Groupe de travail Réseau
Request for Comments : 4493
Catégorie : Information
Traduction Claude Brière de L'Isle

JH. Song, University of Washington
R. Poovendran, University of Washington
J. Lee, Samsung Electronics
T. Iwata, Nagoya University
juin 2006

Algorithme AES-CMAC

Statut de ce mémoire
Le présent mémoire apporte des informations pour la communauté de l'Internet. Le présent mémoire ne spécifie aucune sorte de norme de l'Internet. La distribution du présent mémoire n'est soumise à aucune restriction.

Résumé
L'Institut national des normes et technologies (NIST, *National Institute of Standards and Technology*) a récemment spécifié le code d'authentification de message fondé sur le chiffrement (CMAC, *Cipher-based Message Authentication Code*) qui est équivalent à OMAC1 (One-Key CBC MAC1) soumis par Iwata et Kurosawa. Le présent mémoire spécifie un algorithme d'authentification fondé sur CMAC avec la norme de chiffrement avancée (AES, *Advanced Encryption Standard*) de 128 bits. Ce nouvel algorithme d'authentification est appelé AES-CMAC. L'objet de ce document est de faire de l'algorithme AES-CMAC un algorithme facilement disponible pour la communauté de l'Internet.

Table des matières

1. Introduction

L'Institut national des normes et technologies (NIST, *National Institute of Standards and Technology*) a récemment spécifié le code d'authentification de message fondé sur le chiffrement (CMAC, *Cipher-based Message Authentication Code*). CMAC [NIST-CMAC] est une fonction de hachage chiffré qui se fonde sur un chiffrement de bloc à clés symétriques, comme la norme de chiffrement avancée [NIST-AES]. CMAC est équivalent au MAC1 CBC à une clé (OMAC1) soumis par Iwata et Kurosawa [OMAC1a], [OMAC1b]. OMAC1 est une amélioration du mode de chaînage de bloc de chiffrement étendu (XCBC, *eXtended Cipher Block Chaining*) soumis par Black et Rogaway [XCBCa], [XCBCb], qui lui-même est une amélioration du code d'authentification de message à chaînage de bloc de chiffrement (CBC-MAC, *Cipher Block Chaining-Message Authentication Code)* de base. XCBC traite efficacement les déficiences de sécurité de CBC-MAC, et OMAC1 réduit efficacement la taille de clé de XCBC.

AES-CMAC fournit une plus forte assurance d'intégrité des données qu'une somme de contrôle ou un code de détection d'erreur. La vérification d'une somme de contrôle ou d'un code de détection d'erreurs ne détecte que les modifications accidentelles des données, tandis que CMAC est conçu pour détecter les modifications intentionnelles non autorisées des données, ainsi que les modifications accidentelles.

AES-CMAC réalise des objectifs de sécurité similaires à ceux de HMAC [RFC2104]. Comme AES-CMAC se fonde sur un

chiffrement de bloc à clés symétriques, AES, et que HMAC se fonde sur une fonction de hachage, comme SHA-1, AES-CMAC est approprié pour les systèmes d'informations dans lesquels AES est plus directement disponible qu'une fonction de hachage.

Le présent mémoire spécifie l'algorithme d'authentification fondé sur CMAC avec AES-128. Ce nouvel algorithme d'authentification est appelé AES-CMAC.

2. Spécification de AES-CMAC

2.1 Définitions de base

Le tableau suivant décrit les définitions de base nécessaires pour expliquer la spécification de AES-CMAC.

x || y : enchaînement. x || y est la chaîne x enchaînée avec la chaîne y. 0000 || 1111 est 00001111.

x OUX y : opération OU exclusif. Pour deux chaînes de longueur égale , x et y, x OUX y est leur OU exclusif au bit près.

ceil(x) : fonction plafond. Le plus petit entier non inférieur à x. ceil(3.5) est 4. ceil(5) est 5.

x << 1 : glissement à gauche de la chaîne x d'un bit. Le bit de poids fort disparaît et un zéro vient au bit de moindre poids. 10010001 << 1 est 00100010.

0^n : chaîne qui consiste en n bits de zéro. 0^3 signifie 000 en format binaire. 10^4 signifie 10000 en format binaire. 10^i signifie 1 suivi par des zéros répétés i fois.

MSB(x) : le bit de poids fort de la chaîne x. MSB(10010000) signifie 1.

bourrage(x) : résultat bourré de 10^i de l'entrée x. Il est décrit en détail au paragraphe 2.4.

Clé : clé longue de 128 bits (16 octets) pour AES-128. Notée par K.

Première sous clé : première sous clé longue de 128 bits (16 octets) déduite par l'algorithme de génération de sous clé de la clé K. Notée K1.

Seconde sous clé : seconde sous clé longue de 128 bits (16 octets) déduite par l'algorithme de génération de sous clé de la clé K. Notée K2.

Message : un message à authentifier. Noté M. Le message peut être nul, ce qui signifie que la longueur de M est 0.

Longueur de message : la longueur du message M en octets. Notée len. La valeur minimum de la longueur peut être 0. La valeur maximum de la longueur n'est pas spécifiée dans ce document.

AES-128(K,M) : AES-128(K,M) est le texte chiffré de 128 bits de AES-128 pour une clé de 128 bits, K, et un message M de 128 bits.

MAC : chaîne de 128 bits qui est le résultat de AES-CMAC. Notée T. Valider le MAC donne l'assurance de l'intégrité et de l'authenticité du message provenant de la source.

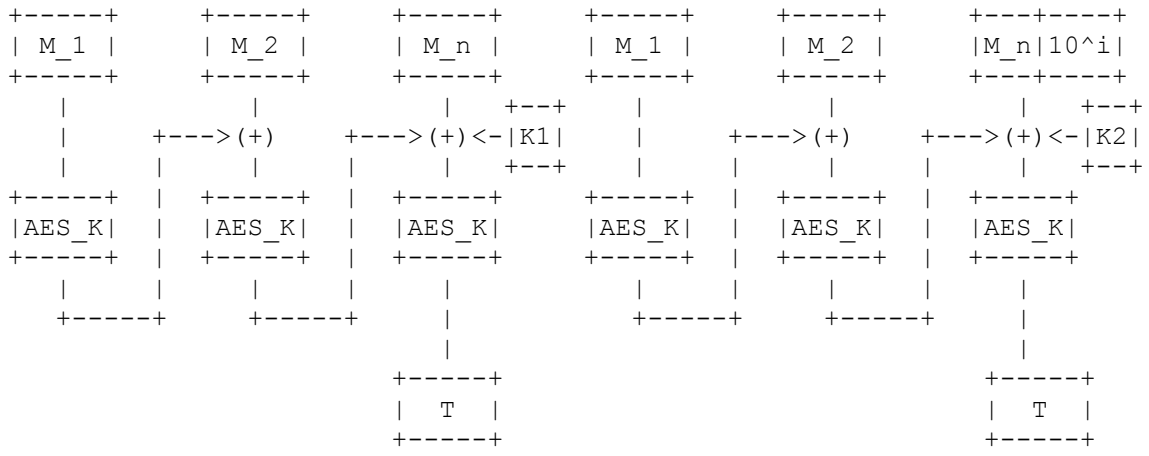
Longueur de MAC : par défaut, la longueur du résultat de AES-CMAC est 128 bits. Il est possible de tronquer le MAC. Le résultat de la troncature devrait être pris dans l'ordre des bits de poids fort en premier. La longueur de MAC doit être spécifiée avant le début de la communication, et elle ne doit pas être changée pendant le durée de vie de la clé.

2.2 Vue d'ensemble

AES-CMAC utilise la norme de chiffrement évoluée [NIST-AES] comme bloc de construction. Pour générer un MAC, AES-CMAC prend une clé secrète, un message de longueur variable, et la longueur du message en octets comme entrées et retourne une chaîne de bits fixe appelée un MAC.

Le cœur de AES-CMAC est le CBC-MAC de base. Pour qu'un message, M, soit authentifié, le CBC-MAC est appliqué à

M. Il y a deux cas de fonctionnement dans CMAC. La Figure 2.1 illustre le fonctionnement de CBC-MAC dans les deux cas. Si la taille du bloc d'entrée de message est égale à un multiple positif de la taille de bloc (à savoir, 128 bits) le dernier bloc devra être OUXé avec K1 avant le traitement. Autrement, le dernier bloc devra être bourré avec 10^i (la notation est décrite au paragraphe 2.1) et OUXé avec K2. Le résultat du traitement précédent sera l'entrée du dernier chiffrement. Le résultat de AES-CMAC assure l'intégrité des données de tout le message d'entrée.



(a) longueur positive de blocs multiples (b) autrement

Figure 2.1. Illustration des deux cas de AES-CMAC

AES_K est AES-128 avec la clé K.
Le message M est divisé en blocs M_1,...,M_n, où M_i est le ième bloc du message.
La longueur de M_i est 128 bits pour i = 1,...,n-1, et la longueur du dernier bloc, M_n, est inférieure ou égale à 128 bits.
K1 est la sous clé pour le cas (a),
K2 est la sous clé pour le cas (b).
K1 et K2 sont générés par l'algorithme de génération de sous clés décrit au paragraphe 2.3.

2.3 Algorithme de génération de sous clé

L'algorithme de génération de sous clé, Generate_Subkey(), prend une clé secrète, K, qui est juste la clé pour AES-128.

Le résultat de l'algorithme de génération de sous clé est deux sous clés, K1 et K2.
On écrit (K1,K2) := Generate_Subkey(K).

Les sous clés K1 et K2 sont utilisées dans les deux algorithmes de génération de MAC et de vérification de MAC. K1 est utilisée dans le cas où la longueur du dernier bloc est égale à la longueur de bloc. K2 est utilisée dans le cas où la longueur du dernier bloc est inférieure à la longueur de bloc.

La Figure 2.2 spécifie l'algorithme de génération de sous clé.

Figure 2.2. Algorithme Generate_Subkey :

Entrée : K (clé de 128 bits)
Résultat : K1 (sous clé de 128 bits)
K2 (seconde sous clé de 128 bits)
Constantes : const_Zero est 0x00000000000000000000000000000000
const_Rb est 0x00000000000000000000000000000087
Variables : L pour les résultat de AES-128 appliqué à 0^128

Étape 1. L := AES-128(K, const_Zero) ;
Étape 2. Si MSB(L) est égal à 0 alors K1 := L << 1 ;
autrement K1 := (L << 1) OUX const_Rb ;
Étape 3. Si MSB(K1) est égal à 0 alors K2 := K1 << 1 ;
autrement K2 := (K1 << 1) OUX const_Rb ;
Étape 4. Retourner K1, K2 ;

Dans l'étape 1, AES-128 avec la clé K est appliqué à un bloc d'entrée tout de zéros.

Dans l'étape 2, K1 est déduite par l'opération suivante :
Si le bit de poids fort est égal à 0, K1 est le décalage à gauche de 1 bit de L.
Autrement, K1 est le OU exclusif de const_Rb et du décalage à gauche de 1 bit de L.

Dans l'étape 3, K2 est déduite par l'opération suivante :
Si le bit de poids fort de K1 est égal à 0, K2 est le décalage à gauche de 1 bit de K1.
Autrement, K2 est le OU exclusif de const_Rb et du décalage à gauche de 1 bit de K1.

Dans l'étape 4, (K1,K2) := Generate_Subkey(K) est retourné.

La signification mathématique des procédures des étapes 2 et 3, incluant const_Rb, se trouve dans [OMAC1a].

2.4 Algorithme de génération de MAC

L'algorithme de génération de MAC, AES-CMAC(), prend trois entrées, une clé secrète, un message, et la longueur du message en octets. La clé secrète, notée K, est juste la clé pour AES-128. Le message et sa longueur en octets sont notés respectivement M et len. Le message M est noté par la séquence de M_i, où M_i est le ième bloc de message. C'est-à-dire que si M comporte n blocs, alors M s'écrit : M = M_1 || M_2 || ... || M_{n-1} || M_n

La longueur de M_i est de 128 bits pour i = 1,...,n-1, et la longueur du dernier bloc M_n est inférieure ou égale à 128 bits.

Le résultat de l'algorithme de génération de MAC est une chaîne de 128 bits, appelée un MAC, qui est utilisée pour valider le message d'entrée. Le MAC est noté T, et on écrit T := AES-CMAC(K,M,len). La validation du MAC donne l'assurance de l'intégrité et de l'authenticité du message provenant de la source.

Il est possible de tronquer le MAC. Conformément à [NIST-CMAC], un MAC d'au moins 64 bits devrait être utilisé comme protection contre les attaques qui cherchent à le deviner. Le résultat de la troncature devrait être pris dans l'ordre des bits de poids fort en premier.

La longueur de bloc de AES-128 est 128 bits (16 octets). Il y a un traitement spécial si la longueur du message n'est pas un multiple positif de la longueur de bloc. Le traitement spécial est de bourrer M avec la chaîne binaire 10^i pour ajuster la longueur du dernier bloc sur la longueur de bloc.

Pour une chaîne d'entrée x de r octets, où $0 \le r < 16$, la fonction de bourrage, bourrage(x), est définie comme suit :

- bourrage(x) = x || 10^i où i est 128-8*r-1

C'est-à-dire que bourrage(x) est l'enchaînement de x et d'un seul '1', suivi par le nombre minimum de "0", de telle sorte que la longueur totale soit égale à 128 bits.

La Figure 2.3 décrit l'algorithme de génération de MAC.

Figure 2.3. Algorithme AES-CMAC

```
Entrées :   K (clé de 128 bits)
            M (message à authentifier)
            len (longueur du message en octets)
Résultat : T (code d'authentification de message)
Constantes :   const_Zero est 0x00000000000000000000000000000000
               const_Bsize est 16
Variables :    K1, K2 pour les sous clés de 128 bits
               M_i est le ième bloc (i=1..ceil(len/const_Bsize))
               M_last est le dernier bloc OUXé avec K1 ou K2
               n pour le nombre de blocs à traiter
               r pour le nombre d'octets du dernier bloc
               fanion pour noter si le dernier bloc est complet ou non

Étape 1. (K1,K2) := Generate_Subkey(K) ;
Étape 2.  n := ceil(len/const_Bsize) ;
```

```
Étape 3. Si n = 0  alors n := 1 ; fanion := faux ;
          Autrement si len mod const_Bsize est 0 alors fanion := vrai ;
                       autrement fanion := faux ;
Étape 4. Si fanion est vrai alors M_last := M_n OUX K1 ;
         Autrement M_last := bourrage(M_n) OUX K2 ;
Étape 5. X := const_Zero ;
Étape 6. Pour i := 1 à n-1 faire
              début
                  Y := X OUX M_i ;
                  X := AES-128(K,Y) ;
               fin
             Y := M_last OUX X ;
             T := AES-128(K,Y) ;
Étape 7.  retourne T ;
```

Dans l'étape 1, les sous clés K1 et K2 sont déduites de K par l'algorithme de génération de sous clés.

Dans l'étape 2, on calcule le nombre de blocs, n. Le nombre de blocs est la plus petite valeur d'entier supérieure ou égale au quotient déterminé par la division du paramètre de longueur par la longueur de bloc, 16 octets.

Dans l'étape 3, la longueur du message d'entrée est vérifiée. Si la longueur d'entrée est 0 (nulle) le nombre de blocs à traiter devra être 1, et le fanion devra être marqué comme bloc incomplet (faux). Autrement, si la longueur du dernier bloc est 128 bits, le fanion est marqué comme bloc complet (vrai) ; autrement on marque le fanion comme bloc incomplet (faux).

Dans l'étape 4, M_last est calculé en OUXant M_n avec une des sous clés calculées précédemment. Si le dernier bloc est complet (vrai), alors M_last est le OU exclusif de M_n a et K1. Autrement, M_last est le OU exclusif de bourrage(M_n) et de K2. Dans l'étape 5, la variable X est initialisée.

Dans l'étape 6, le CBC-MAC de base est appliqué à M_1,...,M_{n-1},M_last.

Dans l'étape 7, le MAC de 128 bits, T := AES-CMAC(K,M,len) est retourné.

Si nécessaire, le MAC est tronqué avant d'être retourné.

2.5 Algorithme de vérification de MAC

La vérification du MAC est simplement faite en le recalculant. On utilise l'algorithme de génération de MAC qui est décrit au paragraphe 2.4.

L'algorithme de vérification de MAC, Verify_MAC(), prend quatre entrées, une clé secrète, un message, la longueur du message en octets, et le MAC reçu. Ils sont notés respectivement K, M, len, et T'.

Le résultat de l' algorithme de vérification de MAC est INVALIDE ou VALIDE.

La Figure 2.4 décrit l'algorithme de vérification de MAC.

```
Entrées : K  (clé de 128 bits)
          M (message à vérifier)
          len ( longueur du message en octets)
          T' (MAC reçu à vérifier)
Résultat : INVALIDE ou VALIDE

Étape 1. T* := AES-CMAC(K,M,len) ;
Étape 2. Si T* est égal à T' alors retourner VALIDE ;
                               autrement retourner INVALIDE.
```

Figure 2.4. Algorithme Verify_MAC

Dans l'étape 1, T* est déduit de K, M, et len par l'algorithme de génération de MAC.

Dans l'étape 2, on compare T* et T. Si T* est égal à T', on retourne VALIDE ; autrement, on retourne INVALIDE.

Si le résultat est INVALIDE, alors le message est prouvé non authentique, c'est-à-dire, il ne provient pas de la source qui a exécuté le processus de génération sur le message pour produire le MAC présenté.

Si le résultat est VALIDE, alors la conception de AES-CMAC donne l'assurance que le message est authentique et donc, n'a pas été corrompu dans le transit ; cependant, cette assurance, comme pour tout algorithme de MAC, n'est pas absolue.

3. Considérations sur la sécurité

La sécurité procurée par AES-CMAC s'appuie sur la force de l'algorithme de chiffrement AES. Cependant, comme c'est vrai pour tous les algorithme de chiffrement, une partie de sa force repose dans la clé secrète, K, et sur la correction de la mise en œuvre dans tous les systèmes participants. Si la clé secrète est compromise ou partagée de façon inappropriée, cela ne garantit pas du tout l'authentification ni l'intégrité du message. La clé secrète doit être générée d'une façon qui satisfasse les exigences de pseudo aléa de la [RFC4086] et devrait être conservée en toute sécurité. C'est seulement si et seulement si AES-CMAC est utilisé de façon appropriée qu'il assure l'authentification et l'intégrité qui satisfont aux bonnes pratiques actuelles de l'authentification de message.

4. Vecteurs d'essai

Les vecteurs d'essai suivants sont les mêmes que ceux de [NIST-CMAC]. Ils sont aussi le résultat du programme d'essai de l'Appendice A.

```
  ---------------------------------------------------
Génération de sous clé
K            2b7e1516 28aed2a6 abf71588 09cf4f3c
AES-128(key,0) 7df76b0c 1ab899b3 3e42f047 b91b546f
K1           fbeed618 35713366 7c85e08f 7236a8de
K2           f7ddac30 6ae266cc f90bc11e e46d513b
---------------------------------------------------

Exemple 1 : len = 0
M            <chaîne vide>
AES-CMAC     bb1d6929 e9593728 7fa37d12 9b756746
---------------------------------------------------

Exemple 2 : len = 16
M            6bc1bee2 2e409f96 e93d7e11 7393172a
AES-CMAC     070a16b4 6b4d4144 f79bdd9d d04a287c
---------------------------------------------------

Exemple 3 : len = 40
M            6bc1bee2 2e409f96 e93d7e11 7393172a ae2d8a57 1e03ac9c 9eb76fac 45af8e51 30c81c46 a35ce411
AES-CMAC     dfa66747 de9ae630 30ca3261 1497c827
---------------------------------------------------

Exemple 4 : len = 64
M            6bc1bee2 2e409f96 e93d7e11 7393172a ae2d8a57 1e03ac9c 9eb76fac 45af8e51
             30c81c46 a35ce411 e5fbc119 1a0a52ef f69f2445 df4f9b17 ad2b417b e66c3710
AES-CMAC     51f0bebf 7e3b9d92 fc497417 79363cfe
---------------------------------------------------
```

5. Remerciements

Des portions du texte présenté ici sont empruntées à[NIST-CMAC]. On remercie les auteurs de OMAC1, l'auteur de SP

800-38B, et Russ Housley des ses utiles commentaires et conseils, qui ont été incorporés ici. On remercie aussi Alfred Hoenes de ses nombreux commentaires utiles. Ce mémoire a été préparé quand Tetsu Iwata était à l'Université Ibaraki au Japon.

On remercie de leur soutien les organismes suivants : Collaborative Technology Alliance (CTA) du US Army Research Laboratory, DAAD19- 01-2-0011 ; Presidential Award from Army Research Office, W911NF-05-1-0491; NSF CAREER ANI-0093187. Les résultats ne reflètent pas nécessairement la position de ces agences de financement.

6. Références

[NIST-AES] NIST, FIPS 197, "Advanced Encryption Standard (AES)", novembre 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[NIST-CMAC] NIST, Special Publication 800-38B, "Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication", mai 2005.

[OMAC1a] Tetsu Iwata and Kaoru Kurosawa, "OMAC: One-Key CBC MAC", Fast Software Encryption, FSE 2003, LNCS 2887, pp. 129-153, Springer-Verlag, 2003.

[OMAC1b] Tetsu Iwata and Kaoru Kurosawa, "OMAC: One-Key CBC MAC", Submission to NIST, décembre 2002. Disponible à http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/omac/omac-spec.pdf

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

[RFC4086] D. Eastlake 3rd, J. Schiller, S. Crocker, "Exigences d'aléa pour la sécurité", juin 2005. (*Remplace* RFC1750) (BCP0106)

[XCBCa] John Black and Phillip Rogaway, "A Suggestion for Handling Arbitrary-Length Messages with the CBC MAC", NIST Second Modes of Operation Workshop, août 2001. Disponible à http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/xcbc-mac/xcbc-mac-spec.pdf

[XCBCb] John Black and Phillip Rogaway, "CBC MACs for Arbitrary-Length Messages: The Three-Key Constructions", Journal of Cryptology, Vol. 18, No. 2, pp. 111-132, Springer-Verlag, Spring 2005.

Appendice A. Code d'essai

Cette source en langage C est destinée à générer les vecteurs d'essai qui apparaissent dans le présent mémoire pour vérifier la correction de l'algorithme. Le code source n'est pas destiné à l'utilisation dans des produits commerciaux.

```
/*****************************************************************/
/* AES-CMAC avec AES à 128 bits*/
/* CMAC : Algorithme décrit dans SP800-38B*/
/* Auteurs : Junhyuk Song (junhyuk.song@samsung.com)          */
/*           Jicheol Lee  (jicheol.lee@samsung.com)           */
 /*****************************************************************/

#inclut <stdio.h>

/* Pour le calcul de CMAC */
 unsigned char const_Rb[16] = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x87
 };
 unsigned char const_Zero[16] = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
 };
```

```
/* Fonctions de base */

 void xor_128(unsigned char *a, unsigned char *b, unsigned char *out)
 {
    int i;
    for (i=0;i<16; i++)
    {
       out[i] = a[i] ^ b[i];
    }
 }

 void print_hex(char *str, unsigned char *buf, int len)
 {
    int     i;

    for ( i=0; i<len; i++ ) {
       if ( (i % 16) == 0 && i != 0 ) printf(str);
       printf("%02x", buf[i]);
       if ( (i % 4) == 3 ) printf(" ");
       if ( (i % 16) == 15 ) printf("\n");
    }
    if ( (i % 16) != 0 ) printf("\n");
 }

 void print128(unsigned char *bytes)
 {
    int         j;
    for (j=0; j<16;j++) {
       printf("%02x",bytes[j]);
       if ( (j%4) == 3 ) printf(" ");
    }
 }

 void print96(unsigned char *bytes)
 {
    int         j;
    for (j=0; j<12;j++) {
       printf("%02x",bytes[j]);
       if ( (j%4) == 3 ) printf(" ");
    }
 }

/* Fonction de génération AES-CMAC */

 void leftshift_onebit(unsigned char *input,unsigned char *output)
 {
    int         i;
    unsigned char overflow = 0;

    for ( i=15; i>=0; i-- ) {
       output[i] = input[i] << 1;
       output[i] |= overflow;
       overflow = (input[i] & 0x80)?1:0;
    }
    return;
 }

 void generate_subkey(unsigned char *key, unsigned char *K1, unsigned char *K2)
 {
    unsigned char L[16];
```

```
    unsigned char Z[16];
    unsigned char tmp[16];
    int i;

    for ( i=0; i<16; i++ ) Z[i] = 0;

    AES_128(key,Z,L);

    if ( (L[0] & 0x80) == 0 ) { /* If MSB(L) = 0, then K1 = L << 1 */
       leftshift_onebit(L,K1);
    } else {    /* Else K1 = ( L << 1 ) (+) Rb */

       leftshift_onebit(L,tmp);
       xor_128(tmp,const_Rb,K1);
    }

    if ( (K1[0] & 0x80) == 0 ) {
       leftshift_onebit(K1,K2);
    } else {
       leftshift_onebit(K1,tmp);
       xor_128(tmp,const_Rb,K2);
    }
    return;
 }

 void padding ( unsigned char *lastb, unsigned char *pad, int length )
 {
    int        j;

/* dernier bloc d'origine */
    for ( j=0; j<16; j++ ) {
       if ( j < length ) {
          pad[j] = lastb[j];
       } else if ( j == length ) {
          pad[j] = 0x80;
       } else {
          pad[j] = 0x00;
       }
    }
 }

 void AES_CMAC ( unsigned char *key, unsigned char *input, int length, unsigned char *mac )
 {
    unsigned char      X[16],Y[16], M_last[16], padded[16];
    unsigned char      K1[16], K2[16];
    int        n, i, flag;
    generate_subkey(key,K1,K2);

    n = (length+15) / 16;                          /* n est le nombre de tours */

    if ( n == 0 ) {
       n = 1;
       flag = 0;
    } else {
       if ( (length%16) == 0 ) {                    /* le dernier bloc est complet */
          flag = 1;
       } else {                                    /* le dernier bloc n'est pas complet */
          flag = 0;
       }

    }
```

```
    if ( flag ) {                                   /* le dernier bloc est complet */
      xor_128(&input[16*(n-1)],K1,M_last);
    } else {
      padding(&input[16*(n-1)],padded,length%16);
      xor_128(padded,K2,M_last);
    }

    for ( i=0; i<16; i++ ) X[i] = 0;
    for ( i=0; i<n-1; i++ ) {
      xor_128(X,&input[16*i],Y);                    /* Y := Mi (+) X  */
      AES_128(key,Y,X);                             /* X := AES-128(KEY, Y); */
    }

    xor_128(X,M_last,Y);
    AES_128(key,Y,X);

    for ( i=0; i<16; i++ ) {
      mac[i] = X[i];
    }
}

int main()
{
    unsigned char L[16], K1[16], K2[16], T[16], TT[12];
    unsigned char M[64] = {
      0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96, 0xe9, 0x3d, 0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a,
      0xae, 0x2d, 0x8a, 0x57, 0x1e, 0x03, 0xac, 0x9c, 0x9e, 0xb7, 0x6f, 0xac, 0x45, 0xaf, 0x8e, 0x51,
      0x30, 0xc8, 0x1c, 0x46, 0xa3, 0x5c, 0xe4, 0x11, 0xe5, 0xfb, 0xc1, 0x19, 0x1a, 0x0a, 0x52, 0xef,
      0xf6, 0x9f, 0x24, 0x45, 0xdf, 0x4f, 0x9b, 0x17, 0xad, 0x2b, 0x41, 0x7b, 0xe6, 0x6c, 0x37, 0x10
    };
    unsigned char key[16] = {
      0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab, 0xf7, 0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c
    };

    printf("---------------------------------------------------\n");
    printf("K             "); print128(key); printf("\n");

    printf("\nSubkey Generation\n");
    AES_128(key,const_Zero,L);
    printf("AES_128(key,0) "); print128(L); printf("\n");
    generate_subkey(key,K1,K2);

    printf("K1            "); print128(K1); printf("\n");
    printf("K2            "); print128(K2); printf("\n");

    printf("\nExample 1: len = 0\n");
    printf("M             "); printf("<empty string>\n");

    AES_CMAC(key,M,0,T);
    printf("AES_CMAC      "); print128(T); printf("\n");

    printf("\nExample 2: len = 16\n");
    printf("M             "); print_hex("               ",M,16);
    AES_CMAC(key,M,16,T);
    printf("AES_CMAC      "); print128(T); printf("\n");
    printf("\nExample 3: len = 40\n");
    printf("M             "); print_hex("               ",M,40);
    AES_CMAC(key,M,40,T);
    printf("AES_CMAC      "); print128(T); printf("\n");
```

```
    printf("\nExample 4: len = 64\n");
    printf("M              "); print_hex("              ",M,64);
    AES_CMAC(key,M,64,T);
    printf("AES_CMAC       "); print128(T); printf("\n");

    printf("------------------------------------------------\n");

    return 0;
}
```

Adresse des auteurs

Junhyuk Song
University of Washington
Samsung Electronics
USA
téléphone : (206) 853-5843
mél : junhyuk.song@samsung.com

Jicheol Lee
Samsung Electronics
téléphone : +82-31-279-3605
mél : jicheol.lee@samsung.com

Radha Poovendran
Network Security Lab
University of Washington
USA
téléphone : (206) 221-6512
mél : radha@ee.washington.edu

Tetsu Iwata
Nagoya University
mél : iwata@cse.nagoya-u.ac.jp

Déclaration complète de droits de reproduction

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