrement, l'algorithme suivant DOIT être utilisé pour choisir une méthode d'échange de clé :
Itérer parmi les algorithmes d'échange de clé du client, un à la fois. Choisir le premier algorithme qui satisfait aux conditions suivantes:
+ le serveur accepte aussi l'algorithme,
+ si l'algorithme requiert une clé d'hôte à capacité de chiffrement, il y a un algorithme capable de chiffrement sur la liste algorithmes_clé_hôte du serveur qui est aussi acceptée par le client, et
+ si l'algorithme requiert une clé d'hôte capable de signature, il y a un algorithme capable de signature sur la liste algorithmes_clé_serveur_hôte du serveur qui est aussi accepté par le client.

Si aucun algorithme satisfaisant toutes ces conditions ne peut être trouvé, la connexion échoue, et les deux côtés DOIVENT se déconnecter.

algorithmes_de_clé_hôte_serveur
Une liste des noms des algorithmes pris en charge pour la clé d'hôte du serveur. Le serveur fait la liste des algorithmes pour lesquels il a des clés d'hôte ; le client fait la liste des algorithmes qu'il est d'accord pour accepter. Il PEUT y avoir plusieurs clés d'hôte pour un hôte, éventuellement avec des algorithmes différents.

Certaines clés d'hôte peuvent ne pas accepter à la fois les signatures et le chiffrement (cela peut être déterminé à partir de l'algorithme) et donc, toutes les clés d'hôte ne sont pas valides pour toutes les méthodes d'échange de clé.

Le choix de l'algorithme dépend du fait que l'algorithme d'échange de clé choisi exige une clé d'hôte capable de signature ou capable de chiffrement. Il DOIT être possible de le déterminer à partir du nom de l'algorithme de clé publique. Le premier algorithme sur la liste de noms du client qui satisfait aux exigences et est aussi accepté par le serveur DOIT être choisi. Si un tel algorithme n'existe pas, les deux côtés DOIVENT déconnecter.

algorithmes_de_chiffrement
Une liste de noms d'algorithmes de chiffrement symétriques acceptables (aussi appelés chiffres) dans l'ordre de préférence. L'algorithme de chiffrement choisi pour chaque direction DOIT être le premier algorithme sur la liste de noms du client qui est aussi sur la liste de noms du serveur. Si un tel algorithme n'existe pas, les deux côtés DOIVENT déconnecter. Noter que "aucun" doit être explicitement sur la liste si on veut que cela soit acceptable. La liste des noms d'algorithmes définis figure au paragraphe 6.3.

algorithmes_mac
Une liste de noms d'algorithmes de MAC acceptables dans l'ordre de préférence. L'algorithme de MAC choisi DOIT être le premier algorithme sur la liste des noms du client qui est aussi sur la liste des noms du serveur. Si un tel algorithme n'existe pas, les deux côtés DOIVENT déconnecter. Noter que "aucun" doit être explicitement sur la liste si on veut que cela soit acceptable. La liste des noms d'algorithmes de MAC figure au paragraphe 6.4.

algorithmes_de_compression
Une liste de noms d'algorithmes de compression algorithmes dans l'ordre de préférence. acceptables dans l'ordre de préférence. L'algorithme de compression choisi DOIT être le premier algorithme sur la liste des noms du client qui est aussi sur la liste des noms du serveur. Si un tel algorithme n'existe pas, les deux côtés DOIVENT déconnecter. Noter que "aucun" doit être explicitement sur la liste si on veut que cela soit acceptable. La liste des noms d'algorithmes de compression figure au paragraphe 6.2.

langages
C'est une liste des noms d'étiquettes de langage dans l'ordre de préférence [RFC3066]. Les deux parties PEUVENT ignorer cette liste de noms. Si il n'y a pas de préférence de langage, cette liste de noms DEVRAIT être vide comme défini à la Section 5 de [RFC4251]. Les étiquettes de langage NE DEVRAIENT PAS être présentes si la partie qui envoie ne les estime pas nécessaires.

premier_paquet_kex_suit
Indique si un paquet d'échange de clé conjecturée suit. Si un paquet de conjecture va être envoyé, cela DOIT être VRAI. Si aucun paquet de conjecture ne sera envoyé, cela DOIT être FAUX.
Après avoir reçu le paquet SSH_MSG_KEXINIT de l'autre côté, chaque partie va savoir si sa conjecture était juste. Si la conjecture de l'autre partie était fausse, et si ce champ était à VRAI, le prochain paquet DOIT être ignoré en silence, et les deux côtés DOIVENT alors agir comme déterminé par la méthode d'échange de clé négociée. Si la conjecture était juste, l'échange de clé DOIT continuer en utilisant le paquet de conjecture.
Après l'échange de message SSH_MSG_KEXINIT, l'algorithme d'échange de clé est lancé. Il peut impliquer plusieurs échanges de paquet, comme spécifié par la méthode d'échange de clé.
Une fois qu'une partie a envoyé un message SSH_MSG_KEXINIT pour l'échange ou le rééchange de clé, et jusqu'à ce qu'elle ait envoyé un message SSH_MSG_NEWKEYS (paragraphe 7.3) elle NE DOIT PAS envoyer de message autre que :
- o de messages génériques de couche transport (1 à 19) (mais SSH_MSG_SERVICE_REQUEST et SSH_MSG_SERVICE_ACCEPT NE DOIVENT PAS être envoyés) ;
- o des messages de négociation d'algorithmes (20 à 29) (mais d'autres messages SSH_MSG_KEXINIT NE DOIVENT PAS être envoyés) ;
- o des messages spécifiques de méthode d'échange de clé (30 à 49).

Les dispositions de la Section 11 s'appliquent aux messages non reconnus.

Noter cependant que durant un rééchange de clé, après l'envoi d'un message SSH_MSG_KEXINIT, chaque partie DOIT être prête à traiter un nombre arbitraire de messages qui peuvent être en cours avant la réception d'un message SSH_MSG_KEXINIT provenant de l'autre partie.

7.2 Résultat de l'échange de clés

L'échange de clé produit deux valeurs : un secret partagé K, et un hachage d'échange H. Les clés de chiffrement et d'authentification en sont déduites. Le hachage d'échange H provenant du premier échange de clé est en plus utilisé comme identifiant de session, qui est un identifiant univoque pour cette connexion. Il est utilisé par les méthodes d'authentification au titre des données qui sont signées comme preuve de possession d'une clé privée. Une fois calculé, l'identifiant de session n'est plus changé, même si les clés sont rééchangées ultérieurement.

Chaque méthode d'échange de clé spécifie une fonction de hachage qui est utilisée dans l'échange de clé. Le même algorithme de hachage DOIT être utilisé dans la déduction de clé. Ici, on l'appelle HASH.

Les clés de chiffrement DOIVENT être calculées comme HASH, d'une valeur connue et K, comme suit :
- o IV initiale de client à serveur : HASH(K || H || "A" || session_id) (ici K est codé comme mpint et "A" comme octet et session_id comme données brutes. "A" signifie un seul caractère A, ASCII 65).
- o IV initiale de serveur à client : HASH(K || H || "B" || session_id)
- o Clé de chiffrement de client à serveur : HASH(K || H || "C" || session_id)
- o Clé de chiffrement de serveur à client : HASH(K || H || "D" || session_id)
- o Clé d'intégrité de client à serveur : HASH(K || H || "E" || session_id)
- o Clé d'intégrité de serveur à client : HASH(K || H || "F" || session_id)

Les données de clé DOIVENT être prises à partir du début du résultat du hachage. Autant d'octets que nécessaire sont pris depuis le début de la valeur hachée. Si la longueur de clé nécessaire est plus longue que le résultat de HASH, la clé est étendue en calculant le HASH de la concaténation de K et de H et de la clé entière obtenue jusqu'alors, et en ajoutant les octets résultants (autant que HASH en génère) à la clé. Ce processus est répété jusqu'à ce qu'assez de matériel de clé soit disponible ; la clé est tirée du début de cette valeur. En d'autres termes :

K1 = HASH(K || H || X || session_id) (X est par exemple, "A")
K2 = HASH(K || H || K1)
K3 = HASH(K || H || K1 || K2)
...

clé = K1 || K2 || K3 || ...

Ce processus va perdre de l'entropie si la quantité d'entropie dans K est plus grande que la taille de l'état interne de HASH.

7.3 Mise en service des clés

Les échanges de clé se terminent par l'envoi des deux côtés d'un message SSH_MSG_NEWKEYS. Ce message est envoyé avec les vieilles clés et les vieux algorithmes. Tous les messages envoyés après ce message DOIVENT utiliser les nouvelles clé et les nouveaux algorithmes.

Lorsque ce message est reçu, les nouvelles clés et algorithmes DOIVENT être utilisés pour la réception.

Ce message a pour objet de s'assurer qu'une partie est capable de répondre par un message SSH_MSG_DISCONNECT que l'autre partie peut comprendre si quelque chose va de travers dans l'échange de clé.

octet SSH_MSG_NEWKEYS

8. Échange de clé Diffie-Hellman

L'échange de clé Diffie-Hellman (DH) procure un secret partagé qui ne peut être déterminé par l'une des parties seule. L'échange de clé est combiné avec une signature à l'aide de la clé d'hôte pour assurer l'authentification d'hôte. Cette méthode échange de clé fournit une authentification explicite du serveur comme défini à la Section 7.

On utilise les étapes suivantes pour un échange de clé. Dans cet échange, C est le client ; S est le serveur ; p est un grand nombre premier sûr ; g est un générateur pour un sous groupe de GF(p) ; q est l'ordre du sous groupe ; V_S est la chaîne d'identification de S ; V_C est la chaîne d'identification de C ; K_S est la clé d'hôte public de S ; I_C est le message SSH_MSG_KEXINIT de C et I_S est le message SSH_MSG_KEXINIT de S qui ont été échangés avant le début de cette partie.

1. C génère un nombre aléatoire x (1 < x < q) et calcule e = g^x mod p. C envoie e à S.

2. S génère un nombre aléatoire y (0 < y < q) et calcule f = g^y mod p. S reçoit e. Il calcule K = e^y mod p, H = hash(V_C || V_S || I_C || I_S || K_S || e || f || K) (ces éléments sont codés conformément à leur type ; voir ci-dessous), et la signature s sur H avec sa clé d'hôte privée. S envoie (K_S || f || s) à C. L'opération de signature peut impliquer une seconde opération de hachage.

3. C vérifie que K_S est réellement la clé d'hôte pour S (par exemple, en utilisant des certificats ou une base de données locale). Il est aussi admis que C accepte la clé sans vérification ; faire ainsi rend cependant le protocole non sûr contre les attaques actives (mais peut être désirable pour des raisons pratiques à court terme dans de nombreux environnements). C calcule alors K = f^x mod p, H = hash(V_C || V_S || I_C || I_S || K_S || e || f || K), et vérifie la signature s sur H.

Les valeurs de 'e' ou 'f' qui ne sont pas dans la gamme [1, p-1] NE DOIVENT PAS être envoyées ou acceptées par l'un ou l'autre côté. Si cette condition est violée, l'échange de clé échoue.

Ceci est mis en œuvre avec les messages suivants. L'algorithme de hachage pour le calcul du hachage de l'échange est défini par le nom de la méthode, et est appelé HASH. L'algorithme de clé publique pour la signature est négocié avec les messages SSH_MSG_KEXINIT.

D'abord, le client envoie ce qui suit :
octet SSH_MSG_KEXDH_INIT
mpint e

Le serveur répond alors avec ce qui suit :
octet SSH_MSG_KEXDH_REPLY
chaîne clé et certificats d'hôte public de serveur (K_S)
mpint f

Le hachage H est calculé comme le hachage HASH de l'enchaînement de ce qui suit :

chaîne	V_C,	chaîne d'identification du client (CR et LF exclus)
chaîne	V_S,	chaîne d'identification du serveur (CR et LF exclus)
chaîne	I_C,	charge utile du SSH_MSG_KEXINIT du client
chaîne	I_S,	charge utile du SSH_MSG_KEXINIT du serveur
chaîne	K_S,	clé d'hôte
mpint	e,	valeur d'échange envoyée par le client
mpint	f,	valeur d'échange envoyée par le serveur
mpint	K,	secret partagé

Cette valeur est appelée le hachage d'échange, et est utilisée pour authentifier l'échange de clé. Le hachage d'échange DEVRAIT rester secret.

L'algorithme de signature DOIT être appliqué sur H, et non sur les données d'origine. La plupart des algorithmes de signature incluent le hachage et le bourrage additionnel (par exemple, "ssh-dss" spécifie le hachage SHA-1). Dans ce cas, les données sont d'abord hachées avec HASH pour calculer H, et H est ensuite haché avec SHA-1 au titre de l'opération de signature.

8.1 diffie-hellman-group1-sha1

La méthode "diffie-hellman-group1-sha1" spécifie l'échange de clé Diffie-Hellman avec SHA-1 comme HASH, et le groupe 2 Oakley [RFC2409] (groupe MODP à 1024 bits). Cette méthode DOIT être acceptée pour l'interopérabilité car toutes les mises en œuvre connues actuellement la prennent en charge. Noter que cette méthode est désignée en utilisant le terme "group1", bien qu'elle spécifie l'utilisation du groupe 2 Oakley.

8.2 diffie-hellman-group14-sha1

La méthode "diffie-hellman-group14-sha1" spécifie un échange de clé Diffie-Hellman avec SHA-1 comme HASH et le groupe 14 Oakley [RFC3526] (groupe MODP à 2048 bits), et elle DOIT aussi être acceptée.

9. Rééchange de clé

Le rééchange de clé est commencé par l'envoi d'un paquet SSH_MSG_KEXINIT lorsqu'il ne fait pas déjà un échange de clé (comme décrit au paragraphe 7.1). Lorsqu'elle reçoit ce message, une partie DOIT répondre par son propre message SSH_MSG_KEXINIT, excepté lorsque le SSH_MSG_KEXINIT reçu est déjà une réponse. L'une ou l'autre partie PEUT initier le rééchange, mais les rôles NE DOIVENT PAS être inversés (c'est-à-dire que le serveur reste le serveur, et le client reste le client).

Le rééchange est effectué quel que soit le chiffrement qui était en effet lorsque l'échange a commencé. Les méthodes de chiffrement, de compression, et de MAC ne sont pas changées avant qu'un nouveau SSH_MSG_NEWKEYS soit envoyé après l'échange de clé (comme dans l'échange de clé initial). Le rééchange est traité de façon identique à celle de l'échange de clé initial, excepté pour l'identifiant de session qui va rester inchangé. Il est permis de changer certains des algorithmes durant le rééchange, ou tous. Les clés d'hôte peuvent aussi être changées. Toutes les clés et les vecteurs d'initialisation sont recalculés après l'échange. Les contextes de compression et de chiffrement sont remis à zéro.

Il est RECOMMANDÉ que les clés soient changées après chaque gigaoctet de données transmises ou après chaque heure de connexion, selon le premier qui survient. Cependant, comme le rééchange est une opération de clé publique, il exige une grosse quantité de puissance de traitement et ne devrait pas être effectué trop souvent.

Plus de données d'application peuvent être envoyées après l'envoi du paquet SSH_MSG_NEWKEYS ; l'échange de clé n'affecte pas les protocoles qui se tiennent en dessous de la couche de transport SSH.

10. Demande de service

Après l'échange de clé, le client demande un service. Le service est identifié par un nom. Le format des noms et des procédures de définition de nouveaux noms sont définies dans les [RFC4250] et [RFC4251].

Actuellement, les noms suivants ont été réservés :
ssh-userauth
ssh-connection

Une politique de dénomination locale similaire est appliquée aux noms de services, comme elle est appliquée aux noms d'algorithmes. Un service local devrait utiliser la syntaxe PRIVATE USE de "nom-de-service@domaine".

octet SSH_MSG_SERVICE_REQUEST
chaîne nom de service

Si le serveur rejette la demande de service, il DEVRAIT envoyer un message SSH_MSG_DISCONNECT approprié et DOIT se déconnecter.

Lorsque le service débute, il peut avoir accès à l'identifiant de session généré durant l'échange de clé.

Si le serveur accepte le service (et permet au client de l'utiliser) il DOIT répondre par ce qui suit :
octet SSH_MSG_SERVICE_ACCEPT
chaîne nom de service

Les numéros de message utilisés par les services devraient être dans la zone qui leur est réservée (voir les [RFC4250] et [RFC4251]). Le niveau de transport continuera de traiter ses propres messages.

Noter qu'après un échange de clé avec authentification implicite du serveur, le client DOIT attendre une réponse à son message de demande de service avant d'envoyer d'autres données.

11. Messages supplémentaires

L'une ou l'autre partie peut envoyer à tout moment l'un des messages suivants.

11.1 Message de déconnexion

octet SSH_MSG_DISCONNECT
uint32 code de cause
chaîne description en codage UTF-8 ISO-10646 [RFC3629]
chaîne étiquette de langage [RFC3066]

Ce message cause la terminaison immédiate de la connexion. Toutes les mises en œuvre DOIVENT être capables de traiter ce message ; elles DEVRAIENT être capables d'envoyer ce message.

L'envoyeur NE DOIT PAS envoyer ou recevoir de données après ce message, et le receveur NE DOIT PAS accepter de données après la réception de ce message. La chaîne 'description' du message de déconnexion donne une explication plus spécifique en forme lisible par l'homme. Le 'code de cause' du message de déconnexion donne la raison dans un format plus lisible par la machine (convenable pour la localisation), et peut avoir les valeurs affichées dans le tableau ci-dessous. Noter que la représentation décimale est affichée dans ce tableau dans un souci de lisibilité, mais les valeurs sont en réalité des valeurs de uint32.

Nom symbolique	**code de cause**
SSH_DISCONNECT_HOST_NOT_ALLOWED_TO_CONNECT	1
SSH_DISCONNECT_PROTOCOL_ERROR	2
SSH_DISCONNECT_KEY_EXCHANGE_FAILED	3
SSH_DISCONNECT_RESERVED	4
SSH_DISCONNECT_MAC_ERROR	5
SSH_DISCONNECT_COMPRESSION_ERROR	6
SSH_DISCONNECT_SERVICE_NOT_AVAILABLE	7
SSH_DISCONNECT_PROTOCOL_VERSION_NOT_SUPPORTED	8
SSH_DISCONNECT_HOST_KEY_NOT_VERIFIABLE	9
SSH_DISCONNECT_CONNECTION_LOST	10
SSH_DISCONNECT_BY_APPLICATION	11

SSH_DISCONNECT_TOO_MANY_CONNECTIONS	12
SSH_DISCONNECT_AUTH_CANCELLED_BY_USER	13
SSH_DISCONNECT_NO_MORE_AUTH_METHODS_AVAILABLE	14
SSH_DISCONNECT_ILLEGAL_USER_NAME	15

Si la chaîne 'description' est affichée, le filtrage des caractères de contrôle exposé dans la [RFC4251] devrait être utilisé pour éviter des attaques par l'envoi de caractères de contrôle au terminal.

Les demandes d'allocations de nouvelles valeurs de 'code de cause' du message de déconnexion (et du texte de 'description' associé) dans la gamme de 0x00000010 à 0xFDFFFFFF DOIVENT être effectuées par la méthode du CONSENSUS de l'IETF, telle que décrite dans la [RFC2434]. Les valeurs de 'code de cause' du message de déconnexion dans la gamme de 0xFE000000 à 0xFFFFFFFF sont réservées pour UTILISATION PRIVÉE. Comme indiqué, les instructions réelles à l'IANA figurent dans la [RFC4250].

11.2 Message Ignored Data

octet SSH_MSG_IGNORE
chaîne données

Toutes les mises en œuvre DOIVENT comprendre (et ignorer) ce message à tout moment (après réception de la chaîne d'identification). Aucune mise en œuvre n'est obligée de l'envoyer. Ce message peut être utilisé comme mesure de protection supplémentaire contre les techniques avancées d'analyse de trafic.

11.3. Message Debug

octet SSH_MSG_DEBUG
booléen toujours_affiché
chaîne message en codage UTF-8 ISO-10646 [RFC3629]
chaîne étiquette de langage [RFC3066]

Toutes les mises en œuvre DOIVENT comprendre ce message, mais elles ont le droit de l'ignorer. Ce message est utilisé pour transmettre des informations qui peuvent aider au débogage. Si 'toujours_affiché' est VRAI, le message DEVRAIT être affiché. Autrement, il NE DEVRAIT PAS être affiché sauf si des informations de débogage ont été explicitement demandées par l'usager.

Le 'message' n'a pas besoin de contenir une nouvelle ligne. Il est cependant permis qu'il consiste en plusieurs lignes séparées par des paires de CRLF (retour chariot – saut à la ligne).

Si la chaîne 'message' est affichée, le filtrage de caractères de contrôle du terminal exposé dans la [RFC4251] devrait être utilisé pour éviter des attaques par envoi de caractères de contrôle terminaux.

11.4 Messages réservés

Une mise en œuvre DOIT répondre à tout message non reconnu par un message SSH_MSG_UNIMPLEMENTED dans l'ordre dans lequel les messages ont été reçus. De tels messages DOIVENT être ignorés par ailleurs. Des versions ultérieures du protocole pourront définir d'autres significations pour ces types de message.

octet SSH_MSG_UNIMPLEMENTED
uint32 numéro de séquence de paquet du message rejeté

12. Résumé des numéros de message

Ci-après figure un résumé des messages et de leur numéro de message associé.

SSH_MSG_DISCONNECT	1
SSH_MSG_IGNORE	2
SSH_MSG_UNIMPLEMENTED	3
SSH_MSG_DEBUG	4

SSH_MSG_SERVICE_REQUEST	5
SSH_MSG_SERVICE_ACCEPT	6
SSH_MSG_KEXINIT	20
SSH_MSG_NEWKEYS	21

Noter que les numéros 30 à 49 sont utilisés pour les paquets kex. Différentes méthodes kex peuvent réutiliser les numéros de message dans cette gamme.

13. Considérations relatives à l'IANA

Le présent document fait partie d'un ensemble. Les considérations relatives à l'IANA pour le protocole SSH telles que définies dans les [RFC4251], [RFC4252], [RFC4254], et le présent document, sont détaillées dans la [RFC4250].

14. Considérations pour la sécurité

Le présent protocole fournit un canal chiffré sécurisé sur un réseau non sûr. Il effectue l'authentification d'hôte serveur, l'échange de clés, le chiffrement et la protection de l'intégrité. Il déduit aussi un identifiant de session univoque qui peut être utilisé par des protocoles de niveau supérieur.

Les considérations pour la sécurité complètes pour ce protocole sont fournies dans la [RFC4251].

15. Références

15.1 Références normatives

[FIPS-180-2] US National Institute of Standards and Technology, "Secure Hash Standard (SHS)", Federal Information Processing Standards Publication 180-2, août 2002.

[FIPS-186-2] US National Institute of Standards and Technology, "Digital Signature Standard (DSS)", Federal Information Processing Standards Publication 186-2, janvier 2000.

[FIPS-197] US National Institute of Standards and Technology, "Advanced Encryption Standard (AES)", Federal Information Processing Standards Publication 197, novembre 2001.

[FIPS-46-3] US National Institute of Standards and Technology, "Data Encryption Standard (DES)", Federal Information Processing Standards Publication 46-3, octobre 1999.

[RFC1321] R. Rivest, "Algorithme de résumé de message MD5", avril 1992. *(Information)*

[RFC1950] P. Deutsch et J-L Gailly, "Spécification du format ZLIB de données compressées, version 3.3", mai 1996.

[RFC1951] P. Deutsch, "Spécification du format DEFLATE de données compressées, version 1.3", mai 1996.

[RFC2104] H. Krawczyk, M. Bellare et R. Canetti, "HMAC : Hachage de clés pour l'authentification de message", février 1997.

[RFC2119] S. Bradner, "Mots clés à utiliser dans les RFC pour indiquer les niveaux d'exigence", BCP 14, mars 1997. *(MàJ par* RFC8174)

[RFC2144] C. Adams, "L'algorithme de chiffrement CAST-128", mai 1997. *(Information)*

[RFC2409] D. Harkins et D. Carrel, "L'échange de clés Internet (IKE)", novembre 1998. *(Obsolète, voir la* RFC4306)

[RFC2434] T. Narten et H. Alvestrand, "Lignes directrices pour la rédaction d'une section Considérations relatives à l'IANA dans les RFC", BCP 26, octobre 1998. *(Rendue obsolète par la* RFC5226)

[RFC2440] J. Callas, L. Donnerhacke, H. Finney et R. Thayer, "Format de message OpenPGP", novembre 1998. *(Obs. voir* 4880)

[RFC3066] H. Alvestrand, "Étiquettes pour l'identification des langues", BCP 47, janvier 2001. *(Obsolète, voir la* RFC4646.)

[RFC3447] J. Jonsson et B. Kaliski, "Normes de cryptographie à clés publiques (PKCS) n° 1 : Spécifications de la cryptographie RSA version 2.1", février 2003. *(Obsolète, remplacée par* RFC8017) *(Information)*

[RFC3526] T. Kivinen et M. Kojo, "Groupes supplémentaires d'exponentiation modulaire (MODP) Diffie-Hellman pour l'échange de clés Internet (IKE)", mai 2003.

[RFC3629] F. Yergeau, "UTF-8, un format de transformation de la norme ISO 10646", STD 63, novembre 2003.

[RFC4250] S. Lehtinen et C. Lonvick, éd., "Numéros alloués du protocole Secure Shell (SSH)", janvier 2006. *(P.S. ; MàJ par* RFC8268)

[RFC4251] T. Ylonen et C. Lonvick, "Architecture du protocole Secure Shell (SSH)", janvier 2006. *(P.S. ; MàJ par* RFC8308)

[RFC4252] T. Ylonen et C. Lonvick, éd., "Protocole d'authentification Secure Shell (SSH)", janvier 2006. *(P.S .; MàJ par* RFC8308, 8332)

[RFC4254] T. Ylonen et C. Lonvick, éd., "Protocole de connexion Secure Shell (SSH)", janvier 2006. *(P.S. ; MàJ par* RFC8308)

[SCHNEIER] Schneier, B., "Applied Cryptography Second Edition: protocols algorithms and source in code in C", John Wiley and Sons, New York, NY, 1996.

[TWOFISH] Schneier, B., "The Twofish Encryptions Algorithm: A 128-Bit Block Cipher, 1st Edition", mars 1999.

15.2 Références pour information

[RFC0894] C. Hornig, "Norme pour la transmission des datagrammes IP sur les réseaux Ethernet", STD 41, avril 1984.

[RFC1661] W. Simpson, éditeur, "Protocole point à point (PPP)", STD 51, juillet 1994. *(MàJ par la* RFC2153)

[RFC2412] H. Orman, "Protocole OAKLEY de détermination de clés", novembre 1998. *(Information)*

[ssh-1.2.30] lonen, T., "ssh-1.2.30/RFC", Fichier compressé en tarball ftp://ftp.funet.fi/pub/unix/security/login/ssh/ssh-1.2.30.tar.gz , novembre 1995.

Adresse des auteurs

Tatu Ylonen
SSH Communications Security Corp
Valimotie 17
00380 Helsinki
Finland
mél : ylo@ssh.com

Chris Lonvick (editor)
Cisco Systems, Inc.
12515 Research Blvd.
Austin 78759
USA
mél : clonvick@cisco.com

Notice de marque commerciale
"ssh" est une marque commerciale déposée aux États Unis d'Amérique et/ou autres pays.

Déclaration complète de droits de reproduction

Propriété intellectuelle
L'IETF ne prend pas position sur la validité et la portée de tout droit de propriété intellectuelle ou autres droits qui pourrait être revendiqués au titre de la mise en œuvre ou l'utilisation de la technologie décrite dans le présent document ou sur la mesure dans laquelle toute licence sur de tels droits pourrait être ou n'être pas disponible ; pas plus qu'elle ne prétend avoir accompli aucun effort pour identifier de tels droits. Les informations sur les procédures de l'ISOC au sujet des droits dans les documents de l'ISOC figurent dans les BCP 78 et BCP 79.

Des copies des dépôts d'IPR faites au secrétariat de l'IETF et toutes assurances de disponibilité de licences, ou le résultat de tentatives faites pour obtenir une licence ou permission générale d'utilisation de tels droits de propriété par ceux qui mettent en œuvre ou utilisent la présente spécification peuvent être obtenues sur répertoire en ligne des IPR de l'IETF à http://www.ietf.org/ipr .

L'IETF invite toute partie intéressée à porter son attention sur tous copyrights, licences ou applications de licence, ou autres droits de propriété qui pourraient couvrir les technologies qui peuvent être nécessaires pour mettre en œuvre la présente norme. Prière d'adresser les informations à l'IETF à ietf- ipr@ietf.org .

Remerciement
Le financement de la fonction d'édition des RFC est actuellement fourni par la Internet Society.