

Groupe de travail Réseau
Request for Comments : 4419
Catégorie : En cours de normalisation
Traduction Claude Brière de L'Isle

M. Friedl
N. Provos
W. Simpson
mars 2006

Échange de groupe Diffie-Hellman pour le protocole de couche Transport Secure Shell (SSH)

Statut de ce mémoire

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

Notice de copyright

Copyright (C) The Internet Society (2006).

Résumé

Le présent mémoire décrit une nouvelle méthode d'échange de clés pour le protocole Secure Shell (SSH). Il permet au serveur SSH de proposer au client de nouveaux groupes sur lesquels effectuer l'échange de clés Diffie-Hellman. Les groupes proposés ne sont pas nécessairement fixés et peuvent changer dans le temps.

1. Généralités et motifs

SSH [RFC4251] est un protocole très courant pour une connexion à distance sur l'Internet. Actuellement, SSH effectue l'échange de clés initial en utilisant la méthode "diffie-hellman-group1-sha1" [RFC4253]. Cette méthode prescrit un groupe fixe sur lequel toutes les opérations sont effectuées.

L'échange de clés Diffie-Hellman fournit un secret partagé qui ne peut être déterminé par une des parties seule. De plus, le secret partagé n'est connu que des parties participantes. Dans SSH, l'échange de clés est signé avec la clé de l'hôte pour fournir l'authentification de l'hôte.

La sécurité de l'échange de clés Diffie-Hellman se fonde sur la difficulté à résoudre le problème du logarithme discret (DLP, *Discrete Logarithm Problem*). Comme on s'attend à ce que le protocole SSH soit utilisé pendant de nombreuses années, on craint que des calculs préalables extensifs et des algorithmes plus efficaces pour calculer le logarithme discret sur un groupe fixé puisse créer une menace pour la sécurité du protocole SSH.

La capacité à proposer de nouveaux groupes réduira l'incitation à utiliser des calculs préalables pour un calcul plus efficace du logarithme discret. Le serveur peut constamment calculer de nouveaux groupes en arrière plan.

2. Notation des exigences

Dans le présent document, les mots clés "DOIT", "NE DOIT PAS", "EXIGE", "DEVRA", "NE DEVRA PAS", "DEVRAIT", "NE DEVRAI PAS", "RECOMMANDE", "PEUT", et "FACULTATIF" sont à interpréter comme décrit dans la [RFC2119].

3. Groupe Diffie-Hellman et échange de clés

Le serveur conserve une liste de nombres premiers sûrs et des générateurs correspondants parmi lesquels il peut faire un choix. Un nombre premier p est sûr si $p = 2q + 1$ et si q est premier. De nouveaux nombres premiers peuvent être générés en arrière plan.

Le générateur g devrait être choisi de telle sorte que l'ordre du sous-groupe généré ne soit pas un facteur de petits nombres premiers ; c'est-à-dire que, avec $p = 2q + 1$, l'ordre doit être q ou $p - 1$. Si l'ordre est $p - 1$, les exposants génèrent alors toutes les valeurs publiques possibles, uniformément réparties sur toute la gamme des modulo p , sans bouclage sur un plus petit sous-ensemble. Un tel générateur est appelé une "racine primitive" (ce qu'il est trivial de trouver lorsque p est "sûr").

Le client demande un modulo au serveur en indiquant la taille préférée.

Dans la description suivante :

C est le client,

S est le serveur,

le modulo p est un grand nombre premier sûr, et

g est un générateur pour un sous-groupe de $GF(p)$,

min est la taille minimale de p en bits qui est acceptable pour le client,

n est la taille du modulo p en bits que le client voudrait recevoir du serveur,

max est la taille maximale de p en bits que le client peut accepter,

V_S est la chaîne de version de S,

V_C est la chaîne de version de C,

K_S est la clé d'hôte public de S,

I_C est le message KEXINIT de C, et

I_S est le message KEXINIT de S qui a été échangé avant que ne commence cette partie.

1. C envoie " $min \parallel n \parallel max$ " à S, indiquant la taille minimale de groupe acceptable, la taille préférée du groupe, et la taille maximale de groupe en bits que le client acceptera.
2. S trouve un groupe qui correspond le mieux à la demande du client, et envoie " $p \parallel g$ " à C.
3. C génère un nombre aléatoire x , où $1 < x < (p-1)/2$. Il calcule $e = g^x \text{ mod } p$, et envoie " e " à S.
4. S génère un nombre aléatoire y , où $0 < y < (p-1)/2$, et calcule $f = g^y \text{ mod } p$. S reçoit " e ". Il calcule $K = e^y \text{ mod } p$, $H = \text{hash}(V_C \parallel V_S \parallel I_C \parallel I_S \parallel K_S \parallel min \parallel n \parallel max \parallel p \parallel g \parallel e \parallel f \parallel K)$ (ces éléments sont codés conformément à leur type ; voir ci-dessous), et signe s sur H avec sa clé d'hôte privée. S envoie " $K_S \parallel f \parallel s$ " à C. L'opération de signature peut impliquer une seconde opération de hachage.
5. 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 pour obtenir la clé publique). C peut aussi accepter la clé sans vérification ; cependant, il rend, ce faisant, le protocole non sûr contre les attaques actives (mais cela peut être souhaitable pour des raisons pratiques à court terme dans de nombreux environnements). C calcule alors $K = f^x \text{ mod } p$, $H = \text{hash}(V_C \parallel V_S \parallel I_C \parallel I_S \parallel K_S \parallel min \parallel n \parallel max \parallel p \parallel g \parallel e \parallel f \parallel K)$, et vérifie la signature s sur H .

Serveurs et clients DEVRAIENT accepter des groupes avec une longueur de modulo de k bits, où $1024 \leq k \leq 8192$. Les valeurs recommandées pour min et max sont respectivement 1024 et 8192.

L'un ou l'autre côté NE DOIT PAS envoyer ou accepter de valeurs e ou f qui ne soient pas dans la gamme $[1, p-1]$. Si cette condition est violée, l'échange de clé échoue. Pour empêcher les attaques de confinement, ils DOIVENT accepter le secret partagé K seulement si $1 < K < p - 1$.

Le serveur devrait retourner le plus petit groupe qu'il connaît qui soit supérieur à la taille demandée par le client. Si le serveur ne connaît pas un groupe qui soit supérieur à la taille demandée par le client, il DEVRAIT alors retourner le plus grand groupe qu'il connaît. Dans tous les cas, la taille du groupe retourné DEVRAIT être d'au moins 1024 bits.

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 signer est négocié avec le message KEXINIT.

D'abord, le client envoie :

```
byte      SSH_MSG_KEY_DH_GEX_REQUEST
uint32    min,   taille minimale en bits d'un groupe acceptable
uint32    n,     taille préférée en bits du groupe que le serveur va envoyer
uint32    max,   taille maximale en bits d'un groupe acceptable
```

Le serveur répond par :

```
byte      SSH_MSG_KEX_DH_GEX_GROUP
mpint     p,     nombre premier sûr
mpint     g,     générateur pour le sous-groupe en GF(p)
```

Le client répond par :

```
byte    SSH_MSG_KEX_DH_GEX_INIT
mpint   e
```

Le serveur répond par :

```
byte    SSH_MSG_KEX_DH_GEX_REPLY
string  clé d'hôte publique et certificats (K_S) du serveur
mpint   fstring signature de H
```

Le hachage H est calculé comme hachage HASH de la concaténation de ce qui suit :

```
string  V_C,    la chaîne version du client (CR et NL exclus)
string  V_S,    la chaîne version du serveur (CR et NL exclus)
string  I_C,    la charge utile du
SSH_MSG_KEXINIT du client
string  I_S,    la charge utile du SSH_MSG_KEXINIT du serveur
string  K_S,    la clé d'hôte
uint32  min,    taille minimale en bits d'un groupe acceptable
uint32  n,      taille préférée en bits du groupe que le server va envoyer
uint32  max,    taille maximale en bits d'un groupe acceptable
mpint   p,      nombre premier sûr
mpint   g,      générateur pour le sous-groupe
mpint   e,      valeur d'échange envoyée par le client
mpint   f,      valeur d'échange envoyée par le serveur
mpint   K,      le secret partagé
```

Cette valeur est appelée le hachage d'échange, et elle est utilisée pour authentifier l'échange de clé conformément à la [RFC4253].

4. Méthodes d'échange de clés

Le présent document définit deux nouvelles méthodes d'échange de clés :

```
"diffie-hellman-group-exchange-sha1"
"diffie-hellman-group-exchange-sha256".
```

4.1. diffie-hellman-group-exchange-sha1

La méthode "diffie-hellman-group-exchange-sha1" spécifie le groupe et l'échange de clés Diffie-Hellman avec SHA-1 [FIPS-180-2] comme HASH.

4.2. diffie-hellman-group-exchange-sha256

La méthode "diffie-hellman-group-exchange-sha256" spécifie le groupe et l'échange de clés Diffie-Hellman avec SHA-256 [FIPS-180-2] comme HASH.

Noter que le hachage utilisé dans l'échange de clés (dans ce cas, SHA-256) doit aussi être utilisé dans la fonction pseudo aléatoire de déduction de clé (PRF), conformément à l'exigence du paragraphe "Résultat de l'échange de clés" de la [RFC4253].

5. Résumé des numéros de message

Les numéros de message suivants ont été définis dans le présent document. Il sont dans un espace de noms privé du présent document et ne sont pas alloués par l'IANA.

```
#define SSH_MSG_KEX_DH_GEX_REQUEST_OLD 30
#define SSH_MSG_KEX_DH_GEX_REQUEST    34
#define SSH_MSG_KEX_DH_GEX_GROUP      31
```

```
#define SSH_MSG_KEX_DH_GEX_INIT      32
#define SSH_MSG_KEX_DH_GEX_REPLY    33
```

SSH_MSG_KEX_DH_GEX_REQUEST_OLD est utilisé pour la rétro compatibilité. Au lieu d'envoyer "min || n || max", le client envoie seulement "n". De plus, le hachage est calculé en utilisant seulement "n" au lieu de "min || n || max".

Les numéros 30-49 sont spécifiques de l'échange de clé et pourront être redéfinis par d'autres méthodes kex.

6. Notes de mise en œuvre

6.1. Choix du générateur

Une technique utile est de choisir le générateur, puis de limiter le filtre de choix du modulo aux nombres premiers avec ce générateur :

- 2 lorsque $p \pmod{24} = 11$.
- 5 lorsque $p \pmod{10} = 3$ ou 7 .

Il est recommandé d'utiliser 2 comme générateur, parce qu'il améliore l'efficacité dans la multiplication. Il est utilisable même lorsqu'il n'est pas une racine primitive, car il couvre encore la moitié de l'espace des résidus possibles.

6.2. Exposants privés

Pour accroître la vitesse de l'échange de clés, le client et le serveur peuvent tous deux réduire la taille de leurs exposants privés. Ils devraient être au moins deux fois plus longs que le matériel de clé qui est généré à partir du secret partagé. Pour des précisions complémentaires, voir l'article de van Oorschot et Wiener [VAN-OORSCHOT].

7. Considérations pour la sécurité

Ce protocole vise la simplicité et n'utilise que des primitives bien comprises. Cela encourage l'acceptation par la communauté et permet de faciliter la mise en œuvre, qui mène heureusement à un système plus sûr.

L'utilisation de plusieurs modulo inhibe les efforts de calcul préalable des valeurs d'échange de modulo par un agresseur déterminé, et décourage de consacrer des ressources à l'analyse d'un modulo particulier.

Il est important de n'employer que des nombres premiers sûrs comme modulo, car ils nous permettent de trouver un générateur g de telle sorte que l'ordre des sous-groupes générés ne se mette pas en facteurs dans de petits nombres premiers, c'est-à-dire que, avec $p = 2q + 1$, l'ordre doit être soit q soit $p - 1$. Si l'ordre est $p - 1$, les exposants génèrent alors toutes les valeurs publiques possibles, uniformément réparties sur toute la gamme du modulo p , sans bouclage sur un plus petit sous-ensemble. Van Oorschot et Wiener notent qu'en utilisant des exposants privés courts avec un nombre premier aléatoire, modulo p rend facile le calcul du logarithme discret [VAN-OORSCHOT]. Cependant, ils déclarent aussi que ce problème ne s'applique pas aux nombres premiers sûrs.

Le bit de moindre poids de l'exposant privé peut être retrouvé lorsque le modulo est un nombre premier sûr [MENEZES]. Toutefois, ce n'est pas un problème si la taille de l'exposant privé est assez grande. En relation avec ceci, Waldvogel et Massey notent : Lorsque les exposants privés sont choisis de façon indépendante et uniformément aléatoire à partir de $\{0, \dots, p-2\}$, l'entropie de la clé est inférieure de 2 bits au maximum $\lg(p-1)$ [WALDVOGEL].

Les considérations pour la sécurité de la [RFC4251] s'appliquent aussi à cette méthode d'échange de clés.

8. Remerciements

Le présent document découle en partie du "Protocole SSH de couche Transport" [RFC4253] de T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne et S. Lehtinen.

Markku-Juhani Saarinen a attiré l'attention sur le fait que le bit de moindre poids de l'exposant privé peut être retrouvé

efficacement lorsqu'on utilise des nombres premiers sûrs et un sous-groupe avec un ordre divisible par deux.

Bodo Moeller a suggéré que le serveur n'envoie qu'un groupe, réduisant la complexité de la mise en œuvre et la quantité de données qui doivent être échangées entre client et serveur.

Appendice A : Génération de nombres premiers sûrs

Le "Manuel de cryptographie appliquée" [MENEZES] fait la liste des algorithmes suivants pour générer un nombre premier p sûr de k bits. Elle a été modifiée de sorte que 2 soit un générateur pour le groupe multiplicatif mod p .

1. Faire ce qui suit :
 - a. Choisir un nombre premier aléatoire p de $(k-1)$ bits, tel que $q \bmod 12 = 5$.
 - b. Calculer $p := 2q + 1$, et regarder si p est premier (en utilisant, par exemple, un essai de division et le test Rabin-Miller).
2. Répéter jusqu'à ce que p soit premier.

Si une mise en œuvre utilise la bibliothèque OpenSSL, un groupe consistant en un nombre premier sûr de 1024 bits et un 2 comme générateur peut être créé comme suit :

```
DH *d = NULL;
d = DH_generate_parameters(1024, DH_GENERATOR_2, NULL, NULL);
BN_print_fp(stdout, d->p);
```

L'ordre du sous-groupe généré par 2 est $q = p - 1$.

Références

Références normatives

- [FIPS-180-2] National Institute of Standards and Technology (NIST), "Secure Hash Standard (SHS)", FIPS PUB 180-2, août 2002.
- [RFC4251] T. Ylonen et C. Lonvick, "Architecture du protocole Secure Shell (SSH)", RFC 4251, janvier 2006.
- [RFC4253] C. Lonvick, "Protocole de [couche Transport Secure Shell](#) (SSH)", RFC 4253, janvier 2006.
- [RFC2119] S. Bradner, "[Mots clés à utiliser dans les RFC](#) pour indiquer les niveaux d'exigence", BCP 14, RFC 2119, mars 1997.

Références informatives

- [MENEZES] Menezes, A., van Oorschot, P., and S. Vanstone, "Handbook of Applied Cryptography", CRC Press, p. 537, 1996.
- [VAN-OORSCHOT] van Oorschot, P. and M. Wiener, "On Diffie-Hellman key agreement with short exponents", Advances in Cryptology -EUROCRYPT'96, LNCS 1070, Springer-Verlag, pp. 332-343, 1996.
- [WALDVOGEL] Waldvogel, C. and J. Massey, "The probability distribution of the Diffie-Hellman key", Proceedings of AUSCRYPT 92, LNCS 718, Springer-Verlag, pp. 492-504, 1993.

Adresse des auteurs

Markus Friedl, mél : markus@openbsd.org
Niels Provos, mél : provos@citi.umich.edu

William A. Simpson, mél : wsimpson@greendragon.com

Déclaration complète de droits de reproduction

Copyright (C) The Internet Society (2006).

Le présent document est soumis aux droits, licences et restrictions contenus dans le BCP 78, et à www.rfc-editor.org, et sauf pour ce qui est mentionné ci-après, les auteurs conservent tous leurs droits.

Le présent document et les informations y contenues sont fournies sur une base "EN L'ÉTAT" et LE CONTRIBUTEUR, L'ORGANISATION QU'IL OU ELLE REPRÉSENTE OU QUI LE/LA FINANCE (S'IL EN EST), LA INTERNET SOCIETY ET LA INTERNET ENGINEERING TASK FORCE DÉCLINENT TOUTES GARANTIES, EXPRIMÉES OU IMPLICITES, Y COMPRIS MAIS NON LIMITÉES À TOUTE GARANTIE QUE L'UTILISATION DES INFORMATIONS CI-ENCLOSES NE VIOLENT AUCUN DROIT OU AUCUNE GARANTIE IMPLICITE DE COMMERCIALISATION OU D'APTITUDE À UN OBJET PARTICULIER.

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 l'Internet Society.