Groupe de travail Réseau
Request for Comments : 2104
Catégorie : Information
Traduction Claude Brière de L'Isle

H. Krawczyk, IBM
M. Bellare, UCSD
R. Canetti, IBM

février 1997

HMAC : Hachage de clé pour l'authentification de message

Résumé
Le présent document décrit HMAC, un mécanisme d'authentification de message qui utilise des fonctions de hachage cryptographiques. HMAC peut être utilisé avec toute fonction de hachage cryptographique itérative, par exemple, MD5, SHA-1, en combinaison avec une clé secrète partagée. La force cryptographique de HMAC dépend des propriétés de la fonction de hachage sous-jacente.

1. Introduction

Fournir le moyen de vérifier l'intégrité des informations transmises sur, ou mémorisées dans, un support non fiable est de la première nécessité dans le monde de libre communication d'aujourd'hui. Les mécanismes qui fournissent une telle vérification d'intégrité fondée sur une clé secrète sont habituellement désignés par le terme de "code d'authentification de message" (MAC, *message authentication code*). Normalement, les codes d'authentification de message sont utilisés entre deux parties qui partagent une clé secrète afin de valider les informations transmises entre ces parties. Dans le présent document, on présente un tel mécanisme MAC fondé sur des fonctions de hachage cryptographique. Ce mécanisme, appelé HMAC, est fondé sur les travaux des auteurs [BCK1] dans lesquels la construction est présentée et analysée du point de vue cryptographique. Prière de se référer à ces travaux pour les détails des raisons et de l'analyse de la sécurité de HMAC, et sa comparaison avec les autres méthodes de hachage de clés.

HMAC peut être utilisé de façon combinée avec toute fonction de hachage cryptographique itérative. MD5 et SHA-1 sont des exemples de telles fonctions de hachage. HMAC utilise aussi une clé secrète pour le calcul et la vérification de l'authentification des valeurs de message. Les principaux objectifs de cette construction sont :

* D'utiliser sans modification les fonctions de hachage disponibles. En particulier, les fonctions de hachage qui se comportent bien dans le logiciel, et pour lesquelles le code est disponible, gratuit et facile d'accès.
* De préserver les performances d'origine de la fonction de hachage sans qu'il résulte de dégradation significative.
* D'utiliser et manipuler les clés de façon simple.
* D'avoir une analyse cryptographique bien comprise de la force du mécanisme d'authentification fondée sur des hypothèses raisonnables sur la fonction de hachage sous-jacente.
* De permettre un remplacement facile de la fonction de hachage sous-jacente au cas où existeraient ou seraient exigées des fonctions de hachage plus sûres ou plus rapides.

Le présent document spécifie HMAC en utilisant une fonction de hachage cryptographique générique (notée H). Des instanciations spécifiques de HMAC doivent définir une fonction de hachage particulière. Les candidats actuels pour de telles fonctions de hachage incluent SHA-1 [SHA], MD5 [MD5], RIPEMD-128/160 [RIPEMD]. Ces diverses réalisations de MAC seront notées HMAC-SHA1, HMAC-MD5, HMAC-RIPEMD, etc.

Note : Au moment de la rédaction du présent document, MD5 et SHA-1 sont les fonctions de hachage cryptographique les plus largement utilisées. MD5 s'est récemment révélé vulnérable aux attaques de recherche par collision [Dobb]. Cette attaque et les autres faiblesses connues actuellement de MD5 ne compromettent pas l'utilisation de MD5 dans HMAC comme spécifié dans le présent document (voir [Dobb]) ; cependant, SHA-1 paraît une fonction cryptographiquement plus forte. À ce jour, l'utilisation de MD5 dans HMAC peut être prise en considération pour les applications où les performances supérieures de MD5 sont très importantes. Dans tous les cas, les mises en œuvre et les utilisateurs doivent savoir qu'il existe de possibilités de développements cryptanalytiques concernant toutes ces fonctions de hachage cryptographiques, et qu'il sera éventuellement nécessaire de remplacer la fonction de hachage sous-jacente. (Voir la section 6 pour plus d'informations sur la sécurité de HMAC.)

2. Définition de HMAC

La définition de HMAC exige une fonction de hachage cryptographique, que nous notons H, et une clé secrète K. On suppose que H est une fonction de hachage cryptographique où les données sont hachées en itérant une fonction de compression de base sur les blocs de données. On note B la longueur en octets de ces blocs (B = 64 pour tous les exemples de fonctions de hachage susmentionnés) et L la longueur en octets des résultats du hachage (L = 16 pour MD5, L = 20 pour SHA-1). La clé d'authentification K peut être de n'importe quelle longueur jusqu'à B, longueur de bloc de la fonction de hachage. Les applications qui utilisent des clés plus longues que B octets vont d'abord hacher la clé en utilisant H et ensuite utiliser la chaîne d'octets résultante L comme clé réelle pour HMAC. Dans tous les cas la longueur minimale recommandée pour K est L octets (comme la longueur du résultat du hachage). Voir à la section 3 des informations complémentaires sur les clés.

On définit deux chaînes ipad et opad fixes et différentes comme suit (le 'i' et le 'o' sont des mnémoniques pour interne et externe) :
ipad = l'octet 0x36 répété B fois
opad = l'octet 0x5C répété B fois.

Pour calculer HMAC sur les données "text" on effectue : H(K XOR opad, H(K XOR ipad, text))

À savoir
(1) Ajouter des zéros à la fin de K pour créer une chaîne d'octets B (par exemple, si K est de 20 octets et B = 64, alors 44 octets zéro 0x00 seront ajoutés à K)
(2) Combiner avec l'opérateur OUX (OU exclusif au bit près) la chaîne d'octets B calculée à l'étape (1) avec ipad
(3) Ajouter le flux de données 'text' à la chaîne d'octets B résultant de l'étape (2)
(4) Appliquer H au flux généré à l'étape (3)
(5) Combiner avec l'opérateur OUX (OU exclusif au bit près) à la chaîne d'octets B calculée à l'étape (1) avec opad
(6) Ajouter le résultat H de l'étape (4) à la chaîne d'octets B résultant de l'étape (5)
(7) Appliquer H au flux généré à l'étape (6) et sortir le résultat

Pour illustrer le propos, un échantillon du code fondé sur MD5 est fourni en appendice.

3. Clés

La clé pour HMAC peut avoir une longueur quelconque (les clés plus longues que B octets sont d'abord hachées en utilisant H). Cependant, moins de L octets est fortement déconseillé car cela minimiserait la force de la sécurité de la fonction. Les clés plus longues que L octets sont acceptables mais la longueur additionnelle n'augmentera pas de façon significative la force de la fonction. (Une clé plus longue serait conseillée si l'aléa de la clé est considéré comme faible.)

Les clés doivent être choisies de façon aléatoire (ou en utilisant un générateur pseudo aléatoire cryptographiquement fort alimenté avec un germe aléatoire) et rafraîchies périodiquement. (Il ne découle pas des attaques actuelles une indication d'une fréquence recommandée spécifique de changement de clés car ces attaques sont infaisables en pratique. Cependant, un rafraîchissement périodique des clés est une pratique de sécurité fondamentale qui aide à surmonter la faiblesse potentielle de la fonction et des clés, et limite les dommages résultant de la compromission d'une clé).

4. Note de mise en œuvre

HMAC est défini de telle façon que la fonction de hachage H sous-jacente puisse être utilisée sans modification de son code. En particulier, il utilise la fonction H avec la valeur initiale IV prédéfinie (une valeur fixe spécifiée par chaque fonction de hachage itérative pour initialiser sa fonction de compression). Cependant, si on le désire, une amélioration des performances peut être réalisée au prix d'une modification (éventuelle) du code de H pour prendre en charge des IV variables.

L'idée est que les résultats intermédiaires de la fonction de compression sur les blocs de B octets (K XOR ipad) et (K XOR opad) ne peuvent être pré calculés qu'une seule fois au moment de la génération de la clé K, ou avant sa première utilisation. Ces résultats intermédiaires sont mémorisés puis utilisés pour initialiser le IV de H chaque fois qu'un message a besoin d'être authentifié. Cette méthode fait l'économie, pour chaque message authentifié, de l'application de la fonction de compression de H sur deux blocs de B octets (c'est-à-dire, sur (K XOR ipad) et (K XOR opad)). De telles économies peuvent être significatives lorsqu'on authentifie des flux de données courts. On souligne que les valeurs intermédiaires mémorisées doivent être traitées et protégées de la même façon que des clés secrètes.

Choisir de mettre en œuvre HMAC de la façon décrite ci-dessus est une décision de la mise en œuvre locale et n'a pas d'effet sur l'interopérabilité.

5. Résultat tronqué

Une pratique bien connue avec les codes d'authentification de message est de tronquer le résultat du MAC et de sortir seulement une partie des bits (par exemple, [MM], [ANSI]). Preneel et van Oorschot [PV] montrent certains avantages analytiques de la troncature du résultat des fonctions MAC fondées sur le hachage. Les résultats dans ce domaine ne sont pas aussi absolus que pour l'avantage global de la troncature pour la sécurité. Il y a des avantages (moins d'informations disponibles pour un attaquant sur le résultat du hachage) et des inconvénients (moins de bits à deviner pour l'attaquant). Les applications de HMAC peuvent choisir de tronquer le résultat de HMAC en mettant en sortie les t bits de gauche du calcul HMAC pour un paramètre t (à savoir que le calcul est fait de la façon normale comme défini à la section 2 ci-dessus, mais le résultat final est tronqué à t bits). On recommande que la longueur t du résultat ne soit pas inférieure à la moitié de la longueur du résultat du hachage (pour tenir la limite de l'attaque de l'anniversaire) et pas moins de 80 bits (une limite inférieure convenable sur le nombre de bits qui doivent être prédits par un attaquant). On propose de noter une réalisation de HMAC qui utilise une fonction de hachage H avec t bits de sortie HMAC-H-t. Par exemple, HMAC-SHA1-80 note HMAC calculé en utilisant la fonction SHA-1 et avec le résultat tronqué à 80 bits. (Si le paramètre t n'est pas spécifié, par exemple HMAC-MD5, on suppose alors que tous les bits du hachage sont sortis.)

6. Sécurité

La sécurité du mécanisme d'authentification de message présenté ici dépend des propriétés cryptographiques de la fonction de hachage H : la résistance aux recherches de collision (limitée au cas où la valeur initiale est secrète et aléatoire, et où le résultat de la fonction n'est pas explicitement disponible pour l'attaquant) et la propriété de l'authentification de message de la fonction de compression de H lorsque appliquée à un seul bloc (dans HMAC ces blocs sont partiellement inconnus d'un attaquant car ils contiennent le résultat du calcul interne de H et, en particulier, ne peuvent pas être librement choisis par l'attaquant).

Ces propriétés, et en fait de plus fortes, sont couramment supposées pour les fonctions de hachage du type utilisé avec HMAC. En particulier, une fonction de hachage pour laquelle les propriétés ci-dessus ne tiennent pas deviendrait impropre pour la plupart (sinon toutes) les applications cryptographiques, y compris les schémas d'authentification de message alternatifs fondés sur de telles fonctions. (Pour une analyse complète et les motifs de la fonction HMAC, le lecteur se reportera à [BCK1].)

Étant donnée la confiance limitée portée jusqu'ici à la force cryptographique des fonctions de hachage candidates, il est important d'observer les deux propriétés suivantes de la construction HMAC et de son utilisation sécurisée pour l'authentification de message :

1. La construction est indépendante des détails d'utilisation de la fonction de hachage H particulière et donc cette dernière peut être remplacée par toute autre fonction de hachage cryptographique sûre (itérative).

2. L'authentification de message, par opposition au chiffrement, a un effet "transitoire". La publication d'un schéma d'authentification de message conduirait au remplacement de ce schéma, mais n'aurait pas d'effet contraire sur les informations déjà authentifiées. Cela présente un vif contraste avec le chiffrement, où les informations chiffrées aujourd'hui peuvent souffrir de leur divulgation à l'avenir si et lorsque l'algorithme de chiffrement est cassé.

La plus forte attaque connue contre HMAC est fondée sur la fréquence des collisions pour la fonction de hachage H ("attaque anniversaire") [PV,BCK2], et elle est totalement impraticable pour des fonctions de hachage raisonnablement minimales.

Par exemple, si nous considérons une fonction de hachage comme MD5 où la longueur L du résultat égale 16 octets (128 bits) l'attaquant doit acquérir les étiquettes correctes d'authentification de message calculées (avec la_même_clé secrète K !) sur environ 2**64 libellés connus. Cela exigerait le traitement d'au moins 2**64 blocs sous H, une tâche impossible dans tout scénario réaliste (pour une longueur de bloc de 64 octets cela prendrait 250 000 ans en continu sur une liaison à 1 Gbit/s, et sans changer la clé secrète K durant tout ce temps). Cette attaque ne pourrait devenir réelle que si de sérieuses fautes dans le comportement de collision de la fonction H était découvertes (par exemple des collisions trouvées après 2**30 messages). Une telle découverte déterminerait le remplacement immédiat de la fonction H (les effets d'une telle défaillance seraient bien plus sévères pour les utilisations traditionnelles de H dans le contexte de signatures

numériques, de certificats de clés publiques, etc.).

Note : Cette attaque devrait être très différente des attaques de collision régulières sur les fonctions de hachage cryptographiques où aucune clé secrète n'est impliquée et où 2**64 opérations hors ligne parallélisables (!) suffisent pour trouver des collisions. Cette dernière attaque se rapproche de la faisabilité [VW] alors que l'attaque anniversaire sur HMAC est totalement impraticable. (Dans les exemples ci-dessus, si quelqu'un utilise une fonction de hachage avec, disons, 160 bit de sortie, alors 2**64 devrait être remplacé par 2**80.)

Une mise en œuvre correcte de la construction ci-dessus, le choix de clés aléatoires (ou cryptographiquement pseudo aléatoires) un mécanisme d'échange de clés sécurisé, de fréquents rafraîchissements de clé et une bonne protection du secret des clés sont tous les ingrédients essentiels pour la sécurité du mécanisme de vérification d'intégrité fourni par HMAC.

Appendice – Échantillon de code

À titre d'illustration, on fournit l'échantillon de code suivant pour la mise en œuvre de HMAC-MD5 ainsi que certains vecteurs d'essai correspondants (le code est fondé sur le code MD5 comme décrit dans [MD5]).

```
/*
** Fonction : hmac_md5
*/

vide
hmac_md5(text, text_len, key, key_len, digest)
unsigned char*  text;           /* pointeur sur le flux de données */
int             text_len;       /* longueur du flus de données */
unsigned char*  key             /* pointeur sur clé d'authentification */
int             key_len;        /* longueur de la clé d'authentification */
caddr_t         digest;         /* résumé de l'appelant à remplir */

{
    MD5_CTX context;
    unsigned char k_ipad[65];       /* bourrage interne - clé OUXée avec ipad */
    unsigned char k_opad[65];       /* bourrage externe - clé OUXée avec opad */
    unsigned char tk[16];
    int i;
    /* si la clé est plus longue que 64 octets la remettre à key=MD5(key) */
       si (key_len > 64) {
       MD5_CTX    tctx;
       MD5Init(&tctx);
       MD5Update(&tctx, key, key_len);
       MD5Final(tk, &tctx);
       key = tk;
       key_len = 16;
       }

       /*
       * la transformation HMAC_MD5 ressemble à ceci :
       *
       * MD5(K XOR opad, MD5(K XOR ipad, text))
       *
       * où K est une clé de n octets
       * ipad est l'octet 0x36 répété 64 fois
       * opad est l'octet 0x5c répété 64 fois
       * et text sont les données à protéger
       */

  /* commencer par mémoriser la clé dans le bourrage */
    bzero( k_ipad, sizeof k_ipad);
    bzero( k_opad, sizeof k_opad);
    bcopy( key, k_ipad, key_len);
```

```
    bcopy( key, k_opad, key_len);

  /* OUXer la clé avec les valeurs de ipad et opad */
    pour (i=0; i<64; i++) {
    k_ipad[i] ^= 0x36;
    k_opad[i] ^= 0x5c;
    }
    /*
    * effectuer le MD5 interne
    */
    MD5Init(&context);                  /* initier le contexte pour la première passe */
    MD5Update(&context, k_ipad, 64)     /* commencer par le bourrage interne */
    MD5Update(&context, text, text_len); /* puis le texte du datagramme */
    MD5Final(digest, &context);         /* finir la première passe */
    /*
    * effectuer le MD5 externe
    */
    MD5Init(&context);                  /* initier le contexte pour la 2ème passe */
    MD5Update(&context, k_opad, 64);    /* commencer par le bourrage externe */
    MD5Update(&context, digest, 16);    /* puis le résultat du premier hachage */
    MD5Final(digest, &context);         /* finir la 2ème passe*/
}
```

Vecteurs d'essai (Les '\0' de queue de la chaîne de caractères ne sont pas inclus dans l'essai) :

clé = 0x0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b0b
key_len = 16 octets
données = "Hi There"
data_len = 8 octets
résumé = 0x9294727a3638bb1c13f48ef8158bfc9d

clé = "Jefe"
données = "what do ya want for nothing?"
data_len = 28 octets
résumé = 0x750c783e6ab0b503eaa86e310a5db738

clé = 0xAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

key_len 16 octets
données = 0xDDDDDDDDDDDDDDDDDDDD...
..DDDDDDDDDDDDDDDDDDDD...
..DDDDDDDDDDDDDDDDDDDD...
..DDDDDDDDDDDDDDDDDDDD...
..DDDDDDDDDDDDDDDDDDDD
data_len = 50 octets
résumé = 0x56be34521d144c88dbb8c733f0e8b3f6

Remerciements

Pau-Chen Cheng, Jeff Kraemer, et Michael Oehler, ont fourni des commentaires utiles sur les premiers projets, et ont effectué les premiers essais d'interopérabilité de la présente spécification. Jeff et Pau-Chen ont gracieusement fourni le code d'échantillon et les vecteurs d'essai qui figurent dans l'appendice. Burt Kaliski, Bart Preneel, Matt Robshaw, Adi Shamir, et Paul van Oorschot ont fourni des commentaires et des suggestions utiles durant les investigations sur la construction de HMAC.

Références

[ANSI] ANSI X9.9, "American National Standard for Financial Institution Authentification de message (Wholesale)," American Bankers Association, 1981. Révisée en 1986.

[Atk] R. Atkinson, "En-tête d'authentification IP", RFC 1826, août 1995.

[BCK1] M. Bellare, R. Canetti, et H. Krawczyk, "Fonctions de hachage de clé et authentification de message", Proceedings of Crypto'96, LNCS 1109, pp. 1-15. (http://www.research.ibm.com/security/keyed-md5.html)

[BCK2] M. Bellare, R. Canetti, et H. Krawczyk, "Pseudorandom Functions Revisited: The Cascade Construction", Proceedings of FOCS'96.

[Dobb] H. Dobbertin, "The Status of MD5 After a Recent Attack", RSA Labs' CryptoBytes, Vol. 2 No. 2, Summer 1996. http://www.rsa.com/rsalabs/pubs/cryptobytes.html

[PV] B. Preneel et P. van Oorschot, "Building fast MACs from fonctions de hachage", Advances in Cryptology -- CRYPTO'95 Proceedings, Lecture Notes in Computer Science, Springer-Verlag Vol.963, 1995, pp. 1-14.

[MD5] R. Rivest, "Algorithme de résumé de message MD5", RFC 1321, avril 1992.

[MM] S. Meyer et S.M. Matyas, Cryptography, New York Wiley, 1982.

[RIPEMD] H. Dobbertin, A. Bosselaers, et B. Preneel, "RIPEMD-160: A strengthened version of RIPEMD", Fast Software Encryption, LNCS Vol 1039, pp. 71-82. ftp://ftp.esat.kuleuven.ac.be/pub/COSIC/bosselae/ripemd/ .

[SHA] NIST, FIPS PUB 180-1: Secure Hash Standard, avril 1995.

[Tsu] G. Tsudik, "Authentification de message avec fonctions de hachage unilatérales", Dans Proceedings of Infocom'92, mai 1992. (Aussi dans "Access Control and Policy Enforcement in Internetworks", thèse de doctorat, département d'informatique, University of Southern California, avril 1991.)

[VW] P. van Oorschot et M. Wiener, "Recherche de collision parallèle avec applications aux fonctions de hachage et logarithmes discrets", Proceedings of the 2nd ACM Conf. Computer et Communications Security, Fairfax, VA, novembre 1994.

Adresse des auteurs

Hugo Krawczyk
IBM T.J. Watson Research Center
P.O.Box 704
Yorktown Heights, NY 10598

mél : hugo@watson.ibm.com

Mihir Bellare
Dept of Computer Science et Engineering
Mail Code 0114
University of California at San Diego
9500 Gilman Drive
La Jolla, CA 92093
mél : mihir@cs.ucsd.edu

Ran Canetti
IBM T.J. Watson Research Center
P.O.Box 704
Yorktown Heights, NY 10598

mél : canetti@watson.ibm.com